ВТУ "СВ. СВ. КИРИЛ И МЕТОДИЙ"
ПЕДАГОГИЧЕСКИ ФАКУЛТЕТ
КАТЕДРА "АЛГЕБРА И ГЕОМЕТРИЯ"

Валентин Пенев Бакоев

АЛГОРИТМИ ЗА ИЗСЛЕДВАНЕ НА КОМБИНАТОРНИ ОБЕКТИ

Д И С Е Р Т А Ц И Я

за присъждане на образователната и научна степен
"доктор"

Научна специалност: 01.01.12 – Информатика

Научен консултант: доц. д-р Красимир Манев

В. Търново, 2003

Съдържание

Увод

Проблематика, обект на изследване, цели и задачи

Възникването и бурното развитие на компютърната техника и софтуерните технологии и повсеместното им приложение във всички сфери на човешката дейност е неразривно свързано с развитието и достиженията в областта на теоретичната информатика. Безспорно важно място в нея заема дискретната математика. Тя обединява в едно направление редица математически теории, чиито граници не могат да бъдат точно очертани. Причината за това е, че едни и същи дискретни обекти (например множества, релации, функции) са в основата на различни теории. Тези или други дискретни обекти и техните свойства много често се изучават и ползват от различни гледни точки, т. е. в рамките на специфичния за съответната математическа теория апарат от понятия, твърдения, подходи, методи и др. Редица дискретни задачи често се решават в рамките на различни дисциплини – например много задачи от теория на булевите функции се свеждат до или имат пряка интерпретация в задачи от други области като теория на графите, комбинаторика, теория на алгоритмите, теория на кодирането, теория на веригите, изкуствения интелект и др. [24, 39, 42, 55]. Въпреки забележителните успехи в областта на дискретната математика през миналия век и дългогодишните усилия на много изследователи редица важни дискретни задачи все още остават нерешени, други имат само частични решения, а при трети остава открит въпросът за сложността им. Класическите математически теории се оказват недостатъчно мощни за пълното им решаване. Теорията и практиката непрекъснато поставят нови подобни задачи. Но Хилберт казва, че "всяка научна област е жизнеспособна докато в нея има излишък от нови задачи. Недостигът на нови задачи означава отмиране или прекратяване на самостоятелното ѝ развитие" [16].

Спецификата на много от отворените задачи в дискретната математика е такава, че за тях се търсят предимно (или като начало) алгоритмични решения. Съществуващите алгоритми, които ги решават (напълно или частично), се характеризират с това, че времето им за изпълнение нараства експоненциално при увеличаване големината на входа, т. е. те са типични представители на т. нар. "труднорешими задачи" [8]. **Обект на изследване** в настоящата дисертация са няколко важни такива задачи, формулирани в трета, четвърта и пета глава. Те се анализират с цел да бъдат разработени по-ефективни алгоритми и най-вече такива с полиномиална времева сложност, които да решават възможно най-много екземпляри на тези задачи. **Основната идея**, която следваме при всички

изследвания, произтича от мисълта на Гари и Джонсън [8], че "полиномиален алгоритъм обикновено може да бъде построен само тогава, когато успеем да проникнем по-дълбоко в същността на решаваната задача" чрез математически методи и средства. Тази мисъл обуславя следната по-точна формулировка на **целите и задачите** на настоящата работа: чрез прилагане на алгоритмичен подход, съчетан с комбинаторни методи и средства, да изследваме математическата структура на тези задачи; да изведем, да опишем формално и докажем характерни нейни свойства, на базата на които да построим по-ефективни алгоритми. После, анализирайки резултатите от изпълнението им, да изведем нови свойства, чрез тях нови алгоритми и т. н.

Методологията на изследване в дисертацията не е нова, но се прилага успешно от много изследователи. Накратко можем да я формулираме като подпомагане решаването на математически задачи чрез компютърни програми. Основният акцент в работата пада върху разработването на алгоритми, резултатите от които да подсказват и подпомагат решаването на задачите, след което получените резултати да бъдат анализирани, проверени и доказани математически.

Представените тук алгоритми са реализирани и проверени чрез компютърни програми, които са малка част от всички разработени програми. До получаването на окончателните версии бяха направени редица експерименти и подобрения с цел постигане на желаните резултати и ефективност. Друга част от програмите (непредставени тук) имат спомагателна роля – за проверка коректността на основните и на алтернативните алгоритми, за сравняване на резултатите от работата им и др. Всички програми са написани на езика Паскал, с изключение на няколко версии на алгоритъма Gen, написани на C++. Мотивите за избора на езика са няколко. В съответствие с основната идея и целите подобряването на ефективността на алгоритмите е търсено чрез по-дълбоко проникване в същността на разглежданите задачи, чрез извеждане на характерни свойства на тяхната структура, които да бъдат отчетени и заложени в алгоритмите. При равни други условия разликите във възможностите на езиците за програмиране са минимални и не водят до значителна промяна във времето за изпълнение на даден алгоритъм. Друг аргумент при избора на език е, че в повечето случаи алгоритмите се публикуват в Паскалоподобен код, който е лесно разбираем. Средите за програмиране на Паскал имат удобни средства за тестване, трасировка и настройка.

Преглед на съдържанието

Освен настоящия увод, дисертацията включва пет глави, заключение, съдържание и списък с цитирана литература и адреси в Интернет. Проучванията се отнасят за конкретни и твърде различни помежду си задачи и изследвания. Поради това те не са обединени, а са дадени поотделно в трета, четвърта и пета глава. Първа и втора глава съдържат основните понятия и твърдения, необходими за останалата част от изложението.

Резюме на първа глава

В първите два раздела на първа глава са представени основни дискретни обекти като азбуки, думи, езици, n-мерен булев куб $\{0,1\}^n$. В следващите три раздела на тази глава са дадени основни дефиниции и твърдения, свързани с булеви функции, които се ползват в следващите глави. Изложението е според учебника [16], като в допълнение са използвани и други известни български и чуждестранни учебници [9, 20, 1, 10, 65, 48].

Резюме на втора глава

Втора глава представя елементи от теория на алгоритмите. В първи раздел се въвеждат и обсъждат понятията масова задача, екземпляр, алгоритъм, алгоритмична разрешимост. Във втори раздел се дефинират машините на Тюринг и се разглеждат изчисленията, които те извършват. Чрез тях формално се определя понятието алгоритъм, споменати са и други еквивалентни на тях по изчислителна мощ формализми. В трети раздел се дефинират различните видове сложности по време и памет чрез модела машини на Тюринг, както и чрез още два модела на абстрактен компютър, наречени RAM и RASP. Четвърти раздел е посветен на асимптотичните означения и порядъците на растеж при функциите на сложност. В пети и шести раздел се дефинират и обсъждат трите основни класа на сложност: $\mathcal{P}$ – на полиномиално-разрешимите задачи, $\mathcal{NP}$ – на полиномиално-проверимите задачи и $\mathcal{NP}$-c – на $\mathcal{NP}$-пълните задачи. Последните са известни като "най-трудните" задачи в класа $\mathcal{NP}$. Характерно за тях е това, че дадена задача от този клас е поне толкова трудна, колкото е трудна всяка друга задача от класа $\mathcal{NP}$-c, т. е. сложностите им са полиномиално свързани. Изложението в тази глава е според учебника [16], както и според получилите класическа популярност [8, 21, 37].

Резюме на трета глава

В трета глава са представени изследванията ни върху задачите:

– *удовлетворимост на булева функция* (УБФ): "Дадена е булевата функция $f(x_1, x_2, \ldots, x_n)$ в конюнктивна нормална форма. Съществуват ли значения на променливите $\sigma_1, \sigma_2, \ldots, \sigma_n$, такива че $f(\sigma_1, \sigma_2, \ldots, \sigma_n) = 1$?";

– *3-удовлетворимост на булева функция* (3-УБФ): "Дадена е булевата функция $f(x_1, x_2, \ldots, x_n)$ в конюнктивна нормална форма, като всяка елементарна дизюнкция съдържа точно по три букви на променливи (с или без отрицания). Съществуват ли значения на променливите $\sigma_1, \sigma_2, \ldots, \sigma_n$, такива че $f(\sigma_1, \sigma_2, \ldots, \sigma_n) = 1$?".

Задачите УБФ и нейната разновидност 3-УБФ, наричани накратко задачи за удовлетворимост (ЗУ), са най-важните в теорията на $\mathcal{NP}$-пълните задачи. Те са основата, върху която тя е построена. Фундаменталният въпрос "Дали $\mathcal{P} \neq \mathcal{NP}$?" все още не е получил еднозначен отговор. ЗУ

са обект на все по-силен интерес и множество изследвания в целия свят и особено в Германия.

Първи раздел на трета глава представлява кратък обзор на изследванията върху ЗУ. Представени са вариантите на ЗУ, направленията, подходите, моделите и методите на изследване, по-важните постигнати резултати.

Във втори раздел, вместо задачата УБФ, се разглежда еквивалентната ѝ задача за неудовлетворимост: "Дали дадена конюнктивна нормална форма на булевата функция f е форма на константата нула?". Изследва се ограничен вариант на тази задача, в който функциите са на не повече от три букви на променливи. С помощта на компютърна програма беше установено, че от възможните 67 108 864 конюнктивни нормални форми (КНФ) на функции на не повече от три променливи 63 367 025 (или $\approx 94,424\%$) са форми на константата нула (означаваме я с $\tilde{0}$). Тези форми се класифицират спрямо еквивалентности, които дефинираме в общия случай – над множеството от всички формули в КНФ, съдържащи до n букви на променливи, без ограничения за дължината и броя на елементарните конюнкции (клаузите). Някои свойства на клаузите от това множество са дадени в Леми 3.8, 3.9, 3.10 и 3.11. С тяхна помощ е доказано основното твърдение в раздела – Теорема 3.12, която дава класификация на нееквивалентните помежду си форми на $\tilde{0}$ с до три букви на променливи (с или без отрицания). Тази класификация е подпомогната и проверена чрез компютърни програми. Резултатите в този раздел са получени съвместно с доц. д-р Красимир Манев. Те са докладвани на Петата международна конференция по дискретна математика и приложения (1995 г.) и на XXVI Пролетна конференция на СМБ (1997 г.), публикувани са в [56, 25]. Резултатите в този и в следващия раздел са получени самостоятелно и независимо от тези в някои от цитираните източници, въпреки че съвпадат частично с резултатите на други изследователи, публикувани по-късно.

Изследванията, представени в трети раздел, са върху оригиналната задача 3-УБФ и съответната ѝ задача за неудовлетворимост. Множеството S от всевъзможните 3-буквени клаузи над променливите $\{x_1, x_2, \ldots, x_n\}$ се разбива на четири подмножества S_0, S_1, S_2, S_3, съдържащи всички клаузи съответно с $0, 1, 2, 3$ отрицания. Последната форма от твърдението на Теорема 3.12 се определя като "минимална основна форма" на $\tilde{0}$. Представени са разбиванията на множеството $S = S_0 \cup \ldots \cup S_3$ на минимални основни форми, както и покритията на векторите от n-мерния булев куб $\{0,1\}^n$ чрез клаузите[1] от множествата $S_0, \ldots, S_3$. На тази основа се извеждат критерии: в Теорема 3.14 се формулират необходими условия, а в Теорема 3.15 – достатъчни условия за неудовлетворимост. Това са основните твърдения в раздела, условията в които са полиномиално проверими. Те се използват за оценяване броя на удовлетворимите и броя на неудовлетворимите КНФ отдолу. Оценките показват, че на базата на формулираните критерии твърде малка част от екземплярите на задачата за 3-УБФ могат да бъдат решени за полиномиално време спрямо

[1] клаузата $c(x_1, \ldots, x_n) = x_{i_1}^{\sigma_1} \vee \ldots \vee x_{i_k}^{\sigma_k}$ покрива вектора $\alpha \in \{0,1\}^n$, ако $c(\alpha) = 0$

големината на входа им. Показано е, че когато не са налице нито необходимите, нито достатъчните условия, сложността на задачата 3-УБФ си остава експоненциална. Направен е още един извод: стойностите 8 и $7\binom{n}{3}$ за броя на клаузите в КНФ на входа се явяват начало и край на "фазов преход" (рязък преход от лесноpешими към трудноpешими екземпляри или обратно при определени гранични стойности на големината на входа; модел, представен в раздел 3.1.2). В този преход с голям диапазон попадат мнозинството (т. е. трудните) екземпляри на задачата 3-УБФ. Резултатите и в този раздел са получени съвместно с доц. д-р Красимир Манев. Докладвани са на XXIX Пролетна конференция на СМБ (2000 г.) и са публикувани в [26].

Резюме на четвърта глава

Разглеждайки покритията на n-мерния булев куб $\{0,1\}^n$ чрез клаузи (в раздел 3.3), потърсихме възможности и начини за покриването му чрез минимален възможен брой клаузи. Така достигнахме до задачата за разбиване на куба на подкубове и преброяване на тези разбивания. Оказа се, че тази задача е тясно свързана с разбивания на числото 2^n по степени на двойката (т. е. представянето му като сума от събираеми, всяко от които е степен на числото 2) и преброяване на тези разбивания. Тя е частен случай на задачата за преброяване разбиванията на числото m^n по степени на числото m (т. нар. m-ични разбивания), където m и n са естествени числа, $m > 1, n > 0$. В тази глава се разглежда една по-обща *задача за преброяването, свойствата и приложенията на m-ичните разбивания на суми от k събираеми, всяко от които е равно на m^n*.

В първи раздел са представени задачата за преброяване на разбиванията на дадено естествено число n по степени на двойката и по-общата – за преброяване на m-ичните разбивания на n. Те са изследвани още от Ойлер, Тантури, Малер, а по-късно и от Чърчхаус, Андрюс, Рьодсет и Гупта [36, 22, 23, 59, 49]. По аналитичен път те успяват да получат рекурентни формули за функциите, представящи броя на съответните разбивания, които нямат добър затворен вид и могат да бъдат пресмятани без компютър само за малки стойности на входните данни m и n. Тези формули са в основата на алгоритъм, наречен Алгоритъм А, който има експоненциална времева сложност (подобен алгоритъм е използван в [72]). С Алгоритъм А се сравняват алгоритмите, разработени в следващите раздели. При определяне и сравняване на техните сложности се оценяват само броя и вида на операциите, които даден алгоритъм изпълнява. Тоест, приема се "критерият за еднаква цена" (представен в раздел 2.3), според който дадена операция има една и съща цена, независимо от големината на операндите.

Във втори раздел на четвърта глава се разглеждат двата основни вида разбивания на сума от k събираеми, всяко от които е равно на m^n (тази сума означаваме кратко с $k.m^n$). От първия вид са тези разбивания, в които не се съдържа нито едно тривиално разбиване на някое от събирае-

мите (разбиването $m^n = m^n$ се нарича тривиално разбиване). При втория вид са позволени тривиални разбивания на някои от събираемите. Чрез прилагане на стратегията "динамично програмиране" се построяват два алгоритъма, наречени B и C, за преброяване на съответните разбивания от двата вида. За целта първо се извеждат рекурентни формули за броя на тези разбивания, доказани в Теореми 4.3 и 4.4. Алгоритмите B и C работят нерекурсивно (по метода "отдолу нагоре") и определят броя на съответните разбивания чрез попълване на таблици. В края на раздела е направена оценка на сложността на двата алгоритъма, която се оказва експоненциална.

В трети раздел се извеждат алгебрични и комбинаторни свойства на числата от таблиците, получени като резултат от изпълнението на алгоритмите B и C. Съпоставени са на други известни числа и таблици – тези на Паскал, на Стирлинг, на Ойлер. Тези няколко свойства пряко следват от дефинициите, твърденията и поясненията в раздел 4.2 и са формулирани като следствия. Те доизясняват смисъла на числата от таблиците, определят връзката между различните видове разбивания и съответните им таблици, както и връзката им с многоъгълните и тетраедралните числа.

В раздел 4.4 се обсъжда полиномиалното представяне на броя на разглежданите разбивания. Теореми 4.15 и 4.16 са основните твърдения в раздела, те утвърждават съществуването и дават вида на старшия едночлен в съответните полиноми. Явният вид на самите полиноми трудно може да бъде изразен чрез формула поради връзката му с числата на Бернули, които нямат представяне в затворен вид. В замяна на това са предложени начини за получаване на явния вид на тези полиноми за малки стойности на n. Направена е оценка на сложността на алгоритъм, който пресмята броя на разглежданите разбивания при известен явен вид на представящия ги полином. На тази основа се обсъждат възможностите за подобряване ефективността на алгоримите B и C в общия случай – сложностите им наистина се подобряват, но пак си остават от експоненциален вид. Предложен е нов алгоритъм, наречен Алгоритъм D, принципно различен от алгоритмите B и C. Той се основава на твърденията, бележките и примерите, представени в раздела. При дадени входни данни m, n и k Алгоритъм D пресмята коефициентите на полинома, представящ броя на m-ичните разбиванията на сумата $k.m^n$, след което изчислява този брой. Направеният анализ на сложността показва, че алгоритъмът решава задачата за полиномиално време $\Theta(n^3)$ и ползва полиномиално количество памет $\Theta(n^2)$ (според критерия за еднаква цена).

В последния пети раздел на четвърта глава са представени две приложения на разглежданите разбивания. Първото от тях е свързано с изброяване на определен вид пълни m-ични дървета, а второто – с изброяване на разбиванията на n-мерния булев куб на подкубове, т. е. първоначалната задача, чиито обобщения са представени в тази глава.

Първите резултати относно разбиванията на n-мерния булев куб на подкубове и тяхното преброяване са публикувани в [3]. Част от пос-

ледвалите изследвания са докладвани на Националния годишен семинар по теория на кодирането и приложения (2000 г.), организиран от Секция МОИ на ИМИ при БАН. Всички изследвания и резултати, представени в четвърта глава, са приети за публикуване в [27].

Резюме на пета глава

В последната пета глава на дисертацията се изследват три отворени задачи в областта на монотонните булеви функции (МБФ). Основните дефиниции и означения, свързани с тях, са дадени в първи раздел.

Във втори раздел се въвежда матрица, представяща предхожданията на векторите от n-мерния булев куб. Теореми 5.13, 5.15 и бележките след тях формулират алгебричните и комбинаторните свойства на тази матрица и биективната връзка между векторите на $\{0,1\}^n$, редовете на матрицата и елементарните конюнкции без отрицания, съдържащи не повече от n букви на променливи. Тези структурни свойства са в основата на алгоритмите, разгледани в следващите раздели.

Трети раздел е посветен на *генерирането на МБФ на n променливи.* Тази задача е свързана с най-старата и все още нерешена задача в областта на МБФ, а именно *задачата на Дедекинд за определяне броя на МБФ на n променливи* (или броя на всички антивериги в частично наредено множество). До 1999 г. беше известен броя на МБФ за не повече от шест променливи. През същата година сръбски учени успяха да получат този брой за $n = 7$ и $n = 8$, използвайки нови аналитични резултати, съчетани с компютърни методи [71, 67]. По тази причина прекратихме опитите си за определяне броя на МБФ на седем променливи чрез "генериране и броене". Това беше първоначалната цел, с която беше разработен представеният в този раздел алгоритъм за генериране на всички МБФ на n променливи в лексикографски ред, наречен *Gen.* Той се съпоставя на друг, принципно различен алгоритъм за решаване на същата задача, даден в [68] и предназначен основно за тестване сортировка на мрежи. След описанието на Gen е представен основният фрагмент от кода му (написан на Паскал), коментирани са особеностите на реализацията и изпълнението му.

В четвърти раздел се разглежда *задачата за определяне на поне един минимален истинен вектор*[2] *(МИВ) и/или поне един максимален неистинен вектор (МНВ) на неизвестна МБФ.* Тази задача, заедно със задачата на Дедекинд и задачата за идентифициране, са едни от най-известните и важни задачи в областта на МБФ [13]. В публикациите на руски език тази задача се разглежда при предположението, че неизвестната МБФ f е зададена с помощта на някакъв оператор A_f, който за произволен вектор $\alpha \in \{0,1\}^n$ връща стойността на $f(\alpha)$. В публикациите на английски език същата задача се разглежда в термините на т. нар. "теория на изчислителното изучаване" (computational learning theory). Към хипотетична оракулна машина се задават т. нар. "въпроси за членство" (membership

[2] вж. Дефиниции 5.2 и 5.3

queries) – дали избран вектор $\alpha \in \{0,1\}^n$ е истинен вектор за f. Машината отговаря с "да" или "не". Представени са някои изследвания върху тази задача и най-вече популярният алгоритъм на Гайнанов [32, 33, 55]. За произволна МБФ f на n променливи този алгоритъм определя вектор, който е или МИВ, или МНВ на f. Той има времева сложност $O(n)$ и използва $O(n)$ обръщения към оператора A_f. Като негова алтернатива е представен друг алгоритъм в две модификации, наречени *Search_First* и *Search_Last*. Те определят съответно лексикографски първия МИВ и лексикографски последния МНВ на неизвестна МБФ на n променливи. Основават се на матричната структура и нейните свойства и представляват обикновено двоично търсене. Те определят съответните вектори, като използват точно n въпроса, без да изпълняват никакви операции над векторите (за разлика от алгоритъма на Гайнанов).

Последният пети раздел на тази глава е посветен на *задачата за идентифициране (разпознаване, разшифровка) на неизвестна МБФ чрез въпроси за членство*. Идентифицирането означава да се определи множеството от всички МИВ и/или множеството от всички МНВ на неизвестната функция. В термините на теорията на изчислителното изучаване това е пример за "точно изучаване" чрез въпроси за членство. В началото на раздела се обсъждат някои характеристики и изисквания към даден изучаващ алгоритъм. След това са представени резултати от изследвания върху ограничен вариант на задачата за идентифициране [32, 33, 54, 55]. Предложените в тях алгоритми имат полиномиална времева сложност и използват полиномиален брой въпроси спрямо сумарната големина на входа и изхода. С помощта на тези резултати в [60] е доказано, че почти всички МБФ са изучаеми чрез въпроси за членство за полиномиално общо време. В този раздел е представен друг алгоритъм за идентифициране на неизвестна МБФ (без ограничения), наречен *Identify*. Той отново се основава на свойствата на матричната структура и ползва алгоритмите Search_First и Search_Last. Най-общо Identify решава задачата, като разделя неизвестната функция на n променливи f на две подфункции g и h (монотонни, на $n-1$ променливи, такива че конкатенацията на вектор-стълбовете им дава вектора-стълб на f). След това алгоритъмът идентифицира рекурсивно g и после h, разделяйки ги на подфункции (както при f), докато на поредната стъпка се получи подфункция на една променлива или подфункция с вектор-стълб от вида $(0,0,\ldots,0,1,1,\ldots,1)$. Тогава започва отделяне на нейните МИВ. Identify се подчинява на стратегията "динамично програмиране". Основната процедура в него е Id, представена е нейна реализация на Паскал. След това са дадени коментари и пояснения относно работата на Identify. Накрая са представени по-важните експериментални резултати от работата на алгоритъма при идентифицирането на всички МБФ на не повече от шест променливи. Те показват, че алгоритъмът използва полиномиален брой въпроси (по-малко отколкото при най-добрите алгоритми, които решават ограничена задача за идентифициране). Времевата му сложност е експоненциална – поради факта, че използва функция-догадка, в която се отразяват натрупаните частични знания за неизвестната

функция. Налице са идеи за съществени промени в алгоритъма с цел подобряване порядъка на времевата му сложност.

Първите изследвания, свързани матрицата P_n и алгоритъма Gen, са публикувани в [2]. Следващите изследвания и резултати са докладвани на Международната конференция по групи и графи (2002 г.), Националния годишен семинар по теория на кодирането и приложения (2002 г.), организиран от Секция МОИ на ИМИ при БАН и на XXXII Пролетна конференция на СМБ (2003 г.) и са публикувани в [28, 29].

Благодарности

Авторът изказва голямата си благодарност и признателност на научния си консултант доц. д-р Красимир Манев за всичко, което научих от него при съвместната ни работа, за това, че ме въведе в проблематиката на дисертацията, за ценните съвети, препоръки и бележки. Благодаря на доц. д-р Стоян Капралов за неговата отзивчивост, за полезните идеи, консултации и напътствия през цялата ни съвместна дейност. Благодарен съм на проф. д.м.н. Стефан Додунеков за неговата подкрепа и ползотворните съвети. Признателен съм на колегите си от катедра "Алгебра и геометрия" при ВТУ "Св. Св. Кирил и Методий", които бяха съпричастни с работата ми и ме подкрепиха във важни моменти. Благодарности на д-р Милен Христов за ценните идеи и съвети и на доц. д-р Стефка Буюклиева за бележките и препоръките към четвърта глава. Признателен съм и на колегите от Секция МОИ на ИМИ при БАН в гр. В. Търново за възможността да представя и обсъдя някои резултати на семинарите, организирани от тях, за оказаната ми помощ с литература и консултации. Благодаря на организаторите и участниците в националния годишен семинар по теория на кодирането и приложения (организиран от Секция МОИ на ИМИ при БАН) за възможностите да участвам в докладите и обсъжданията, за ползотворните срещи и творческия дух на семинарите. Благодарности и на всички неспоменати колеги, приятели и близки, които по един или по друг начин подпомогнаха написването на дисертацията.

Глава 1

Основни дискретни обекти и елементи от теория на булевите функции

В тази глава са представени дискретни обекти като азбуки, езици, n-мерен булев куб, а също по-важните дефиниции, означения и твърдения от теория на булевите функции, необходими за следващите части. Те са дадени основно според учебника [16], с малки изключения за някои означения, утвърдени в периодичните издания. Ползвани са и други известни у нас български и руски източници, като учебниците по дискретна математика [9, 20], сборникът [5] и излезлите наскоро учебници [1, 10]. Към тях добавяме и популярните в западните страни [65, 48].

В тази и в следващата глава представяме доказателства само на по-важните твърдения. Доказателствата на останалите понякога са очевидни, или могат да бъдат намерени в основните източници [16, 9, 20].

1.1 Азбуки, думи, езици

Понятията в този раздел не са предмет на изследване в дисертацията, но се използват при дефиниране на други дискретни обекти.

Дефиниция 1.1 Нека $X = \{a_1, a_2, \ldots, a_n\}$ е крайно множество, което наричаме *азбука*, а елементите му – *букви*. Нека $\epsilon \notin X$.

а) ϵ е дума над X, несъдържаща никакви букви и наречена *празна дума*. *Дължината* на празната дума е $d(\epsilon) = 0$;

б) нека α е дума над азбуката X. Тогава αa_i, $\forall\ i = 1, 2, \ldots, n$ е дума над азбуката X. Нейната *дължина* е $d(\alpha a_i) = d(\alpha) + 1$;

в) няма други думи над X, освен ϵ и думите, получени в б).

Нека X^k е множеството от всички думи над азбуката X, които имат дължина k, при $k = 0$ имаме $X^0 = \{\epsilon\}$. Всяка дума от X^k представлява наредена k-орка от букви, т. е. вариация с повторение от k-ти клас над X и следователно $|X^k| = |X|^k = n^k$. Да означим с X^* множеството от всички думи над азбуката X. Тогава фамилията от множества $\{X^0, X^1, \ldots, X^m, \ldots\}$, такива че $X^* = X^0 \cup X^1 \cup \ldots \cup X^m \cup \ldots$ е разбиване на множеството X^* на подмножества с една и съща дължина на думите във всяко от тях.

Множеството X^* е изброимо, понеже е обединение на изброимо безкрайна фамилия от крайни множества.

Дефиниция 1.2 Нека $\alpha = a_{i_1}a_{i_2}\ldots a_{i_k}$ и $\beta = a_{j_1}a_{j_2}\ldots a_{j_m}$ са думи над азбуката X. Двуместната операция $\kappa : X^* \times X^* \to X^*$, такава че $\kappa(\alpha,\beta) = a_{i_1}\ldots a_{i_k}a_{j_1}\ldots a_{j_m}$ наричаме *конкатенация* на α и β и я записваме съкратено $\kappa(\alpha,\beta) = \alpha\beta$. При това $d(\alpha\beta) = d(\alpha) + d(\beta)$.

Очевидно операцията конкатенация е асоциативна, но не е комутативна, а ϵ е неин неутрален елемент, понеже $\epsilon\alpha = \alpha\epsilon = \alpha$, $\forall\alpha \in X^*$.

Дефиниция 1.3 Нека $\alpha \in X^*$. Ще казваме, че думата $\beta \in X^*$ е *поддума* на α, ако съществуват думи $\gamma_1, \gamma_2 \in X^*$, такива че $\alpha = \gamma_1\beta\gamma_2$.

Дефиниция 1.4 Всяко подмножество $L \subseteq X^*$ наричаме *език* над азбуката X.

Според дефиницията $\emptyset \subseteq X^*$ и значи е език – наричаме го *празния език*. Отбелязваме, че той е различен от езика $L = \{\epsilon\}$.

Операциите *обединение, сечение* и *разлика* на езици се дефинират както обичайните операции над множества. Аналог на операцията декартово произведение на множества се явява операцията *произведение* на езици: $\forall L_1, L_2 \in X^*$, $L_1.L_2 = \{\alpha\beta | \alpha \in L_1, \beta \in L_2\}$. Операцията произведение на езици е асоциативна, но не е комутативна – тъй като пораждащата я операция конкатенация на думи има същите свойства.

За всяко естествено число n операцията *степен* на език $L \subseteq X^*$ се дефинира чрез операцията произведение на езици по следния начин:

1) $L^0 = \{\epsilon\}$;

2) $L^n = L^{n-1}.L$, ако $n > 0$.

Операцията *итерация* (още *звезда на Клини*, или *затваряне* [37]) на език $L \subseteq X^*$ се дефинира така:

$$L^* = L^0 \cup L^1 \cup \ldots \cup L^m \cup \ldots .$$

Дефиниция 1.5 Множеството от всички подмножества на дадено множество A наричаме *булеан* на A и го означаваме с 2^A.

Като следствие от твърдението, че булеанът на безкрайно изброимо множество не е изброимо множество получаваме, че множеството 2^{X^*} от всички езици над дадена крайна азбука X не е изброимо. Този факт означава, че множеството 2^{X^*} не може да бъде описвано чрез думи над някаква крайна азбука (които са изброимо много). За практически цели и за нуждите на теоретичната информатика е достатъчно да се работи не с всички езици, а с изброимо подмножество от тях, които могат да бъдат описвани по този начин. Те се наричат *формални езици*.

1.2 N–мерен булев куб

Дефиниция 1.6 Декартовата n–та степен $\{0,1\}^n$ на множеството $\{0,1\}$ наричаме *n–мерен булев куб*. Той се състои от всички n–мерни двоични вектори, т. е. $\{0,1\}^n = \{(a_1, a_2, \ldots, a_n) |\ a_i \in \{0,1\},\ i = 1,2,\ldots,n\}$.

Очевидно $\{0,1\}^n$ съдържа точно 2^n вектора.

Дефиниция 1.7 *Пореден номер* на вектора $\alpha = (a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ наричаме естественото число $\#(\alpha) = a_1.2^{n-1} + a_2.2^{n-2} + \ldots + a_n.2^0$, т. е. естественото число с двоичен запис $a_1 a_2 \ldots a_n$.

Дефиниция 1.8 Нека $\alpha = (a_1, a_2, \ldots, a_n)$ и $\beta = (b_1, b_2, \ldots, b_n)$ са произволни вектори от $\{0,1\}^n$. Естественото число $\rho(\alpha,\beta) = \sum_{i=1}^{n} |a_i - b_i| = \sum_{i=1}^{n} (a_i \oplus b_i)$ (или броят на позициите, в които двата вектора се различават), наричаме *разстояние по Хеминг* между векторите α и β. Векторите α и β наричаме *съседни*, ако се различават само в една позиция ($\rho(\alpha,\beta) = 1$), или *противоположни*, ако те се различават във всичките си n позиции ($\rho(\alpha,\beta) = n$). Ако векторите α и β са съседни и се различават в i-та позиция, наричаме ги *съседни по i-та позиция*.

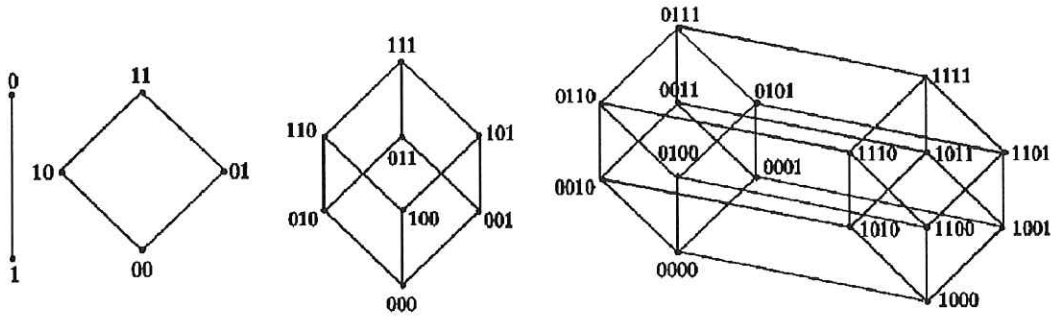
Разстоянието по Хеминг е метрика в $\{0,1\}^n$, тъй като удовлетворява условията:

а) $\rho(\alpha,\beta) \geq 0$, $\forall \alpha, \beta \in \{0,1\}^n$, като $\rho(\alpha,\beta) = 0$ само при $\alpha = \beta$;

б) $\rho(\alpha,\beta) = \rho(\beta,\alpha)$, $\forall \alpha, \beta \in \{0,1\}^n$;

в) $\rho(\alpha,\beta) + \rho(\beta,\gamma) \geq \rho(\alpha,\gamma)$, $\forall \alpha, \beta, \gamma \in \{0,1\}^n$.

Булевите кубове с размерности 1, 2, 3 и 4 са представени на Фиг. 1.1 като графи, чиито върхове са векторите на съответния куб, а ребрата свързват всеки два съседни вектора.



Фиг. 1.1: $\{0,1\}^n$ за $n = 1, 2, 3, 4$.

Дефиниция 1.9 *Тегло* на вектора $\alpha = (a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ наричаме естественото число $wt(\alpha)$, равно на броя на ненулевите компоненти на α.

Теглото на вектора α може да се дефинира още като $wt(\alpha) = \rho(\tilde{0}, \alpha)$, където $\tilde{0}$ е нулевият вектор от $\{0,1\}^n$, или като $wt(\alpha) = \sum_{i=0}^{n} a_i$.

Дефиниция 1.10 Множеството $\{0,1\}_k^n = \{\alpha | \, \alpha \in \{0,1\}^n, \, wt(\alpha) = k\}$ наричаме *k-ти слой* на n-мерния булев куб, за $k = 0, 1, \ldots, n$.

В k-тия слой на n-мерния булев куб се съдържат точно $\binom{n}{k}$ вектора, тъй като k единици могат да бъдат разположени в n позиции по $\binom{n}{k}$ начина. Самият n-мерен булев куб може да се представи чрез разбиване по слоеве:

$$\{0,1\}^n = \{0,1\}_0^n \cup \{0,1\}_1^n \cup \ldots \cup \{0,1\}_n^n \, .$$

Дефиниция 1.11 *k-мерен подкуб*, $1 \leq k \leq n$, на $\{0,1\}^n$ наричаме множеството от всички вектори на $\{0,1\}^n$, които имат еднакви стойности в $(n-k)$ фиксирани позиции, а стойностите им в останалите k позиции (разглеждани като k-мерни вектори) образуват k-мерен булев куб.

Конкретен подкуб в множеството от всички възможни k-мерни подкубове на $\{0,1\}^n$ можем да определим, като зададем позициите с фиксирани стойности и самите стойности във всяка от тях, например:

$$\begin{pmatrix} i_1, i_2, \ldots, i_{n-k} \\ \sigma_1, \sigma_2, \ldots, \sigma_{n-k} \end{pmatrix}, \quad 1 \leq i_1 < i_2 < \ldots < i_{n-k} \leq n,$$

където $\sigma_j \in \{0,1\}$ е фиксираната стойност на всеки вектор в позицията i_j, за $j = 1, 2, \ldots, n-k$.

Броят на всички възможни k-мерни подкубове на $\{0,1\}^n$ получаваме, като умножим броя на начините за избор на $(n-k)$ фиксирани позиции по броя на начините, по които могат да бъдат фиксирани стойностите в тях, т. е.

$$\binom{n}{n-k} 2^{n-k} = \binom{n}{k} 2^{n-k}$$

Дефиниция 1.11 и поясненията след нея са верни и при $k = 0$, ако приемем, че нулевомерен булев куб е множеството $\{0,1\}^0 = \{()\}$, съдържащо единствения нулевомерен вектор – празният вектор. Основание за това е фактът, че всеки двоичен вектор може да бъде разглеждан като дума над множеството $X = \{0,1\}$ и обратно, тоест между множеството от всички думи над азбуката $X = \{0,1\}$ и множеството от всички двоични вектори съществува биекция. Тогава на $X^0 = \{\epsilon\}$ съответства $\{0,1\}^0 = \{()\}$ (на празната дума съответства празният вектор). Приемаме, че поредният номер и теглото на празния вектор са нули. И така при $k = 0$ се получава подкубът $\{\alpha\}$ с минимална възможна размерност, α е неговият единствен n-мерен вектор с n фиксирани (в определени стойности) компоненти.

Дефиниция 1.12 Векторът α *предхожда лексикографски* вектора β, ако съществува естествено число i, $1 \leq i \leq n$, такова че $a_1 = b_1, \ldots, a_{i-1} = b_{i-1}$ и $a_i < b_i$, или ако $\alpha = \beta$. Векторите от $\{0,1\}^n$ са *подредени лексикографски* в редицата $\alpha_0, \alpha_1, \ldots, \alpha_{2^n-1}$, ако α_i предхожда лексикографски α_j за $0 \leq i < j \leq 2^n - 1$.

Отбелязваме, че лексикографското предхождане на вектори може да се дефинира като релация над $\{0,1\}^n \times \{0,1\}^n$, която (очевидно) е рефлексивна, антисиметрична и транзитивна. Такава релация се нарича *релация на частична наредба*, а множеството $\{0,1\}^n$, над което тя е дефинирана – *частично наредено множество*. Освен това, за всеки два вектора от n-мерния булев куб единият винаги предхожда лексикографски другия, т. е. частичната наредба е всъщност *пълна наредба*. Лексикографската наредба на векторите в n-мерния булев куб се нарича още *стандартна наредба* и по-нататък винаги ще считаме, че векторите в $\{0,1\}^n$ са подредени по този начин.

Следващата дефиниция на понятието n-мерен булев куб е конструктивна – дава процедура за получаване на векторите му в стандартна наредба.

Дефиниция 1.13 1) *Едномерен булев куб* наричаме множеството $\{0,1\} = \{(0),(1)\}$, чиито елементи (0) и (1) са едномерни двоични вектори и са подредени лексикографски.

2) Нека $\{0,1\}^{n-1} = \{\alpha_0, \alpha_1, \ldots, \alpha_{2^{n-1}-1}\}$ е $(n-1)$-*мерният булев куб* и нека неговите $(n-1)$-мерни двоични вектори $\alpha_0, \alpha_1, \ldots, \alpha_{2^{n-1}-1}$ са подредени лексикографски.

3) n-*мерният булев куб* $\{0,1\}^n$ построяваме от $\{0,1\}^{n-1}$, като вземем два пъти векторите му: първия път добавяме 0, а втория път добавяме 1 в началото на всеки от тях, т. е.

$$\{0,1\}^n = \{(0\alpha_0),(0\alpha_1),\ldots,(0\alpha_{2^{n-1}-1}),(1\alpha_0),(1\alpha_1),\ldots,(1\alpha_{2^{n-1}-1})\}$$

и следователно те също са подредени лексикографски.

От последната дефиниция следва, че когато векторите на $\{0,1\}^n$ са в стандартна наредба, техните поредни номера образуват редицата от естествени числа $0, 1, \ldots, 2^n - 1$.

Известни са и други наредби на векторите от $\{0,1\}^n$ – например *наредба в код на Грей* (когато векторите са подредени в редица така, че $\rho(\alpha_i, \alpha_{i+1}) = 1$, $i = 0, 1, \ldots, 2^n - 2$, т. е. всеки два съседни вектора в редицата са съседни и в смисъл разстояние по Хеминг), наредби според теглата на векторите (т. е. по слоеве) във възходящ или в низходящ ред и др.

Дефиниция 1.14 Нека $\alpha = (a_1, a_2, \ldots, a_n)$ и $\beta = (b_1, b_2, \ldots, b_n)$ са произволни вектори от $\{0,1\}^n$. Релацията *предхожда*, която означаваме с "$\preceq$", се дефинира над $\{0,1\}^n \times \{0,1\}^n$ както следва: $\alpha \preceq \beta$ (четем α *предхожда* β), ако $a_i \leq b_i$ за $i = 1, 2, \ldots, n$.

Лесно се проверява, че релацията "предхожда" е релация на частична наредба и спрямо нея $\{0,1\}^n$ отново е частично наредено множество. Но в n-мерния булев куб съществуват двойки вектори, за които не може да се каже, че единият предхожда другия, т. е. не са в релация и следователно релацията "предхожда" не е релация на пълна наредба. Когато $\alpha \preceq \beta$ или $\beta \preceq \alpha$, векторите α и β се наричат *сравними*, в противен случай те се наричат *несравними*.

1.3 Булеви функции. Суперпозиции и формули

В [16, 65] се разглеждат т. нар. дискретни функции, които се дефинират като функции от едно изброимо множество в друго такова множество. В по-тесен смисъл, всяка функция $f : A^n \to A$, където $A = \{0, 1, \ldots, q-1\}$, $q \geq 2$ и $n \geq 1$, се нарича *дискретна* (или q-*ична*) функция. В частност, при $q = 2$ се получават функции, наречени двоични или още булеви – в чест на Джордж Бул[1].

[1] Джордж Бул (1815-1864) е английски математик и логик, без специално математическо образование. От 1849 г. до края на живота си е професор по математика в Куинс Колидж, Корк, Ирландия. През 1848 г. той публикува "The Mathematical Analysis of Logic" – първата работа по символна логика, а през 1854 г. публикува най-известния от своите трудове "The Laws of Thought", с които поставя основите на математическата (и в частност на двоичната) логика.

Дефиниция 1.15 *Булева* (или *двоична*) функция на n независими променливи $x_1, x_2, \ldots, x_n$ наричаме всяка функция $f : \{0,1\}^n \to \{0,1\}$.

Според дефиницията произволна булева функция на n променливи съпоставя стойност 0 или 1 на всеки вектор от n-мерния булев куб, а нейните променливи $x_1, \ldots, x_n$ съответстват на координатите на векторите от $\{0,1\}^n$. Задаването на такава функция f може да стане чрез посочване (например в таблица) на двойките: вектор α_i, стойност на функцията $a_i = f(\alpha_i)$ за този вектор, за $i = 0, 1, \ldots, 2^n - 1$. От съображения за икономичност, задаването на векторите от $\{0,1\}^n$ може да се пропусне – приехме, че те винаги са в стандартна наредба. Достатъчно е да се изброи последователността от стойности, които булевата функция приема за всеки вектор – например $f(x_1, x_2, \ldots, x_n) = (a_0, a_1, \ldots, a_{2^n-1})$, като $a_i \in \{0,1\}$ е стойността на f за вектора с пореден номер равен на i, за $i = 0, \ldots, 2^n - 1$. Векторът $(a_0, a_1, \ldots, a_{2^n-1})$ се нарича *вектор с функционални стойности* на f, а по-нататък ще го наричаме кратко *вектор* на f.

Означаваме с $\mathcal{F}_2^n$ множеството от булевите функции на n променливи, $\mathcal{F}_2^n = \{f | f : \{0,1\}^n \to \{0,1\}\}$. Всяка функция от $\mathcal{F}_2^n$ се определя еднозначно чрез нейния вектор и следователно броят на всички функции в това множество е равен на броя на начините, по които стойностите в 2^n бита могат да варират, тоест $|\mathcal{F}_2^n| = 2^{2^n}$. Множеството от всички булеви функции означаваме с $\mathcal{F}_2$, т. е. $\mathcal{F}_2 = \mathcal{F}_2^0 \cup \mathcal{F}_2^1 \cup \ldots \cup \mathcal{F}_2^n \cup \ldots$, като множеството $\mathcal{F}_2^0$ ще бъде разгледано по-долу.

Дефиниция 1.16 Нека $f(x_1, x_2, \ldots, x_n) \in \mathcal{F}_2^n$ и $g(y_1, y_2, \ldots, y_m) \in \mathcal{F}_2^m$. Функцията

$h(x_1, \ldots, x_{i-1}, y_1, \ldots, y_m, x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, g(y_1, \ldots, y_m), x_{i+1}, \ldots, x_n)$

наричаме *суперпозиция* на функцията g в f на мястото на променливата x_i.

Операцията суперпозиция може да бъде прилагана многократно не само върху буквите на променливи на f, но и върху буквите на променливи на h. Така се получават суперпозиции, съдържащи в себе си суперпозиции, тоест вложени суперпозици.

Нека F е дадено множество от булеви функции. Множеството от всевъзможните суперпозиции на функциите от F означаваме с $[F]$ и го наричаме *затваряне* (още *затворена обвивка*) на множеството F по отношение на суперпозицията.

Да разгледаме функцията $f(x) = (a, a)$, която приема една и съща стойност $a \in \{0,1\}$, независимо от стойностите на аргумента x. В зависимост от стойността на a, такава функция се нарича *константа 0* и се означава с $\tilde{0}$, или *константа 1* и се означава с $\tilde{1}$. При суперпозиция (или заместване) на константа на мястото на буква на променлива в произволна булева функция се получава нова функция, чиито стойности не се влияят от стойностите на заместената променлива. Във вектора на новата функция се повтарят (в определен ред) стойностите на вектора на друга функция, която има една променлива по-малко.

Дефиниция 1.17 Функцията $f \in \mathcal{F}_2^n$ *не зависи съществено* от променливата си x_i, ако $f(\alpha) = f(\beta)$ за всички двойки вектори $\alpha, \beta \in \{0,1\}^n$, съседни

по i-та позиция. Казваме още, че променливата x_i е *фиктивна* (или *несъществена*) за функцията f. И обратно: променливата x_i е *съществена* за функцията f (или f *зависи съществено* от x_i), ако съществува двойка съседни по i-та позиция вектори $\alpha, \beta \in \{0,1\}^n$, такива че $f(\alpha) \neq f(\beta)$.

Ако $f \in \mathcal{F}_2^n$ и $f(x_1, \ldots, x_n) = (a_0, a_1, \ldots, a_{2^n-1})$, можем да добавим фиктивната променлива x_{n+1} като дефинираме новата функция $g(x_1, \ldots, x_n, x_{n+1}) = (a_0, a_0, a_1, a_1 \ldots, a_{2^n-1}, a_{2^n-1})$. Така например при двете константи, разглеждани като функции на n променливи, всичките им променливи са фиктивни. Това ни дава основание да отъждествяваме булевите стойности 0 и 1 съответно с $\tilde{0}$ и $\tilde{1}$ и да ги разглеждаме като единствени функции на нула променливи (или дефинирани върху нулевомерния булев куб). По-нататък няма да различаваме функциите f и g, ако едната е получена от другата чрез добавяне (или премахване) на фиктивни променливи. Като такива ще разглеждаме и функциите от следващата дефиниция, получени чрез подходящо добавяне на фиктивни променливи към функцията $f(x) = x$.

Дефиниция 1.18 Функция от вида $f(x_1, x_2, \ldots, x_n) = x_i,\ i = 1, \ldots, n$ наричаме *идентитет* (още *променлива* или *тъждествена функция*).

Представянето на булеви функции чрез вектори в много от случаите също е неикономично. По-компактен и много по-удобен за работа е един друг начин за представяне. Нека $X = \{x_1, x_2, \ldots\}$ е изброимо множество от букви на променливи и нека $F = \{f_1, f_2, \ldots\} \subseteq \mathcal{F}_2$. Нека I е крайна азбука и $\tau : N \to I^*$ е биекция, чрез която записът f_i или x_j да означава думите $f\alpha$ или $x\beta$, такива че $\alpha = \tau(i)$, $\beta = \tau(j)$, $\alpha, \beta \in I^*$. Образуваме азбуката $H = I \cup \{f, x, (\,,\,), <\text{запетая}>\}$.

Дефиниция 1.19 *Формули* над множеството от функции F наричаме всички думи от H^*, зададени чрез условието а) или получени чрез операцията б), където:

а) за всяка $f_i \in F$ думата $f_i(x_1, x_2, \ldots, x_n) \in H^*$ е формула над F;

б) нека $f_i \in F$ е функция на n променливи, а $\varphi_1, \varphi_2, \ldots, \varphi_n$ са формули над F или променливи от X. Тогава думата $f_i(\varphi_1, \varphi_2, \ldots, \varphi_n) \in H^*$ е формула над F.

Например, ако $f_1(x_1, x_2)$ и $f_2(x_1)$ са функции от F, то думите $f_1(x_1, x_2)$, $f_2(x_3)$, $f_1(f_2(x_1), f_1(x_2, x_3))$ са формули над F. Всяка поддума φ' на формулата φ над F, $\varphi' \neq \varphi$, която е формула, наричаме *подформула* на φ.

Дефиниция 1.20 Нека $F = \{f_1, f_2, \ldots\} \subseteq \mathcal{F}_2$ и нека Φ_F е множеството от формули над F. На всяка формула $\varphi \in \Phi_F$ съпоставяме функция от $f \in F$, следвайки дефиницията на формула по следния начин:

а) ако φ е някоя $f_i \in F$, тогава ѝ съпоставяме функцията $f = f_i$;

б) нека $f_i \in F$ е функция на n променливи, а $\varphi_1, \varphi_2, \ldots, \varphi_n \in \Phi_F$ са формули, на които сме съпоставили съответно функциите $g_1, g_2, \ldots, g_n \in F$ (ако $\varphi_j = x_k$, то g_j е идентитетът x_k). Тогава на формулата $\varphi = f_i(\varphi_1, \varphi_2, \ldots, \varphi_n)$ съпоставяме функцията (суперпозицията) $h = f_i(g_1, g_2, \ldots, g_n)$.

Казваме, че формулата φ *реализира* функцията f, ако φ е съпоставена на f според горната дефиниция. Отбелязваме, че връзката между суперпозициите на функциите от дадено множество и формулите, които ги реализират, не е биективна. По-нататък ще забележим, че в редица случаи една и съща функция може да се реализира чрез много (безкрайно много) формули. Казваме, че формулите φ_1 и φ_2 са *еквивалентни* (означаваме $\varphi_1 = \varphi_2$), ако реализират една и съща функция f.

Дефиниция 1.21 На всяка формула $\varphi \in \Phi_F$ съпоставяме еднозначно естественото число $\nu(\varphi)$, наречено *дълбочина* на формула, според правилата:

а) буквите на променливи имат дълбочина 0;

б) ако $\varphi = f_i(\varphi_1, \varphi_2, \ldots, \varphi_n)$, то $\nu(\varphi) = 1 + max_{1 \leq i \leq n}\{\nu(\varphi_i)\}$.

Например, за всяка $f_i \in F$ имаме $\nu(f_i) = 1$, а за $h = f(x_1, g_1(x_2, g_2(x_3)))$ имаме $\nu(h) = 3$.

Следващите две теореми представят някои свойства на формулите.

Теорема 1.22 *Нека $F = \{f_1, f_2, \ldots\} \subseteq \mathcal{F}_2$ и $G = \{g_1, g_2, \ldots\} \subseteq \mathcal{F}_2$ са такива множества, че f_i и g_i имат един и същи брой променливи за $i = 1, 2, \ldots$. Операцията θ заменя в произволна формула φ над F всички поддуми $f_1, f_2, \ldots$ съответно с думите $g_1, g_2, \ldots$. Тогава резултатът $\theta(\varphi)$ от операцията е формула над G.*

Теорема 1.23 *Нека $F = \{f_1, f_2, \ldots\} \subseteq \mathcal{F}_2$, а φ е формула над F, в която участват променливите $x_1, x_2, \ldots, x_n$. Нека $\psi_1, \psi_2, \ldots, \psi_n$ са формули над F или променливи. Операцията θ заменя във φ всяко срещане на променливата x_i с ψ_i, $i = 1, 2, \ldots, n$. Тогава резултатът $\theta(\varphi)$ е формула над F.*

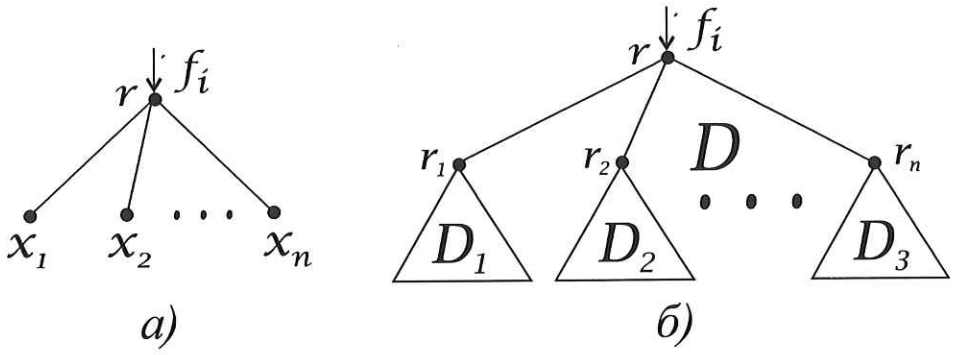
Последните три дефиниции могат да бъдат обединени в една алтернативна дефиниция [16], която илюстрира трите понятия чрез коренови дървета и е по-полезна за теоретичната и приложната информатика.

Дефиниция 1.24 Нека $F = \{f_1, f_2, \ldots\} \subseteq \mathcal{F}_2$ и нека $X = \{x_1, x_2, \ldots\}$ е множество от букви на променливи.

а) Всяко кореново дърво $D = (V, E)$ с корен r, височина 1 и надписваща върховете му функция $t : V \to F \cup X$, такава че коренът му r е надписан с $f_i \in F$, а всеки лист – с променлива от X, е формула над F. Функцията, съпоставена на тази формула D, е $f_i(x_1, \ldots, x_n)$, а дълбочината ѝ $\nu(D)$ се определя от височината на това дърво, а именно $\nu(D) = h(D) = 1$ – както на Фиг. 1.2 а).

б) Нека $f_i \in F$ е функция на n променливи и $D_1 = (V_1, E_1)$, $\ldots, D_n = (V_n, E_n)$, $V_i \cap V_j = \emptyset$, $1 \leq i < j \leq n$ са формули (съотв. надписани по върховете коренови дървета) над F или са самостоятелни върхове (т. е. тривиални коренови дървета), надписани с букви на променливи. Нека $r_1, r_2, \ldots, r_n$ са съответно техните корени, $t_1, t_2, \ldots, t_n$ – съответно функциите, надписващи върховете им, а $g_1, g_2, \ldots, g_n$ – съответните им функции от $\mathcal{F}_2$ (идентитетът съответства на тривиално дърво). Нека $r \notin V_1 \cup V_2 \cup \ldots \cup V_n$. Тогава дървото $D = (V, E)$, $V = V_1 \cup \ldots \cup V_n \cup \{r\}$, $E = E_1 \cup \ldots \cup E_n \cup \{(r, r_1), \ldots, (r, r_n)\}$

и с корен r – вж. Фиг. 1.2 б), за което $t : V \to F \cup X$ се определя като $t(r) = f_i$ и $t(v) = t_k(v)$, ако $v \in V_k$, $1 \leq k \leq n$, е също формула над F. Функцията, която ѝ съпоставяме, е суперпозицията $h = f_i(g_1, g_2, \ldots, g_n)$, а дълбочината ѝ $\nu(D)$ се определя от височината $h(D)$ и очевидно тя е $\nu(D) = 1 + max_{1 \leq i \leq n}\{h(D_i)\} = 1 + max_{1 \leq i \leq n}\{\nu(D_i)\}$.



Фиг. 1.2: Представяне на формули чрез дървета.

1.4 Булеви функции на една и две променливи

В предишния раздел показахме, че $|\mathcal{F}_2^n| = 2^{2^n}$ и за $n = 1$ имаме $|\mathcal{F}_2^1| = 4$. Тези четири функции са представени чрез техните вектори в Таблица 1.1.

x	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

Таблица 1.1: Булеви функции на 1 променлива.

Функциите f_0 и f_3 са познатите вече константи – съответно $\tilde{0}$ и $\tilde{1}$, като функции на една фиктивна променлива. Функцията $f_1(x) = x$ е идентитетът, а $f_2(x) = \bar{x}$ е *отрицание* на x.

Булевите функции на 2 променливи са 16 и са представени по аналогичен начин в Таблица 1.2.

x y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Таблица 1.2: Булеви функции на 2 променливи.

Функциите f_0 и f_{15} са отново двете константи, вече като функции на 2 (фиктивни) променливи.

Функциите $f_3(x, y) = x$ и $f_5(x, y) = y$ са двата идентитета, а $f_{12}(x, y) = \bar{x}$ и $f_{10}(x, y) = \bar{y}$ са съответните им отрицания – при всички тях само едната променлива е съществена.

Останалите функции в $\mathcal{F}_2^2$ зависят съществено и от двете си променливи и при тях ще използваме инфиксен запис за двуместните операции над x и y. Ще ги означаваме и наричаме така:

• $f_1(x,y) = xy = x \wedge y$ – *конюнкция*, *логическо умножение* (като умножение над полето $GF(2)$), или още *логическо "и"*;

• $f_7(x,y) = x \vee y$ – *дизюнкция*, *логическо "или"*, или още *включващо "или"*;

• $f_6(x,y) = x \oplus y$ – *събиране (сума) по модул 2* (като събиране над полето $GF(2)$), или още *изключващо "или"*;

• $f_9(x,y) = x \equiv y$ – *еквивалентност*, явява се отрицание на сумата по модул 2;

• $f_{13}(x,y) = x \to y$ – *импликация*, четем x *имплицира* y, или *от* x *следва* y;

• $f_{11}(x,y) = y \to x$ – *обратна импликация*, четем y *имплицира* x, или *от* y *следва* x;

• $f_8(x,y) = x \downarrow y$ – *стрелка на Пирс*, явява се отрицание на дизюнкцията;

• $f_{14}(x,y) = x|y$ – *черта на Шефер*, явява се отрицание на конюнкцията.

Функциите f_2 и f_4 (отрицанията съответно на f_{13} и f_{11}) нямат общоприет инфиксен запис. При суперпозиции на функции на две променливи заменяме съответните подформули с техния инфиксен запис – например, вместо формулата $f_7(f_1(x,y), f_6(f_{12}(x,y), f_5(y)))$ пишем: $(xy) \vee ((\bar{x}) \oplus y)$, като добавяме необходимите скоби. С въвеждането на приоритети на функциите някои скоби могат да отпаднат и записът им да се опрости. Общоприето е отрицанието да има най-висок приоритет сред останалите, след него е конюнкцията, а след нея с равен приоритет са дизюнкцията и сумата по модул 2. Формулата от дадения пример добива вида $xy \vee (\bar{x} \oplus y)$.

При функциите на 3 и повече променливи инфиксният (или друг подобен) запис е невъзможен. По-нататък ще покажем как те могат да бъдат записвани като формули над подходящо множество булеви функции.

Някои свойства на най-често използваните булеви функции на 2 променливи са представени в следващата теорема.

Теорема 1.25 *За булеви функции на 2 променливи са в сила:*

а) идемпотентни свойства: $xx = x,\ x \vee x = x$;

б) комутативни свойства: $xy = yx,\ x \vee y = y \vee x,\ x \oplus y = y \oplus x$;

в) асоциативни свойства:

$(xy)z = x(yz),\ (x \vee y) \vee z = x \vee (y \vee z),\ (x \oplus y) \oplus z = x \oplus (y \oplus z)$;

г) дистрибутивни свойства:

$x(y \vee z) = xy \vee xz,\ x \vee (yz) = (x \vee y)(x \vee z),\ x(y \oplus z) = xy \oplus xz$;

д) свойства на отрицанието: $x\bar{x} = \tilde{0},\ x \vee \bar{x} = \tilde{1},\ x \oplus \bar{x} = \tilde{1}$;

е) свойства на константите:

$x\tilde{0} = \tilde{0},\ x \vee \tilde{0} = x,\ x \oplus \tilde{0} = x;\ x\tilde{1} = x,\ x \vee \tilde{1} = \tilde{1},\ x \oplus \tilde{1} = \bar{x}$;

ж) закон за двойното отрицание: $\bar{\bar{x}} = x$;

з) закони на Де Морган: $\overline{(xy)} = \bar{x} \vee \bar{y},\ \overline{(x \vee y)} = \bar{x}\bar{y}$.

Ще докажем верността на първия закон на Де Морган, останалите свойства се доказват аналогично. С това илюстрираме един от начините за проверка еквивалентността на формули – като пресметнем двата

вектора на функциите, реализирани чрез формулите от двете страни на равенството и ги сравняваме – в случая трета и шеста колона в Таблица 1.3.

$x\ y$	xy	$\overline{(xy)}$	$\bar{x}$	$\bar{y}$	$\bar{x} \vee \bar{y}$
0 0	0	1	1	1	1
0 1	0	1	1	0	1
1 0	0	1	0	1	1
1 1	1	0	0	0	0

Таблица 1.3: Доказателство на първия закон на Де Морган.

Към свойствата, дадени в теоремата, можем да добавим, че формулираните комутативни свойства са валидни и за отрицанията на съответните функции, че идемпотентните свойства не са в сила за други функции и т. н. Ще използваме разгледаните свойства, за да докажем други полезни свойства на булефите функции. Нека $f \in \mathcal{F}_2$ и нека да опростим формулата $fx \vee f\bar{x}$. Получаваме веригата от равенства $fx \vee f\bar{x} = f(x \vee \bar{x}) = f\tilde{1} = f$. Второто равенство следва от първото съгласно дистрибутивните свойства, третото следва от второто съгласно свойствата на отрицанието, а четвъртото следва от третото от свойствата на константите. Полученото свойство $fx \vee f\bar{x} = f$ се нарича *слепване* на fx и $f\bar{x}$. Аналогично получаваме, че за произволни $f, g \in \mathcal{F}_2$ е в сила свойството *поглъщане*: $fg \vee f = f$.

Асоциативността на функциите дизюнкция, конюнкция и сума по модул 2 позволява да се освободим от скобите във формули, в които една от тях се прилага многократно. Подобно на означенията $\sum$ и $\prod$ при реалните числа, ще използваме означенията:

$$\bigvee_{i=1}^{m} f_i = f_1 \vee \ldots \vee f_m, \quad \bigwedge_{i=1}^{m} f_i = f_1 \wedge \ldots \wedge f_m, \quad \bigoplus_{i=1}^{m} f_i = f_1 \oplus \ldots \oplus f_m.$$

съответно за многократна дизюнкция, многократна конюнкция и многократна сума по модул 2.

1.5 Пълни множества от булеви функции

В раздел 1.3 дефинирахме понятието затваряне на множество от булеви функции $F \subseteq \mathcal{F}_2$ като множество от всевъзможните суперпозиции над F и го означихме с $[F]$.

Дефиниция 1.26 Множеството $F \subseteq \mathcal{F}_2$ е *пълно*, ако $[F] = \mathcal{F}_2$.

Съществуването на пълни множества от булеви функции не е очевидно, а е изключително важно. Ако такова множество съдържа сравнително малко на брой функции, с тях могат да се построят формули за всички функции от $\mathcal{F}_2$, чрез които да се оперира много по-лесно, отколкото с вектори. Това ще покажем в този раздел и за целта първо дефинираме функцията:

$$f(x, \sigma) = x^{\sigma} = \begin{cases} x \text{ ако } \sigma = 1 \\ \bar{x} \text{ ако } \sigma = 0. \end{cases}$$

б) Ако $f \neq \tilde{1}$, използваме Лема 1.30 и построяваме формулата

$$f(x_1, x_2, \ldots, x_n) = \bigwedge_{\substack{\forall(\sigma_1,\sigma_2,\ldots,\sigma_n):\\ f(\sigma_1,\ldots,\sigma_n)=0}} (x_1^{\overline{\sigma_1}} \vee x_2^{\overline{\sigma_2}} \vee \ldots \vee x_n^{\overline{\sigma_n}}) \tag{1.2}$$

над $\{x \vee y, xy, \bar{x}\}$. Ще докажем, че и тя реализира f.

Нека $(\tau_1, \tau_2, \ldots, \tau_n) \in \{0,1\}^n$ е такъв, че $f(\tau_1, \tau_2, \ldots, \tau_n) = 0$ и да пресметнем стойността на дясната част на равенството (1.2) за него. Тя е конюнкция по $\forall\, (\sigma_1, \sigma_2, \ldots, \sigma_n) :\ f(\sigma_1, \ldots, \sigma_n) = 0$ и следователно в нея участва пълната дизюнкция $x_1^{\overline{\tau_1}} \vee x_2^{\overline{\tau_2}} \vee \ldots \vee x_n^{\overline{\tau_n}}$. След заместване на променливите $x_1, \ldots, x_n$ съответно с $\tau_1, \ldots, \tau_n$ тя добива вида $\tau_1^{\overline{\tau_1}} \vee \tau_2^{\overline{\tau_2}} \vee \ldots \vee \tau_n^{\overline{\tau_n}}$ и според Лема 1.30 стойността ѝ е 0. Следователно стойността на дясната част също става 0.

Нека сега $(\tau_1, \tau_2, \ldots, \tau_n) \in \{0,1\}^n :\ f(\tau_1, \tau_2, \ldots, \tau_n) = 1$. Стойността на дясната част за този вектор ще бъде 1, тъй като във всяка от пълните дизюнкции, участващи в нея, ще има поне по един член $\tau_i^{\overline{\sigma_i}}$, за който $\tau_i \neq \overline{\sigma_i}$, т. е. $\tau_i = \sigma_i$ и според Лема 1.30 стойността на всяка такава дизюнкция е 1.

Следователно формулата на дясната част реализира f. $\diamond$

Двата начина, по които доказахме теоремата на Бул ни дават в явен вид и съответните формули. При $f \neq \tilde{0}$ съответната ѝ формула от вида (1.1) се нарича *Съвършена Дизюнктивна Нормална Форма (СъвДНФ)* на f. При $f \neq \tilde{1}$ нейната формула от вида (1.2) се нарича *Съвършена Конюнктивна Нормална Форма (СъвКНФ)* на f. Форми, аналогични на СъвДНФ (респ. СъвКНФ), в които поне една от елементарните им конюнкции (респ. дизюнкции) не е пълна, се наричат *Дизюнктивни Нормални Форми (ДНФ)* (респ. *Конюнктивни Нормални Форми (КНФ)*). Константата $\tilde{0}$ няма нито една ДНФ (но има една СъвКНФ), а константата $\tilde{1}$ – нито една КНФ (но има една СъвДНФ). Всички останали булеви функции имат точно по една СъвДНФ и СъвКНФ, но могат да имат и повече от една ДНФ или КНФ.

Пример 1.32 Да построим СъвДНФ и СъвКНФ на функцията $f(x,y,z) = (1,0,1,1,0,0,1,0)$.

Векторите, за които f приема стойност 1, са $(0,0,0), (0,1,0), (0,1,1)$ и $(1,1,0)$. Следователно СъвДНФ на f ще бъде

$$f(x,y,z) = x^0y^0z^0 \vee x^0y^1z^0 \vee x^0y^1z^1 \vee x^1y^1z^0 = \bar{x}\bar{y}\bar{z} \vee \bar{x}y\bar{z} \vee \bar{x}yz \vee xy\bar{z}.$$

Векторите, за които f приема стойност 0, са $(0,0,1), (1,0,0), (1,0,1)$ и $(1,1,1)$ и следователно нейната СъвКНФ ще бъде

$$f(x,y,z) = (x^{\bar{0}} \vee y^{\bar{0}} \vee z^{\bar{1}})(x^{\bar{1}} \vee y^{\bar{0}} \vee z^{\bar{0}})(x^{\bar{1}} \vee y^{\bar{0}} \vee z^{\bar{1}})(x^{\bar{1}} \vee y^{\bar{1}} \vee z^{\bar{1}}) =$$
$$(x^1 \vee y^1 \vee z^0)(x^0 \vee y^1 \vee z^1)(x^0 \vee y^1 \vee z^0)(x^0 \vee y^0 \vee z^0) = (x \vee y \vee \bar{z})(\bar{x} \vee y \vee z)(\bar{x} \vee y \vee \bar{z})(\bar{x} \vee \bar{y} \vee \bar{z}).$$

Теорема 1.33 *Нека $F \subseteq \mathcal{F}_2$ е пълно, нека $G \subseteq \mathcal{F}_2$ и $f \in [G]\ \forall f \in F$. Тогава и G е пълно.*

Чрез Теорема 1.33 може да се докаже пълнотата и на други множества, например на: $\{x \vee y, \bar{x}\}$, $\{xy, \bar{x}\}$, $\{x \downarrow y\}$, $\{xy, x \oplus y, \tilde{1}\}$ и т. н.

Глава 2

Елементи от теория на алгоритмите

В тази глава са дадени основните понятия, означения и твърдения, свързани с масови задачи, алгоритми, сложност, $\mathcal{NP}$-пълнота. Изложението основно е според [16], а в допълнение са ползвани получилите класическа известност [8, 21, 37].

2.1 Масови задачи, алгоритми и разрешимост на масови задачи

Понятието *масова задача* е с особена важност не само за математиката, но и за информатиката. Известни са редица неформални определения на това понятие, едно от които е следното. Нека имаме множество от математически обекти със зададени свойства и нека са дефинирани операции над обектите, резултатът от които е (са) обект (обекти) от същото множество. Под масова задача ще разбираме всяка достатъчно ясна цел, състояща се в намиране на някакви обекти $\tilde{y} = (y_1, y_2, \ldots)$ (с предварително определени свойства), започвайки от дадени обекти $\tilde{x} = (x_1, x_2, \ldots)$ (също с предварително определени свойства), чрез последователно прилагане на допустими операции. Всеки параметър на $\tilde{x}$ може да се мени в определени граници. Тяхното ограничаване до единствено възможно значение води до получаване на уникална (според стойностите на параметрите си) масова задача, която наричаме *екземпляр* на първоначалната масова задача.

Като примери на масови задачи и техни екземпляри можем да посочим:

1) *"Дадени са естественото число $q > 1$ и елементите a, b и c на крайното поле $GF(q)$. Да се намерят всички $x \in GF(q)$ такива, че $ax^2 + bx + c = 0$."* и *"Дадени са елементите a, b, c и x на крайното поле $GF(q)$. Да се провери дали $ax^2 + bx + c = 0$."* са масови задачи, а

"Да се намерят всички елементи $x \in GF(5)$ такива, че $x^2 - 3x + 2 = 0$." и *"Да се провери дали за $x = 2$ в полето $GF(5)$ е в сила $x^2 - 3x + 2 = 0$."* са съответно екземпляри на тези масови задачи.

2) *"Дадени са естествените числа i и j. Да се намери НОД на i и j."* и *"Дадени са естествените числа i, j и k. Да се провери дали k е НОД на i и j."* са масови задачи, а

"Да се намери НОД на 45 *и* 27*."* и *"Да се провери дали* 9 *е НОД на* 45 *и* 27*."* са съответно екземпляри на тези масови задачи.

3) *"Даден ъгъл* α *да се раздели на три равни части само с линийка (без деления) и пергел"* е масова задача, известна от древността като *задача за трисекцията на ъгъл.* Когато за ъгъл α е дадена конкретна стойност, се получава екземпляр на тази масова задача.

Двойките масови задачи от първите два примера си приличат - Дьорд Пойа нарича първите *задачи за намиране* (на решение), а вторите – *задачи за доказателство.* За да се избегнат недоразуменията, произтичащи от масови задачи, в които трябва да се докаже нещо ("Да се докаже, че ..."), т. е. да се намери решение, задачите от втория тип ще наричаме *задачи за проверка* (на решение). Ако сравним задачите от първия пример със задачите от втория пример ще забележим още една съществена разлика. В задачите от втория пример се търси не просто общ делител на двете числа, а най-голям общ делител, т. е. търси се решение с екстремални свойства спрямо останалите решения. Такива масови задачи ще наричаме *задачи за оптимизация.* При много от тях намирането на оптималното решение не може да стане в рамките на определен интервал от време и/или е достатъчно да бъде намерено някакво решение, близко до оптималното. Тогава се въвежда нов параметър p, който да е в лесно проверимо отношение с даден параметър на възможните решения. Търсенето спира, когато параметърът на поредното решение стане по-добър от p. Чрез тази техника масовите задачи от втория пример се трансформират в: *"Дадени са естествените числа* i, j *и* p*. Да се намери общ делител* k *на* i *и* j*, такъв че* $k > p$*."* и *"Дадени са естествените числа* i, j, k *и* p*. Да се провери дали* k *е общ делител на* i *и* j*, такъв че* $k > p$*."*

За да можем да изследваме с математически средства понятието масова задача, е нужен математически формализъм, който достатъчно добре да отразява същността на това понятие. Такъв формализъм може да бъде понятието функция. Ще разглеждаме само масови задачи с изброимо много екземпляри. Нека A е крайна азбука, която позволява да представим всеки екземпляр $\tilde{x}$ на масовата задача π за намиране на решение и съответния резултат $\tilde{y}$ чрез думи $\alpha_{\tilde{x}}$ и $\alpha_{\tilde{y}}$ над A. Масовата задача π можем да представим чрез функцията $f_\pi : A^* \to A^*$, такава че $f_\pi(\alpha_{\tilde{x}}) = \alpha_{\tilde{y}}$, за всички екземпляри $\tilde{x}$ на π. Ако π е задача за проверка на решение, можем да я представим чрез функцията $f_\pi : A^* \to \{true, false\}$, такава че $f_\pi(\alpha_{\tilde{x}}) = true$ когато проверката е успешна и $f_\pi(\alpha_{\tilde{x}}) = false$ в противен случай, за всеки екземпляр $\tilde{x}$ на π. Това позволява задачите за намиране на решение (от първия случай) да наречем "задачи за изчисляване на функция", а задачите за проверка на решение (от втория случай) да наречем "задачи за разпознаване на език", които очевидно са частен случай на първите.

Изчисляването на $f_\pi(\alpha_{\tilde{x}})$ по даден екземпляр $\tilde{x}$ на π с помощта на последователност от допустими операции се явява математически модел на неформалното понятие "решаване на задача".

Масовите задачи от първите два примера могат да бъдат решени с помощта на добре известни процедури (последователности от краен брой

допустими операции), независимо от това кой от екземплярите на съответната задача е зададен. В първия пример процедурата се определя от формулата за намиране корените на квадратно уравнение, във втория пример процедурата е известният алгоритъм на Евклид. Съответните задачи за разпознаване на език се решават чрез елементарни проверки. За масовата задача от третия пример подобна процедура не е била открита в продължение на повече от 20 века, докато през 1837 Ванцел доказва невъзможността да съществува такава процедура. Въпреки това, за отделни екземпляри на задачата (при $\alpha = \pi/(2^n)$, $n = 0, 1, \ldots$) отдавна са известни процедури, които ги решават.

Процедурите, които решават масови задачи, независимо от това кой екземпляр на масовата задача е даден, наричаме *алгоритми* (в чест на ал Хорезми[1]). Масова задача, за която съществува алгоритъм, който я решава, ще наричаме *алгоритмично разрешима*, а в противен случай – *алгоритмично неразрешима*. Тази класификация не изчерпва всички масови задачи – има масови задачи, за които не се знае дали са алгоритмично разрешими или не. Класификацията на масовите задачи се пренася и върху съответните им моделиращи ги функции. Функциите, съответстващи на алгоритмично разрешимите задачи, се наричат *ефективно (алгоритмично) изчислими* функции. Могат да бъдат посочени алгоритмични процедури за цели класове от функции и по този начин да се установи тяхната ефективна изчислимост, т. е. алгоритмичната разрешимост на съответните им масови задачи. Някои задачи могат да бъдат решавани само в рамките на ограничена съвкупност от операции (или средства) – за тях казваме, че те са разрешими само в рамките на тази тясна съвкупност. Така например масовата задача от третия пример става разрешима в рамките на по-ограничената (но и по-силна) съвкупност от линийка с деления и пергел.

След казаното дотук възниква основният въпрос какъв формализъм за изчисление на функции да се избере, така че с голяма степен на сигурност да можем да твърдим, че множеството на изчислимите в рамките на този формализъм функции съвпада с множеството на ефективно изчислимите функции. Математиката предлага няколко такива формализма, възникнали независимо един от друг през 30-те и 40-те години на миналия век във връзка с решаването на знаменития и широкообхватен *десети проблем на Хилберт* (известен като *Entscheindungsproblem*, поставен през 1900 г. на международния конгрес на математиците в Париж, уточнен през 1928 г. на конгреса им в Болоня [17]). Формулировката му гласи: "Съществува ли механична процедура, с помощта на която да се получават отговори на всички математически задачи, принадлежащи на широк, но добре дефиниран клас?".

[1] Абу Джафар Мохамед Ибн Муса Ал-Хорезми (787 – ок. 850) – средноазиатски математик и астроном. Автор на аритметичен трактат, преведен на латински през XII в., чрез който Европа се запознава с индийската позиционна бройна система. В труда му *"Kitab al jabr w'al-muqabala"* ("алгебра" произхожда от името му) за първи път алгебрата се разглежда като самостоятелен дял от математиката. Латинизираното на ал Хорезми - *Algorithmi* е станало нарицателно за изчисления, изпълнявани по строго определени правила

2.2 Машини на Тюринг

В този раздел е представена най-непосредствената, най-убедителната и най-важната в исторически план идея, чрез която се дава отговор на Хилбертовия *Entscheindungsproblem*. Някои от дадените понятия с редица коментари и примери са разгледани обстойно в [17], макар и в по-популярен и по-неформален стил.

Математическият модел *машини на Тюринг*[2] е общо наименование на група абстрактни машини, сходни по предназначение, устройство и начин на действие. Сред тях от най-прост вид е *машината на Тюринг (МТ) с безкрайна в едната посока лента*, която се състои от и работи както следва:

1) безкрайно в едната посока (например надясно) устройство за четене и запис, наречено *входно-изходна лента*. Тя е разделена на клетки, номерирани последователно с естествени числа. На тази лента се записват думи над някаква крайна *входно-изходна азбука*, по една буква в клетка. Във входно-изходната азбука е отделен специален символ $\flat$, наречен *бленк*, чрез който се маркират (запълват) празните клетки на лентата. Приемаме, че всяка входна дума, записана върху лентата, е крайна, че започва от първата ѝ клетка и е разположена надясно до последната клетка, различна от бленк.

2) *четящо-записваща глава*, която във всеки момент от работата на машината се намира върху определена клетка от лентата и може да прочете буквата, записана в нея, да запише буква в нея и евентуално да се премести върху съседна клетка. Първоначално главата е позиционирана върху първата клетка от лентата;

3) крайно множество от състояния, наречени *вътрешни състояния*, едно от които е определено за *начално*, а част от останалите са отделени като *заключителни (крайни)* или още *stop*-състояния. Машината започва работа винаги от началното си състояние, във всеки момент се намира в едно от вътрешните си състояния, а когато попадне в заключително състояние тя спира да работи;

4) машината работи на дискретни интервали от време, наречени *тактове*, като работата ѝ се управлява от *функция на преходите* или *програма*. На всеки такт машината повтаря следното: според входната буква от текущата клетка (над която се намира главата) и вътрешното състояние, в което се намира машината, функцията на преходите определя коя буква да се запише в текущата клетка, в кое вътрешно състояние да премине машината и (евентуално) в коя съседна клетка да се премести главата. Означаваме с L, R и S съответно преместването на главата една клетка наляво, една клетка надясно и оставането ѝ в същата клетка.

МТ с безкрайна в едната посока лента е представена схематично на Фиг. 2.1, а нейната формална дефиниция е:

[2] наречени така в чест на създателя им – английският математик, логик, "кодоразбивач" и информатик Алан Тюринг (1912–1954), автор на идеята за абстрактни изчислителни машини (1935–1936 г.), чрез които формализира и уточнява понятието алгоритъм.

При работа на МТ M върху дадена дума α са възможни няколко ситуации. Първата е *M да спре по естествен начин*, достигайки заключително състояние – тогава казваме кратко, че *M спира*. Втората е *да спре по неопределеност*, т. е. да достигне до входна буква и състояние, за които функцията δ не е дефинирана. Третата ситуация е *M да спре при опит за преместване на главата вляво от най-лявата клетка*. И четвъртата е *M да зацикли* – например ако в редицата от непосредствени преминавания се получи повтаряне на едни и същи конфигурации.

Сред възможните видове изчисления с МТ най-важни са определените чрез следната

Дефиниция 2.5 1) Нека M е МТ. Нека функцията $f : X^* \to X^*$ е дефинирана така, че за произволна лентова дума α, $f(\alpha) = \beta$, ако за α има спиращо по естествен начин изчисление $(\epsilon, q_0, \alpha) \models (\beta', q, \beta'')$, при което върху лентата остава думата $\beta = \beta'\beta''$, или $f(\alpha)$ е неопределена, ако M спира по друг начин или зацикля. Тогава функцията f наричаме *изчислима по Тюринг*.

2) Нека $F = \{q_y, q_n\}$. Дефинираме език, *разпознаван* от МТ M като множеството от думи $L_M \subseteq X^*$, такова че $\forall\alpha \in L_M$, $(\epsilon, q_0, \alpha) \models (\alpha', q_y, \alpha'')$ и $\forall\beta \notin L_M$, $(\epsilon, q_0, \beta) \models (\beta', q_n, \beta'')$.

3) Нека $F = \{q_y, q_n\}$. Дефинираме език, *допускан* от МТ M като множеството от всички думи $L_M \subseteq X^*$, такова че $\forall\alpha \in L_M$, $(\epsilon, q_0, \alpha) \models (\alpha', q_y, \alpha'')$ и $\forall\beta \notin L_M$, $(\epsilon, q_0, \beta) \models (\beta', q_n, \beta'')$, M спира по друг начин, или зацикля.

Нека M е МТ с входна азбука $X = \{x_1, x_2, \ldots, x_n\}$ и нейната функция на преходите е зададена чрез оператори от вида:

$$q_i)\ \delta(q_i, x_1);\ \delta(q_i, x_2);\ \ldots\ ;\ \delta(q_i, x_n)$$

за всяко незаключително състояние $q_i \in Q$ и

$$q_j)\ stop$$

за всяко заключително състояние $q_j \in F$. Тогава казваме, че M е зададена чрез *програма* (съдържаща оператори за представяне на незаключителни и заключителни състояния).

Ще разгледаме два примера, в които $X = \{0, 1\}$, като 0 е в ролята на бленк.

Пример 2.6 МТ M_1, зададена чрез програмата

$$\begin{array}{lll} q_0) & (q_1, 0, L) & ;\ (q_0, 1, R) \\ q_1) & - & ;\ (q_1, 0, L) \end{array}$$

спира винаги при опит да премести главата си вляво от най-лявата клетка. Ако α е лентова дума от вида $\alpha = 0^i\beta$, M_1 я оставя непроменена, а ако е от вида $\alpha = 1^i0\beta$, M_1 оставя на лентата думата $0^{i+1}\beta$. Следователно M_1 изчислява функцията

$$f_{M_1}(\alpha) = \begin{cases} 0^{i+1}\beta, & \text{ако } \alpha = 1^i0\beta \\ 0^i\beta, & \text{ако } \alpha = 0^i\beta. \end{cases}$$

Пример 2.7 МТ M_2, зададена чрез програмата

$$\begin{array}{lll} q_0) & (q_2, 0, S) & ; \ (q_1, 1, R) \\ q_1) & (q_3, 0, S) & ; \ (q_0, 1, R) \\ q_2) & stop & \\ q_3) & stop & \end{array}$$

завършва работа винаги в едно от състоянията q_2 или q_3. В първия случай това става за думи от вида $1^{2k}0\beta$, $k = 0, 1, 2, \ldots$, а във втория случай – за думи от вида $1^{2k+1}0\beta$, $k = 0, 1, 2, \ldots$. Ако приемем $q_2 = q_y$ за допускащо заключително, а $q_3 = q_n$ за отхвърлящо заключително състояние, следва че M_2 разпознава езика $L_{M_2} = \{1^{2k}0\beta |\ \beta \in X^*, k = 0, 1, 2, \ldots\}$. Ако разменим ролите на q_2 и q_3, ще получим нова МТ M_2', която разпознава допълнението $L_{M_2'}$ на езика L_{M_2}, т. е. $L_{M_2'} = X^* \backslash L_{M_2}$. Оттук следва твърдението, че ако езикът $L \subseteq X^*$ се разпознава от МТ M, тогава $\exists$ МТ M', разпознаваща езика $X^* \backslash L$.

Сред разновидностите (обобщенията) на класическата МТ с безкрайна в едната посока лента най-известни са: МТ с безкрайна в двете посоки лента, k-лентовата МТ, недетерминираната МТ, на които няма да се спираме. Доказано е, че изчисляваните чрез тях функции (допусканите/разпознаваните от тях езици) съвпадат с изчисляваните от класическата МТ функции (допусканите/разпознаваните от нея езици) и в този смисъл те са еквивалентни помежду си. Всички опити за разширяване възможностите на споменатите МТ са довели до конструкции, еквивалентни с класическата МТ. Доказано е още, че класическата МТ с произволна входна азбука е еквивалентна с друга такава, чиято входна азбука е $X = \{0, 1\}$ (вж. [16, 17]).

Както споменахме в края на предишния раздел, съществуват и редица други формализми за дефиниране на неформалното понятие алгоритъм, като *λ-смятането* (или *λ-анализът*, създаден от Алонсо Чърч с помощта на Ст. Клини, предложен за първи път около 1930 г.), *частично-рекурсивните функции* (на Ст. Клини), *машините на Пост* (въведени от Е. Пост, подобни на МТ), *нормалните алгоритми на Марков* (на А. А. Марков). Доказана е еквивалентността на тези формализми с МТ, т. е. че функциите, които тези модели пораждат (или изчисляват), съвпадат с изчисляваните по Тюринг функции. Това дава сериозно основание да се формулира следващото неформално твърдение (през 1937 г. Тюринг и Чърч, независимо един от друг, показват еквивалентността на предложените от тях формални модели):

Тезис на Тюринг-Чърч (1937): *Множеството на ефективно (алгоритмично) изчислимите функции съвпада с множеството на изчислимите по Тюринг функции.*

Това твърдение не може да бъде доказвано поради използваното в него неформално понятие "ефективно (алгоритмично) изчислими функции", но в негова подкрепа могат да се посочат редица доводи, като някои прилики между реалното програмиране и създаването на програми за МТ, или т. нар. *универсална машина на Тюринг*, способна да имитира вър-

ху себе си работата на произволна друга МТ (т. е. върху произволна лентова дума α двете машини работят по един и същи начин) и явяваща се математически модел и функционален първообраз на съвременните компютри.

Един по-сложен алгоритъм може да включва в себе си други алгоритми (подалгоритми), които да се реализират от подпрограми в една реална компютърна програма. По-сложни МТ също могат да бъдат създавани чрез композиция на вече готови такива. Нека M_1 и M_2 са МТ, зададени чрез техните програми, които изчисляват съответно функциите f_{M_1} и f_{M_2}. Без ограничение на общността ще считаме, че азбуката им е една и съща $X = \{0, 1\}$, че състоянията им са номерирани последователно $q_0, q_1, q_2, \ldots$, като q_0 е начално състояние и при двете машини, q_f е единственото заключително състояние на M_1 и при нея q_n е състоянието с най-голям номер. Композираме двете програми: първо записваме програмата на M_1, заменяйки оператора ѝ $q_f)$ *stop* с оператора $q_f)$ $(q_{n+1}, 0, S)$; $(q_{n+1}, 1, S)$, а после добавяме програмата на M_2, в която преномерираме всички състояния, добавяйки $(n+1)$ към номерата им. Така получаваме нова МТ M, която при достигане на състояние q_f не спира, а продължава да работи така, както би работила M_2 върху резултата от работата на M_1 (замененият оператор q_f изпълнява ролята на преход към началното състояние на M_2, преномерирано от q_0 в q_{n+1}). Очевидно е, че M изчислява композицията $f_M = f_{M_1} \circ f_{M_2}$. Така доказахме следващото твърдение, което е още един аргумент в подкрепа на тезиса на Тюринг-Чърч.

Лема 2.8 *Ако f_{M_1} и f_{M_2} са изчислими по Тюринг функции, тогава композицията им $f_M = f_{M_1} \circ f_{M_2}$ също е изчислима по Тюринг.*

Приемайки верността на тезиса на Тюринг-Чърч, по-нататък **под алгоритъм формално ще разбираме машина на Тюринг**.

В края на този раздел ще спомена някои важни факти (по-подробно те са разгледани в [16, 21, 37]).

Теорема 2.9 *Множеството от машините на Тюринг е изброимо.*

Както споменахме в края на раздел 1.1, множеството от всички езици над крайна азбука не е изброимо. Като следствие от този факт и от Теорема 2.9 получаваме:

Теорема 2.10 *Съществува език над крайна азбука, който не се разпознава от никоя МТ.*

Това означава, че съществуват масови задачи, които не са разрешими алгоритмично. Най-известна сред тях е задачата "По дадена МТ M и лентова дума α да се определи дали M спира (в заключително състояние или по друг начин) или зацикля при работа върху α.", наречена *стоп-проблем за машини на Тюринг*. Следващата теорема (доказана от Тюринг, както и от Чърч, но с по-различен апарат) утвърждава, че стоп-проблемът не е разрешим алгоритмично.

Теорема 2.11 *Не съществува МТ, разпознаваща езика от всички двойки думи* (M, α), *такива че МТ* M *спира при работа върху лентовата дума* α.

Следователно не съществува общ алгоритъм за решаване на всички математически задачи и Хилбертовият *Entscheindungsproblem* получава отрицателен отговор.

Следващата дефиниция дава едно мощно средство за доказване на някои твърдения, свързани с алгоритми.

Дефиниция 2.12 Функцията $f : A^* \to A^*$ наричаме *алгоритмично сводима* към функцията $g : B^* \to B^*$, ако съществува изчислима по Тюринг биекция $\varphi : A^* \to B^*$, такава че φ^{-1} е също изчислима по Тюринг и при това $f(\alpha) = \varphi^{-1}(g(\varphi(\alpha)))$. При разпознаване на език аналогичната дефиниция е: функцията $f : A^* \to \{true, false\}$ наричаме *алгоритмично сводима* към функцията $g : B^* \to \{true, false\}$, ако съществува изчислима по Тюринг биекция $\varphi : A^* \to B^*$, такава че $f(\alpha) = g(\varphi(\alpha))$.

С помощта на алгоритмичната сводимост можем да докажем неразрешимостта на масовата задача π_g, като сведем алгоритмично към функцията ѝ g функцията f на известна неразрешима масова задача π_f. Например, ако допуснем, че π_g е разрешима (т. е. g е изчислима по Тюринг), ще се окаже, че и $f = \varphi^{-1} \circ g \circ \varphi$ е изчислима по Тюринг, което е в противоречие с неразрешимостта на f.

2.3 Сложност на алгоритми и масови задачи

След като веднъж е установена разрешимостта на една масова задача, възниква важният въпрос за цената (като разход на изчислителни ресурси) за получаването на решение на отделните ѝ екземпляри, или на самата масова задача като цяло. Този въпрос има редица аспекти и е предмет на специална математическа теория, която ще представим накратко.

Когато една и съща масова задача може да бъде решавана чрез два или повече алгоритъма, са нужни критерии, по които да бъдат оценявани качествата на тези алгоритми с цел да бъде избран "най-ефективният" от тях. Тези критерии трябва да отразяват обективно същността на даден алгоритъм, а не особеностите на езиците за програмиране, чрез които той се реализира или параметрите на компютърните системи, върху които той се изпълнява. В този смисъл ще оценяваме отделните алгоритми според разходите на време и на памет, необходими за изпълнението им. Ограниченията по време са се утвърдили като доминиращ фактор и затова времето за изпълнение е основният критерий за ефективност на алгоритмите. Често се отчита и изразходваната памет, а понякога може да се вземат предвид и някои по-субективни критерии като яснота, простота за реализиране, големина на кода и др.

Удобно е времето за работа на един алгоритъм да се изразява като функция от големината на входа му, т. е. от обема на входните данни, описващи даден екземпляр на масова задача. За тези данни ще приемем изискването за "разумното" им представяне чрез думи над дадена азбука.

Това означава думата, представяща екземпляра на задачата да съдържа всичко необходимо за решаването ѝ и да няма излишък на информация (т. е. информация, при премахването на която същността на екземпляра не се променя).

Дефиниция 2.13 Нека $f_M : X^* \to X^*$ е тотална изчислима по Тюринг функция. Функцията

$$t_M(n) = \max_{\alpha \in X^*,\ d(\alpha)=n} [\text{ брой тактове на } M \text{ при работа върху } \alpha]$$

наричаме *сложност по време на M в най-лошия случай*, а функцията

$$\tilde{t}_M(n) = \frac{\sum_{\alpha \in X^*,\ d(\alpha)=n} [\text{ брой тактове на } M \text{ при работа върху } \alpha]}{|X^n|}$$

наричаме *средна сложност по време на МТ M*. Аналогично, за сложност по памет дефинираме функциите:

$$s_M(n) = \max_{\alpha \in X^*,\ d(\alpha)=n} [\text{ брой клетки, ползвани от } M \text{ при работа върху } \alpha],$$

която наричаме *сложност по памет на M в най-лошия случай* и

$$\tilde{s}_M(n) = \frac{\sum_{\alpha \in X^*,\ d(\alpha)=n} [\text{ брой клетки, ползвани от } M \text{ при работа върху } \alpha]}{|X^n|},$$

която наричаме *средна сложност по памет на МТ M.*

Понятията *сложност по време (по памет) в най-добрия случай* се дефинират аналогично на тези в най-лошия случай, като вместо максимум се вземе минимум.

За сложността по време и по памет е в сила:

Лема 2.14 *За всяка МТ M и за всяко естествено число n, $s_M(n) \leq t_M(n)$*

Твърдението на лемата следва от факта, че за $t_M(n)$ такта не могат да бъдат използвани повече от $t_M(n)$ клетки от лентата на M.

Моделът МТ и дадените формални дефиниции имат принципно приложение при определяне на горни граници за време и памет, необходими за изпълнението на даден алгоритъм. Но те трудно могат да бъдат използвани за практически цели – между МТ и съвременните компютри има значителни различия, произтичащи най-вече от хипотетично безкрайната памет (като брой клетки върху лентата) на МТ от една страна и от ограничената памет на компютрите и големината на компютърните думи от друга страна. Стъпките (тактовете) при МТ са различни от стъпките, които изпълнява друго устройство, когато реализират един и същи алгоритъм. Ето защо сложността на даден алгоритъм е свързана и с вида на изчислителното устройство (реално или хипотетично), върху което той се изпълнява. За съжаление няма компютърен модел, който да е еднакво подходящ и удобен за всички случаи. Освен МТ, в [21] са разгледани още

два модела на изчислително устройство, наречено *абстрактен компютър*, с цел да се открие вътрешноприсъщата изчислителна сложност на различни задачи, да се докажат горни граници на тази сложност. Накратко, тези модели са:

1) *RAM-модел* (от Random Access Machine). Представлява модел на компютър с един акумулатор, с лента за вход и отделна лента за изход, с програма и с памет. Паметта се състои от неограничен брой регистри, способни да съхраняват произволно големи цели числа. Първият от тях е акумулатор и съхранява единия операнд и евентуалния резултат при някои операции. RAM-програмата се състои от инструкции, за които се предполага, че не се самомодифицират и затова тя не се съхранява в паметта. Типът на самите инструкции не е толкова съществен, те са подобни на основните инструкции в асемблерните езици при реалните компютри. Предполага се наличието на: аритметични инструкции (ADD, SUB, MUL, DIV), входно-изходни инструкции (READ, WRITE), инструкции за сравнения и преход (JGTZ, JZERO, JUMP), инструкции за работа с паметта (LOAD, STORE), за край (HALT). Всяка инструкция има три части: етикет, код на операция и операнд, като етикетът не е задължителен. При инструкциите за разклонения вместо операнд се задава етикет, а при HALT операнд липсва. Операндът може да бъде три вида: $=i$ – означава самото число i; цяло число i – означава съдържанието на регистъра с номер i; $*i$ – означава косвено адресиране, т. е. операнд е числото в регистъра с номер j, а j е числото, записано в регистъра с номер i.

2) *RASP-модел* (от Random Access Stored Program). Този модел е подобен на RAM-модела с тази разлика, че програмата се съхранява в регистрите на паметта (така тя може да се самомодифицира), а косвеното адресиране става излишно и не е позволено.

И двата модела функционират по начин, аналогичен на този при реалните компютри. При RAM и при RASP-модела, както при MT, се дефинира *сложност по време в най-лошия случай* (или само *сложност по време*) като максимум (по всички входни думи с дължина n) на функцията $f(n)$, равна на сумата от времената на всички изпълнени инструкции при работа върху входната дума. Аналогично се дефинират *средна сложност по време*, както и видовете сложности по памет (като се отчита броят на използваните регистри).

За да се определят коректно различните сложности, т. е. времето и паметта, необходими за изпълнението на всяка инструкция, при двата модела се въвеждат ценовите критерии [21]:

1) *критерий за еднаква цена* – при него всяка RAM или RASP инструкция се изпълнява за единица време и всеки регистър съдържа единица памет, независимо от големината на данните.

2) *критерий за логаритмична цена* – при него всяка RAM или RASP инструкция се изпълнява за време и използва памет, пропорционални на големината на операндите им. Отчита се фактът, че $\lfloor \log n \rfloor + 1$ бита са нужни за представянето на естественото число n в даден регистър.

Един и същи алгоритъм (програма) може да има коренно различни

цени спрямо двата критерия. Критерият за еднаква цена е приложим и приемлив при алгоритми, при които входните данни, междинните резултати и крайните резултати се побират изцяло в определен брой машинни думи с фиксирана големина. В противен случай вторият критерий е значително по-точен и реалистичен.

В [21] е доказано, че и при двата критерия времевите сложности на RAM-програма и RASP-програма, реализиращи един и същи алгоритъм, се различават с константен множител. Същият извод е валиден и за сложността по памет. Доказано е също, че изчисленията върху RAM (RASP)-модела и изчисленията върху МТ са полиномиално свързани, т. е.

Дефиниция 2.15 Функциите $f(n)$ и $g(n)$ са *полиномиално свързани*, ако съществуват полиноми $p_1(x)$ и $p_2(x)$, такива че $f(n) \leq p_1(g(n))$ и $g(n) \leq p_2(f(n))$, за всяко $n \in N$.

Макар и по-съвършени от МТ, RAM и RASP моделите също са доста примитивни машини, за да пресмятаме реални сложности в термините на техните инструкции. При реалните компютри и езиците от високо ниво се определя цена за изпълнение на всеки оператор (включваща и цените на подоперациите му) – според възприетия в случая критерий за еднаква или за логаритмична цена. Тогава различните видове сложности по време и по памет се дефинират по същия начин, както при RAM и RASP моделите.

2.4 Асимптотични означения и порядък на растеж при функциите на сложност

В общия случай функциите на сложност са растящи функции с особено поведение. Понякога техният явен вид, дори за прости алгоритми, се получава твърде трудно, а и усилието за получаването им е излишно. За практически нужди е достатъчно да работим не със самите функции, а с техни приближения, които се получават по-лесно и показват достатъчно ясно поведението на функцията на сложност на даден алгоритъм при нарастване големината на входа. Ще се интересуваме от порядъка на растеж, от горни и долни граници на асимптотичното нарастване на времето (или на паметта) за изпълнение на даден алгоритъм при увеличаване стойностите на входните данни.

Функциите на времева сложност се дефинират върху множеството от естествените числа N (дължината на входната дума е цяло число) и приемат също целочислени стойности (означаващи броя на изпълнените стъпки). За да се ползват техники от математическия анализ, кообластта им се разширява до множеството от реалните неотрицателни числа, т. е. R^+, а при нужда и дефиниционната им област може да бъде разширена до R^+. Означаваме с N_{R^+} множеството от функции $N_{R^+} = \{f | f : N \to R^+\}$.

Дефиниция 2.16 Нека е дадена функцията $g \in N_{R^+}$. За функцията $f \in N_{R^+}$ казваме, че:

а) f принадлежи на множеството функции $O(g)$, ако съществуват константи $c \in R^+$ и $n_0 \in N$, такива че $0 \leq f(n) \leq cg(n)$, за всяко $n > n_0$;

б) f принадлежи на множеството функции $\Omega(g)$, ако съществуват константи $c \in R^+$ и $n_0 \in N$, такива че $0 \leq cg(n) \leq f(n)$, за всяко $n > n_0$;

в) f принадлежи на множеството функции $\Theta(g)$, ако съществуват константи $c_1, c_2 \in R^+$ и константа $n_0 \in N$, такива че $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, за всяко $n > n_0$;

г) f принадлежи на множеството функции $o(g)$, ако за всяка константа $c \in R^+$ съществува константа $n_0 \in N$, такава че $0 \leq f(n) < cg(n)$, за всяко $n > n_0$;

д) f принадлежи на множеството функции $\omega(g)$, ако за всяка константа $c \in R^+$ съществува константа $n_0 \in N$, такава че $0 \leq cg(n) < f(n)$, за всяко $n > n_0$.

Всички означения от дефиницията са за асимптотични граници, т. е. описват поведението на функцията f спрямо друга функция g при достатъчно големи n. При първите три означения неравенствата са нестроги, при тях границата е асимптотично тясна, без да се определя точно колко при O и Ω. Последните две означения са дадени само за пълнота, понеже се използват рядко. При тях неравенствата са строги, границата не е асимптотично тясна. От дефиницията непосредствено следва:

Теорема 2.17 *За функциите f и g, $f \in \Theta(g)$ тогава и само тогава, когато $f \in O(g)$ и $f \in \Omega(g)$.*

Ако времевата сложност f на даден алгоритъм принадлежи на $O(g)$, ще казваме, че тя "е $O(g)$" и понякога е по-удобно да пишем $f = O(g)$ вместо $f \in O(g)$. Това важи и за останалите означения. Когато използваме асимптотично означение във формула, ще го интепретираме като анонимна функция от съответното множество, без да сме длъжни да я именоваме и указваме. Например записът $2n^2 + 5n - 3 = 2n^2 + \Theta(n)$ означава, че $2n^2 + 5n - 3 = 2n^2 + f(n)$, където f е някоя функция от множеството $\Theta(n)$. Ясно е, че такава функция, за която равенството става валидно, съществува в случая $f(n) = 5n - 3$. А записа $2n^2 + \Theta(n) = \Theta(n^2)$ интерпретираме според правилото: "Независимо как са избрани анонимните функции в лявата част на равенството, винаги могат да бъдат избрани анонимни функции в дясната част, така че равенството да се удовлетворява." [37].

Като пример да покажем, че $n^2 - 3n = \Theta(n^2)$. От неравенствата $c_1 n^2 \leq n^2 - 3n \leq c_2 n^2$, като разделим на n^2, получаваме $c_1 \leq 1 - 3/n \leq c_2$. Ако изберем $c_1 = 1/2$, лявото неравенство е в сила при $n \geq 6$ и следователно $n^2 - 3n = \Omega(n^2)$. Можем да изберем $c_2 = 1$ и тогава дясното неравенство е в сила за всяко $n > 0$ – следователно $n^2 - 3n = O(n^2)$. От Дефиниция 2.16 в) и от Теорема 2.17 следва, че $n^2 - 3n = \Theta(n^2)$. Аналогично може да се покаже, че $4n^3 \neq \Theta(n^2)$ – понеже не съществува константа c_2, такава че $n \leq c_2/4$ за всяко достатъчно голямо n. Но $4n^3 = O(n^3)$, $4n^3 = O(n^4)$ или $4n^3 = \Omega(n)$. С тези примери още веднъж напомняме, че знакът "=" е само заместител на знака за принадлежност към множество "$\in$". И още,

че записите $f = O(g)$, $f = \Omega(g)$ означават, че g, умножена по някоя константа $c > 0$, е просто асимптотична (горна или долна) граница на f, без ограничение колко тясна е тя. Разграничаването на асимптотични граници (O и Ω-означението) от асимптотично тесни граници (Θ-означението) напоследък се утвърждава като стандарт в литературата по алгоритми [37].

O-означението се използва за означаване на времева сложност в най-лошия случай, Ω-означението – за означаване на времева сложност в най-добрия случай. А когато се използва Θ-означението, това значи че горната и долната граница (или времевата сложност в най-лошия и тази в най-добрия случай) се различават с константен множител.

Много от свойствата, свързани с релации между реални числа, се пренасят и при асимптотичните означения.

Лема 2.18 *За функциите $f, g, h \in N_{R^+}$ са в сила свойствата:*

1) рефлексивност: $f = O(f)$, $f = \Theta(f)$ и $f = \Omega(f)$;

2) симетричност: $f = \Theta(g) \iff g = \Theta(f)$;

3) транзитивност: от $f = O(g)$ и $g = O(h)$ следва $f = O(h)$, това е валидно и за означенията Θ, Ω;

4) кососиметричност: $f = O(g) \iff g = \Omega(f)$.

В допълнение, за означението O (респ. Ω), разглеждано като релация над $N_{R^+} \times N_{R^+}$, ще кажем, че тя не е симетрична (например $n = O(n^2)$, но $n^2 \neq O(n)$), нито антисиметрична (например $n^2 + 5n + 6 = O(n^2)$ и $n^2 = O(n^2 + 5n + 6)$, но $n^2 \neq n^2 + 5n + 6$).

Нека $R_{\asymp} \subseteq N_{R^+} \times N_{R^+}$ е релацията $R_{\asymp} = \{(f, g) | f, g \in N_{R^+}, f \in \Theta(g)\}$. От Лема 2.18 следва, че $R_{\asymp}$ е релация на еквивалентност и следователно тя разбива множеството N_{R^+} на класове на еквивалентност. За техни представители избираме по-прости и характерни функции – например $1 = \Theta(n^0), \log_2 n$, $n, n.\log_2 n, n^2, \ldots, 2^n \ldots$. Класът на еквивалентност на функцията $f \in N_{R^+}$ бележим с $\Theta(f)$ и го наричаме *порядък на растеж* на f.

При реалните числа е в сила свойството "трихотомия" – $\forall a, b \in R$ е изпълнено $a < b$, или $a = b$, или $a > b$, докато за функциите в N_{R^+} не можем да кажем, че всички те са асимптотично сравними. Например, ако $f(n) = n$ и $g(n) = n^{1+\sin n}$, тогава нито $f = O(g)$, нито $g = O(f)$.

Нека $\mathcal{C}$ е множеството от всички класове на еквивалентност на релацията $R_{\asymp}$. Релацията $R_{\preceq} \subseteq \mathcal{C} \times \mathcal{C}$ дефинираме като $R_{\preceq} = \{(\Theta(f), \Theta(g)) | f = O(g)\}$. Когато $A \preceq B$ и $A \neq B$, пишем $A \prec B$. Съгласно Лема 2.18 и поясненията след нея $R_{\preceq}$ е релация на частична наредба. Според това, което вече казахме за асимптотичната сравнимост на функции, тази наредба не е пълна. Но за класовете, които са в релация, наредбата означава, че можем да сравняваме порядъците на растеж на сложностите на различни алгоритми за решаване на една и съща задача. Например, при сортировка на масиви са популярни два вида алгоритми: едните със сложност $\Theta(n \log n)$, а другите – със сложност $\Theta(n^2)$. Понеже $\Theta(n \log n) \prec \Theta(n^2)$, алгоритмите от първия вид са за предпочитане.

Лема 2.19 *За всяко $k \in N$ е в сила $\Theta(n^k) \prec \Theta(2^n)$.*

Лемата ни дава основание да разделим алгоритмите според времевата им сложност на две основни категории, представени в следващия раздел.

2.5 Полиномиално разрешими и полиномиално проверими задачи

Полиномиален алгоритъм наричаме алгоритъм, чиято времева сложност е $O(n^k)$ за някоя константа $k \in N$. Алгоритъм, за чиято времева сложност не е валидна подобна оценка, ще наричаме най-общо "експоненциален". Повечето експоненциални алгоритми са варианти на схемата "пълно изчерпване". Обикновено полиномиален алгоритъм може да се построи тогава, когато успеем да проникнем по-дълбоко в същността на задачата – най-вече с помощта на математически средства [8].

Една задача наричаме *полиномиално разрешима*, ако съществува полиномиален алгоритъм, който я решава. Прието е такава задача да се счита за "лесно решима", а ако за нея не е известен полиномиален алгоритъм, който да я решава, да се счита за "трудно решима". Едни от първите резултати по "трудна решимост" са класическите резултати по неразрешимост за машини на Тюринг.

Различието между двата типа алгоритми става особено забележимо, когато нараства големината на входа, съгласно Лема 2.19. И все пак тази класификация крие редица условности и не бива да се приема като догма. По-важните фактори, които трябва да се отчитат, са:

1) времевата сложност изразява поведението на даден алгоритъм в най-лошия случай. Възможно е съответният времеви максимум да се достига при сравнително малко на брой екземпляри на дадена масова задача, докато за останалите екземпляри разходът на време да бъде от друг порядък или в напълно приемливи граници. Например, доказано е, че симплекс-методът за решаване задачата на линейното програмиране има експоненциална времева сложност, но за практически цели той работи достатъчно добре;

2) при избор на алгоритъм трябва да се има предвид реалната големина на входа, за която той ще се ползва. Например, за функциите $f(n) = 2^n$ и $g(n) = 5n^4$ имаме $f(n) < g(n)$ при $n < 20$. Ако подобни функции изразяват времевите сложности на два алгоритъма, разумно е да се пресметне граничната стойност и според големината на очаквания вход да се прецени кой от двата алгоритъма да се използва;

3) полиномиални алгоритми с времева сложност от типа n^{100} или $10^{99}n$ няма да считаме за ефективни от практическа гледна точка. Полиномиалните алгоритми, които решават естествено възникнали практически задачи, обикновено имат времева сложност от порядък n^2 и n^3, като коефициентите пред старшите членове на съответните полиноми не са много големи;

4) при сравняване времевите сложности на два алгоритъма трябва да се отчитат и константите, скрити в съответните означения. Може да се окаже, че алгоритъм с времева сложност $\Theta(n^3)$ (с малка константа, скрита

Наименованието на класа $\mathcal{NP}$ произтича от *Nondeterministically Polynomial* и е свързано с второто важно понятие – *недетерминиран алгоритъм.* Това е алгоритъм, който работи на два стадия: *стадий на отгатване* и *стадий на проверка.* По даден екземпляр на масова задача на първия стадий се изпълнява просто "отгатване" на някаква структура. Описанието на екземпляра и отгатнатата структура се подават като вход на стадия за проверка, която се изпълнява по детерминиран начин и завършва с отговор "да" или "не". Формален модел на понятието недетерминиран алгоритъм е *недетерминираната еднолентова машина на Тюринг* (НМТ) с безкрайна в двете посоки лента. Тя е подобна на класическата еднолентова МТ, но за разлика от нея е снабдена с хипотетичен отгатващ модул. Той има собствена глава, чрез която може само да записва произволна догадка (кандидат за решение) върху лентата. Отгатващият модул има недетерминирано поведение: започва работа от клетката с номер -1 и управлява отгатващата глава. На всеки такт тя или записва произволна буква от азбуката и се премества една клетка вляво, или спира, като отгатващият модул преминава в пасивно състояние (решавайки това произволно). Така приключва стадият на отгатване и се стартира стадият проверка. Тя се извършва детерминирано, по познатия вече начин за разпознаване на език от класическата МТ [8].

Полиномиалната проверимост на една задача се установява сравнително лесно. Да вземем например задачата за намиране на Хамилтонов цикъл в граф. Досега не е известен полиномиален алгоритъм, който да я решава. Но проверката дали дадена последователност от върхове образува Хамилтонов цикъл е елементарна. Първо проверяваме дали върховете на графа участват в последователността точно по веднъж, с изключение на началото и краят, които трябва да съвпадат. Ако "да", проверяваме дали всеки два съседни върха в тази последователност са свързани с ребро. И двете проверки могат да се извършат за време, пропорционално на броя на върховете в дадения граф.

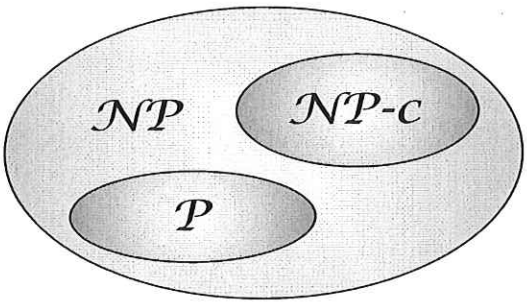
Въпросът за взаимното разположение на класовете $\mathcal{P}$ и $\mathcal{NP}$ е от фундаментално значение в теорията на $\mathcal{NP}$-пълните задачи. Очевидно е, че $\mathcal{P} \subseteq \mathcal{NP}$. За да покажем полиномиалната проверимост на произволна задача от $\mathcal{P}$, е достатъчно само да я решим за полиномиално време. Стадият отгатване се игнорира, а получаването на решение става в стадия за проверка, т. е. вместо проверка детерминираният полиномиален алгоритъм просто решава задачата и проверката за решение става излишна. Основният въпрос е *дали $\mathcal{P}$ е същинско подмножество на $\mathcal{NP}$ или не.* Засега и двете хипотези са възможни и имат свои привърженици и аргументи. Само от практически съображения по-вероятно е $\mathcal{P} \subset \mathcal{NP}$ и това се приема от повечето изследователи. Затова всички по-нататъшни разглеждания и изводи относно класовете ще бъдат направени при предположението, че $\mathcal{P} \neq \mathcal{NP}$.

2.6 $\mathcal{NP}$-пълни задачи

Още една много основателна причина да вярваме, че $\mathcal{P} \neq \mathcal{NP}$ е съществуването на класа от т. нар. "$\mathcal{NP}$-пълни" задачи. Те имат забележителното свойство, че ако само една задача от този клас може да бъде решена за полиномиално време, тогава всяка задача от $\mathcal{NP}$ също има полиномиално решение, т. е. $\mathcal{P} = \mathcal{NP}$. Или обратно: ако за някоя задача от $\mathcal{NP}$ се докаже, че тя не е полиномиално разрешима, тогава всяка $\mathcal{NP}$-пълна задача също не е полиномиално разрешима. В този смисъл $\mathcal{NP}$-пълните задачи са "най-трудните" задачи в класа $\mathcal{NP}$. При определянето им основна роля играе полиномиалната сводимост. Тя показва, че една задача е поне толкова трудна, колкото е трудна друга задача, или, че сложностите им са полиномиално свързани.

Дефиниция 2.25 Задача f, за която:
а) $f \in \mathcal{NP}$;
б) $\forall\ g \in \mathcal{NP},\ g \propto f$,
наричаме $\mathcal{NP}$-пълна. Класа на $\mathcal{NP}$-пълните задачи означаваме с $\mathcal{NP}$-c.

Взаимното разположение на класовете $\mathcal{P}$, $\mathcal{NP}$ и $\mathcal{NP}$-c е представено на Фиг. 2.2, при предположението, че $\mathcal{P} \neq \mathcal{NP}$.



Фиг. 2.2: Взаимно разположение на класовете на сложност, ако $\mathcal{P} \neq \mathcal{NP}$.

Лема 2.26 *Ако* $f, g \in \mathcal{NP}$, $f \in \mathcal{NP}$-c *и* $f \propto g$, *то* $g \in \mathcal{NP}$-c.
Доказателство. Понеже $g \in \mathcal{NP}$ остава да покажем, че за всяка $h \in \mathcal{NP}$ е изпълнено $h \propto g$. Но $\forall h \in \mathcal{NP}$ е в сила $h \propto f$ и от транзитивността на полиномиалната сводимост (Лема 2.23) следва, че $h \propto g$. $\diamond$

Твърдението на лемата означава, че ако имаме поне една $\mathcal{NP}$-пълна задача, тогава чрез полиномиално свеждане на тази задача към задачата $\pi \in S$ можем да покажем, че и π е $\mathcal{NP}$-пълна. Честта да бъде "**първата $\mathcal{NP}$-пълна задача**" и да постави основата на класа $\mathcal{NP}$-c се пада на задача от областта на булевите функции. Формулировката ѝ гласи: *"Дадена е булева функция $f(x_1, x_2, \ldots, x_n)$ в конюнктивна нормална форма. Съществуват ли значения на променливите $\sigma_1, \sigma_2, \ldots, \sigma_n$, такива че $f(\sigma_1, \sigma_2, \ldots, \sigma_n) = 1$?"*. Казваме, че векторът $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ *удовлетворява* f, ако $f(\sigma_1, \sigma_2, \ldots, \sigma_n) = 1$ и поради това тази задача се нарича *задача за удовлетворимост на булева функция* (УБФ).

Теорема 2.27 (С. Кук) *Задачата УБФ е $\mathcal{NP}$-пълна.*

Доказателствата на тази и на следващата теорема са твърде дълги, не се ползват в по-нататъшното изложение и затова не са дадени тук. За справка те могат да бъдат намерени в [16, 37, 8].

Както споменахме, след като вече е известна поне една $\mathcal{NP}$-пълна задача, с помощта на Лема 2.26 може да се докаже $\mathcal{NP}$-пълнотата и на редица други задачи. Сред тях особено важно място заема задача, подобна на УБФ, в която е наложено ограничението всяка елементарна дизюнкция в нейната конюнктивна нормална форма (КНФ) да има точно по 3 букви на променливи (с или без отрицания) – такава КНФ означаваме с 3-КНФ. Задачата *"Дадена е булева функция $f(x_1, x_2, \ldots, x_n)$ в 3-КНФ. Съществува ли вектор $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \{0,1\}^n$, който да удовлетворява f?"* се нарича *задача за 3-удовлетворимост на булева функция* (3-УБФ).

Теорема 2.28 *Задачата 3-УБФ е $\mathcal{NP}$-пълна.*

Доказателството, че тази или произволна друга задача за разпознаване π попада в класа $\mathcal{NP}$- c, включва четири стъпки:

1) доказателство, че $\pi \in \mathcal{NP}$;

2) избиране на подходяща $\mathcal{NP}$-пълна задача π';

3) построяване на функция f, свеждаща задачата π' към π;

4) доказателство, че извършваната от f сводимост става за полиномиално време.

По този начин е доказана $\mathcal{NP}$-пълнотата на стотици задачи, а голяма част от тях са класифицирани и представени в приложението на [8]. Класоветете на сложност, разгледани в тази глава, не изчерпват всички известни досега такива. Няколко други класове на сложност се дефинират в [8, 37].

Въпреки дългогодишните усилия на много изследователи, досега не е открит нито един полиномиален алгоритъм за решаване на коя да е $\mathcal{NP}$-пълна задача. В следващата глава ще разгледаме задачите УБФ и 3-УБФ, ще ги класифицираме и ще опитаме да установим дали те винаги са толкова трудни, т. е. дали и при колко от техните екземпляри има полиномиално решение.

трети раздел са представени изследванията ни върху ЗУ, както и резултатите, получени чрез комбинаторни и компютърни методи. Във втори раздел се дефинират еквивалентности над множеството от всички КНФ на n променливи. Спрямо тях се класифицират КНФ на $\tilde{0}$ с до 3 променливи. Изследванията от втори раздел са докладвани на Петата международна конференция по дискретна математика и приложения (1995 г.) и на XXVI Пролетна конференция на СМБ (1997 г.) и са публикувани в [56, 25]. Подобни резултати са получени по-късно и в други източници, споменати в раздел 3.1.4. В трети раздел се извеждат някои необходими и някои достатъчни условия за неудовлетворимост на 3-КНФ на n променливи, които са полиномиално проверими. Чрез тези условия са получени оценки за броя на удовлетворимите и на неудовлетворимите 3-КНФ. Резултатите са докладвани на XXIX Пролетна конференция на СМБ (2000 г.) и са публикувани в [26]. Резултатите, представени в тази глава, са получени съвместно с доц. д-р Красимир Манев.

3.1 Обзор на изследванията на задачите за удовлетворимост

3.1.1 Варианти на задачите за удовлетворимост

Задачите за удовлетворимост се изследват в различни варианти, които могат да бъдат класифицирани по няколко признака. Според броя на търсените решения това са задачите: SAT-ONE (самата задача УБФ или 3-УБФ), SAT-ALL (за намиране на множеството от всички вектори, които удовлетворяват дадена булева функция (БФ)) и SAT-COUNT (за намиране броя на векторите, удовлетворяващи дадена БФ) [66]. Според ограниченията върху КНФ на БФ имаме задачата УБФ (или SAT), в която няма ограничения за дължината на дизюнкциите в КНФ и най-общо задача k-УБФ (или k-SAT) – всички дизюнкции в КНФ са с дължина, равна на k [57]. Отбелязваме, че задачата 3-УБФ е частен случай на задачата k-УБФ и при $k > 3$ всяка задача за k-УБФ може да се преобразува в задача за 3-УБФ за линейно време [58], а задачата 2-УБФ е разрешима за полиномиално време [8, 47]. Други варианти на ЗУ се получават в зависимост от подходите за решаването им в конкретни области. В тях се използват различни начини за трансформиране и представяне на ЗУ, като [47]:

1) побитово представяне кандидатите за решение на дадена ЗУ, като с всяка променлива се свързва един бит. Това е един от най-естествените и ефективни начини за представяне, използван в много алгоритми;

2) представяне чрез плаваща точка. Предложено е през 1988 г. с цел преобразуването на дискретната задача УБФ в задача за оптимизация в непрекъснатия случай. На дискретните стойности 0 и 1 се съпоставят съответно -1 и 1. Дадена БФ $f : \{0,1\}^n \to \{0,1\}$ се трансформира по подходящ начин в непрекъсната функция $g : [-1,1]^n \to R$, която трябва да се минимизира;

3) представяне чрез клаузи и в термините на съждителното смятане.

Предложено е през 1995 г. от Хао. Използва се в различни алгоритми, най-вече в изкуствения интелект;

4) представяне чрез графи и/или в термините на теория на графите. Дадена задача за k-УБФ се трансформира в задачи над двуделни графи [62], над хиперграфи [51], в задачи за търсене на пътища, свързващи литерали (т. е. букви на променливи с или без отрицания) от различни клаузи (т. е. елементарни дизюнкции). Тук споменаваме и модела "диаграми на решенията" (ДР) и неговите разновидности "двоични ДР", "наредени двоични ДР", "наредени функционални ДР" [66, 64]. По същество, ДР представляват ориентирани (двоични) коренови дървета (поради това се използва и терминът "дърво на решенията" [37]), чиито върхове и ребра са надписани по определен начин, като при различните ДР са наложени различни допълнителни ограничения. Създадени първоначално като структури от данни за ефективно представяне на булеви функции, тези модели имат и редица други приложения.

3.1.2 Изследвания на задачите за удовлетворимост в изкуствения интелект

Както споменахме, една от областите, в която много усилено се изследват ЗУ, е изкуственият интелект. Разработените алгоритми за решаване на ЗУ могат да бъдат класифицирани най-общо като *пълни (точни)* и *непълни (евристични)* [47, 57]. Пълните алгоритми винаги дават точен отговор на произволен екземпляр на ЗУ, но имат експоненциална времева сложност в най-лошия случай. Непълните алгоритми се прилагат в ЗУ с наложени ограничения.

Ефективните примери от точните алгоритми са основани на популярната процедура на Дейвис и Патнам [44, 64], която оперира с формули в клаузна форма (еквивалент на КНФ). На входа ѝ се подава множество от клаузи Σ, всяка от които е дизюнкция от литерали, а всеки литерал е буква на променлива или нейно отрицание. Процедурата винаги определя дали дадената формула се удовлетворява или не, а псевдокодът ѝ изглежда така:

Procedure DP (Σ)
  if ($\Sigma = \emptyset$) **then return** *удовлетворима*
  (Тавтология) **if** (Σ съдържа клауза-тавтология c) **then return** DP ($\Sigma \backslash \{c\}$)
  (Празна клауза) **if** (Σ съдържа празна клауза) **then**
    return *неудовлетворима*
  (Клауза-литерал) **if** (Σ съдържа клауза-литерал $c = \{l\}$) **then**
    return DP (Σ, *опростена чрез присвояването* $l = True$)
  (Премахване на литерал) **if** (Σ *съдържа литерал* l *и не съдържа* $\bar{l}$) **then**
    return DP (Σ, *опростена чрез присвояването* $l = True$)
  (Разцепване) **if** (Σ, *опростена чрез присв.* $l = True$, *е удовлетворима*)
    then return *удовлетворима*
    esle return DP (Σ, *опростена чрез присвояването* $l = False$)

Празната клауза не съдържа нито един литерал, клаузата-литерал съ-

държа единствен литерал, а клаузата-тавтология съдържа едновременно литерал и отрицанието му. След присвояването на стойност *True* на литерала l множеството от клаузи се опростява, като от него се премахват всички клаузи, съдържащи l, а също така се изтрива и отрицанието на l във всички останали клаузи. Процедурата на Дейвис и Патнам е недетерминирана, тъй като в правилото за разцепване изборът на литерал не е определен. В някои от по-старите версии се избира първият литерал от първата клауза, а в следващите версии се прилагат евристични правила за избор [64].

Непълните алгоритми се прилагат при ЗУ с наложени ограничения (например върху комбинации от стойности на отделни групи от литерали) и се основават на подходи като *локално претърсване, еволюционни изчисления* и др. [57, 38]. В средата на 90-те години на миналия век усилено се разработват алгоритми, основани на локално претърсване и генериране на вектори – кандидати за решение. Това са варианти на т. нар. процедура GSAT, които намират решение на удовлетворим екземпляр на ЗУ, зададена в КНФ [41, 30]. Те оперират, като на всяка стъпка променят стойността на една променлива в кандидата за решение, така че да бъдат удовлетворени максимален брой клаузи. За целта екземплярът на ЗУ се трансформира в неотрицателна частична функция, която има глобален минимум 0, ако формулата е удовлетворима. В някои от версиите на GSAT с всяка клауза се асоциират тегла, които се модифицират по време на търсенето според зададени евристични правила [41]. Основни проблеми при тези алгоритми е откриването и избягването от локален минимум, а също и критериите за избор на променлива, чиято стойност да бъде променена. Освен алгоритмите от типа GSAT, на базата на локалното претърсване се развиват и т. нар. "генетични алгоритми", които създават, поддържат и ползват евристична информация чрез подходящи самонастройващи се функции и/или специфични и основаващи се на натрупани знания "генетични" оператори – за избор, за заместване, за кръстосване, мутиране и др. [57]. Еволюционните изчисления са в основата на т. нар. "еволюционни алгоритми", които се разделят условно на два класа. Едните ползват подходящи оценъчни функции – например (както при GSAT), в повечето еволюционни алгоритми се използва функция, която на даден вектор (кандидат за решение) съпоставя броя на клаузите (евентуално с приписани тегла), които той удовлетворява. Другите ползват евристични техники, свързани с локално търсене, самонастройване, "замразяване" на клаузи (литерали), вариране стойностите на литерали, оценки и избор и т. н. [47, 38]. Алгоритмите, които съчетават техники генетичните и еволюционните подходи, са т. нар. "хибридни алгоритми". Разработени са специални примери[1] (екземпляри на ЗУ с различен брой променливи, достигащи до няколкостотин), създадени са редица методи (теоретични, вероятностни, експериментални и др.) за тестване и оценка на ефективността на различните алгоритми [47, 44].

[1] ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances, http://www.in.tu-clausthal.de/~gottlieb/benchmarks/3sat

Теоретичните и експерименталните резултати показват добра средна времева сложност за определени класове от ЗУ. Важно е, обаче, да се знае дали и доколко останалите екземпляри на ЗУ са трудни и представителни спрямо тези, срещани в практиката. В [35] е показано, че не само ЗУ, но и редица други $\mathcal{NP}$-пълни задачи могат да бъдат обобщени с помощта на поне един параметър, като най-трудните екземпляри на съответните задачи се появяват при определена критична стойност на този параметър. Тя разделя пространството от екземпляри на две области с различни характеристични свойства: в едната са тези от тях, при които вероятността за съществуване на решение е почти единица, а в другата тази вероятност е почти нула. На границата между двете области с висока плътност са разположени екземпляри, съдържащи полурешения (дълбоки локални минимуми), явяващи се капани за алгоритмите за търсене. Тази граница се нарича *фазов преход* и се запазва при полиномиалната сводимост между $\mathcal{NP}$-пълните задачи, изследвани в [35]. При ЗУ този параметър се дефинира като отношението на броя клаузи към броя на променливите в съответния екземпляр, а фазов преход се получава при определени стойности на този параметър. Експериментите показват, че когато това отношение нараства, се получава моделът "лесни-трудни-лесни" и че в областите на фазов преход се появяват най-трудните екземпляри на ЗУ – те се идентифицират като най-трудни за решаване или за доказване, че са неудовлетворими [45, 35, 38].

3.1.3 Вероятностно-статистически методи и модели при изследване на задачите за удовлетворимост

При изследване на ЗУ широко се използват вероятностно-статистическите методи и подходи – най-вече за генериране на тестови примери и за оценка резултатите от изпълнението на различни алгоритми. В [45, 44] (в някои други източници, както и в адресите, дадени в бележката под линия) се предлагат и изследват три вероятностни модела на ЗУ. В т. нар. "модел с постоянна вероятност" всяка клауза се генерира така, че да съдържа всеки от $2N$ възможни литерала (N променливи и толкова отрицания) с вероятност p. При L на брой клаузи се разглеждат екземпляри на ЗУ, в които $2Np = 3$. Експерименталните резултати при този и при следващите два модела са получени чрез две версии на процедурата на Дейвис-Патнам върху 1000 и 100 000 екземпляри-тестове. Те са генерирани за няколко постоянни стойности на N (до $N = 100$) и при отношение L/N, което се мени в определен диапазон с определена стъпка. Резултатите показват, че когато $2Np = 3 = const$ и N нараства, дължината на клаузите остава приблизително постоянна и фазов преход се появява при $L/N \approx 2.80$. Направени са изводи за максималния брой разклонения, изпълнявани от алгоритмите. Екземплярите в модела "случайна k-УБФ" съдържат отново L клаузи, всяка клауза съдържа k литерала, избрани случайно между N променливи и всеки литерал е или променлива, или нейно отрицание с вероятност 0.5. Експериментите показват, че за $k = 3$ фазов преход се появява при $L/N \approx 4.24$. Този резултат е получен тео-

ретично и в [53] само чрез вероятностно-статистически методи. Третият модел, "случайна смесена УБФ", е обобщение на втория модел. При него дължината на клаузите се генерира случайно в съответствие с дадено разпределение. Самите клаузи се генерират както в модела "случайна k-УБФ". Получените варианти на ЗУ са изследвани както при първите два модела.

Вероятностни техники (при обхождане, избор, оценки) са използвани в [61] за получаването на алгоритъм, който намира вектор, удовлетворяващ произволна удовлетворима БФ на n променливи за време $O(1.3302^n)$.

3.1.4 Комбинаторни и алгебрични методи за изследване на задачите за удовлетворимост

Освен споменатите методи и подходи, при изследване на ЗУ се прилагат комбинаторни и алгебрични методи, които се съчетават и допълват. Много от конюнктивните нормални форми, представящи екземпляри на ЗУ, притежават важни структурни свойства, чрез които сложността на проверката за удовлетворимост може да се редуцира от експоненциална в полиномиална. Съществуващите алгоритми в малка степен отчитат този факт [46]. Комбинаторните и алгебричните методи се прилагат с цел да бъдат открити такива свойства, както и на произтичащите от тях свойства на пространството от възможни решения – n-мерния булев куб. Наличието на структурни свойства предполага класификация на екземплярите на ЗУ, т. е. формулиране на критерии – необходими или достатъчни условия (най-често за неудовлетворимост) – и получаване на цели класове от формули. Важно е да се отбележи, че тези критерии трябва да са полиномиално проверими, за да има смисъл включването им в алгоритми.

Комбинаторните и алгебричните методи се прилагат най-вече за изследване на *минимални неудовлетворими формули* (МНФ), т. е. на множеството от всички неудовлетворими формули, за които премахването на произволна клауза от всяка от тях я превръща в удовлетворима [34, 40, 63]. За произволна МНФ F се дефинира *недостиг* на F, като разлика между броя на клаузите и броя на променливите в F и се означава с $\delta(F)$. Лемата на Тарси [40] утвърждава, че за произволна МНФ F е в сила $\delta(F) \geq 1$. За $k \in N$ се дефинират класовете $MU(k)$ от всички МНФ F, за които $\delta(F) = k$. Пак в [40] е доказано предположението, че за фиксирано $k \in N$ може за полиномиално време да се определи дали формулата $F : \delta(F) \leq k$ е МНФ и е представен съответен алгоритъм.

Друг подход в изследванията е основан на изображения над множеството от всички КНФ в себе си. В [34, 63] се дефинират биекциите *преименуване на променлива* (като пермутация на променливите) и по-общата *преименуване на литерал* (като пермутация на променливите и едновременно с това замяна на променлива с нейно отрицание). Тези преименувания могат да бъдат използвани например в подобрения вариант на алгоритъма на Дейвис-Патнам, известен като DPLL (Davis-Putnam-Loveland-Logemam) алгоритъм. При него пространството за търсене е организирано като двоично дърво с цел да се избегне преминаването през един и

същи възел повече от веднъж. Ако в някой възел правилото за разцепване генерира две формули, които могат да бъдат изобразени една в друга чрез преименуване, тогава едната формула може да се замести с празна клауза, а алгоритъмът да продължи работа с другата формула [46]. В [63] Szeider въвежда понятието *хомоморфизъм* при формули с цел доказване на неудовлетворимост. Хомоморфизмът $\varphi : H \to F$ е изображение на литералите от формулата H в литерали от формулата F, което запазва допълненията и клаузите, т. е. $\varphi(\bar{l}) = \overline{\varphi(l)}$ $\forall l \in H$ и за всяка клауза $C \in H$ е изпълнено $\{\varphi(l)|l \in C\}$ е литерал в F. Хомоморфизмът на формули може да се разглежда и като преименуване на литерали, което допуска изобразяване на различни литерали в един литерал. Той се явява обобщение на т. нар. "симетрии с отрицания", което запазва неудовлетворимостта, т. е. ако съществува хомоморфизъм $\varphi : H \to F$, то от неудовлетворимостта на формулата H следва неудовлетворимост на формулата F. Основен резултат в [63] е хомоморфичната пълнота на класа $MU(1)$ (тоест: (1) $\forall F \in MU(1)$ $\exists H \in MU(1)$ и хомоморфизъм $\varphi : H \to F$ и (2) $MU(1)$ може да бъде разпознат за полиномиално време). В [34] този резултат е обобщен. Доказано е, че за $\forall k, t \geq 1$ и за $\forall F \in MU(t)$ $\exists H \in MU(k)$ и хомоморфизъм $\varphi : H \to F$ такъв, че $\varphi(H) = F$. За фиксирани $k, t \geq 1$ формулата H и хомоморфизмът φ могат да бъдат конструирани за полиномиално време.

В [46] е използван трети подход, основан на структурни свойства на пространството от решения спрямо дадена формула F. Ако F е формула в КНФ, множеството от вектори, за които F се нулира, е означено с $Z(F)$. Ако $P \subseteq Z(F)$, функцията $g : P \to F$, която на всеки вектор от P съпоставя клауза от F, която се нулира в този вектор, се нарича *транспортна функция*. Ако $p \in P$ и тя нулира клаузата C, то $Nbhd(p, C)$ означава множеството от вектори, които също нулират C и са съседни (по Хеминг) на p. Множеството P се нарича *стабилно* спрямо F и транспортната функция g, ако за всяко $p \in P$, $Nbhd(p, g(p)) \subseteq P$. Основен резултат в [46] е твърдението, че формулата F е неудовлетворима тогава и само тогава, когато съществува стабилно множество от вектори спрямо F. Представен е и алгоритъм за конструиране на това множество.

В [51] се използва четвърти подход, основан на принципа за включване и изключване. Показано е, че броя на векторите, удовлетворяващи дадена формула в k-КНФ, се определя еднозначно от множеството от вектори, неудовлетворяващи множества от клаузи с големина до $\lfloor \log k \rfloor + 2$. За целта е необходима предварителна информация за големините на тези множества от вектори.

3.2 Неприводими форми на константата нула

В този раздел са представени изследвания на ограничена ЗУ, в която формулите са в КНФ, съдържат до 3 букви на променливи и дължината на елементарните дизюнкции (по-нататък ще ги наричаме клаузи, както е прието в чужбина) не надминава 3. Множеството от всички тези формули означаваме с C_3. Аналогично на изследванията в края на предишния

раздел, разглеждаме съответната задача за неудовлетворимост. Ако една формула от C_3 е неудовлетворима, то тя е КНФ (или кратко форма) на $\tilde{0}$, множеството от всички тези форми означаваме с C_3^0. Възможните клаузи, които могат да се образуват чрез не повече от 3 променливи, са 26 (6 еднобуквени, 12 двубуквени и 8 трибуквени) и следователно $|C_3| = 2^{26} = 67\,108\,864$. Чрез компютърна програма за генериране на всички форми и пресмятане на стойностите им върху $\{0,1\}^3$ установихме, че $|C_3^0| = 63\,367\,025 \approx 94,424\%|C_3|$. Целта, която си поставяме, е да класифицираме формулите от C_3^0 спрямо еквивалентности, които ще дефинираме в общия случай – над множеството от всички формули в КНФ, съдържащи не повече от n букви на променливи, без ограничения за дължината и броя на клаузите. Това множество означаваме с C_n, а с C_n^0 – неговото подмножество от форми на $\tilde{0}$.

Дефиниция 3.1 Нека $\varphi \in C_n$ и π е произволна пермутация на буквите на променливи от $\{x_1, x_2, \ldots, x_n\}$. Операцията *преименуване* на променливи дефинираме като замяна на всяко срещане на x_i във φ с $x_{\pi(i)}$, $i = 1, 2, \ldots, n$.

Дефиниция 3.2 Нека $\varphi \in C_n$. Операцията *отрицание* на променливата x_i дефинираме като едновременна замяна на x_i с $\bar{x}_i$ и на $\bar{x}_i$ с x_i във всичките им срещания във φ.

Очевидно всяка от двете операции, приложена върху φ, дава формула $\varphi' \in C_n$, която е удовлетворима, ако φ е удовлетворима, или е неудовлетворима, ако φ е неудовлетворима. Тоест преименуването на променливи и замяната на променлива с нейно отрицание могат да бъдат разглеждани като изображения на C_n в себе си, които запазват както удовлетворимостта, така и неудовлетворимостта. По очевидни причини това е валидно и за композициите от такива изображения.

Дефиниция 3.3 Формулата $\varphi' \in C_n$ наричаме *pn-еквивалентна* на формулата $\varphi \in C_n$, ако φ' може да бъде получена от φ чрез преименуване на променливи и/или замяна на променливи с отрицанията им.

Лесно се проверява, че релацията *pn*-еквивалентност е релация на еквивалентност над множеството $C_n \times C_n$. Поради това ще разглеждаме само представители на нейните класове на еквивалентност.

Дефиниция 3.4 Нека $\varphi = D_1 \wedge D_2 \wedge \ldots \wedge D_r \in C_n$. Операцията, която премахва клаузата D_j, $1 \leq j \leq r$, от φ и резултат е формулата $\varphi' = D_1 \wedge \ldots \wedge D_{j-1} \wedge D_{j+1} \wedge \ldots \wedge D_r$, наричаме *отрязване на клаузата* D_j от φ. Формата φ на $\tilde{0}$ наричаме *неприводима при отрязване на клауза форма (НОКФ)*, ако след отрязване на произволно D_j от φ получената формула φ' вече не е форма на $\tilde{0}$.

Дефиниция 3.5 Нека $\varphi = D_1 \wedge D_2 \wedge \ldots \wedge D_r \in C_n^0$ и $D_j = x_{j_1}^{\sigma_1} \vee x_{j_2}^{\sigma_2} \vee \ldots \vee x_{j_s}^{\sigma_s}$, $2 \leq s \leq n$. Операцията, която премахва буквата $x_{j_k}^{\sigma_k}$, $1 \leq k \leq s$, от клаузата D_j и резултат е клаузата $D_j' = x_{j_1}^{\sigma_1} \vee \ldots \vee x_{j_{k-1}}^{\sigma_{k-1}} \vee x_{j_{k+1}}^{\sigma_{k+1}} \vee \ldots \vee x_{j_s}^{\sigma_s}$, наричаме

отрязване на буквата $x_{j_k}^{\sigma_k}$ от клаузата D_j във φ. НОКФ φ наричаме *приводима при отрязване на буква*, ако при отрязване на някоя буква $x_{j_k}^{\sigma_k}$ от някоя нейна клауза се получава формата φ' на $\tilde{0}$, която пак е НОКФ. И обратно: НОКФ φ наричаме *неприводима при отрязване на буква*, ако при отрязване на коя да е буква $x_{j_k}^{\sigma_k}$ от коя да е клауза D_j във φ получаваме формата φ' на $\tilde{0}$, която вече не е НОКФ.

Дефиниция 3.6 Нека $Q = \{\varphi_1, \varphi_2, \ldots, \varphi_s\}$ е множеството от форми на $\tilde{0}$, представители на класовете на pn-еквивалентност в множеството C_n^0. Формата $\varphi_i \in Q$ наричаме *неприводима*, ако тя е НОКФ, неприводима при отрязване на буква.

Дефиниция 3.7 Булевата функция $f \in \mathcal{F}_2^n$ *покрива* вектора $\alpha \in \{0,1\}^n$, ако $f(\alpha) = 0$.

Означаваме $Z_f = \{\alpha | \alpha \in \{0,1\}^n,\ f(\alpha) = 0\}$.

Лема 3.8 *Ако разглеждаме клаузата* $D = x_{i_1}^{\sigma_1} \vee x_{i_2}^{\sigma_2} \vee \ldots \vee x_{i_k}^{\sigma_k}$, $0 \leq k \leq n$, *като булева функция на* n *променливи, тогава* Z_D *е* $(n-k)$*-мерен подкуб на* $\{0,1\}^n$ *и* $|Z_D| = 2^{n-k}$.

Твърдението следва непосредствено от Лема 1.30, Дефиниция 1.11 и поясненията след нея.

Лема 3.9 *За произволни клаузи* D_1 *и* D_2, *ако* x_i *участва в* D_1 *и* $\bar{x}_i$ *участва в* D_2, *тогава* $Z_{D_1} \cap Z_{D_2} = \emptyset$.

Твърдението следва от факта, че всички вектори от Z_{D_1} имат фиксирана стойност 0 в i-та позиция, а тези от Z_{D_2} – фиксирана стойност 1 в същата позиция.

Лема 3.10 *Нека клаузите* $D_1 = x_{i_1}^{a_1} \vee \ldots \vee x_{i_k}^{a_k} \vee x_{j_1}^{b_1} \vee \ldots \vee x_{j_p}^{b_p}$ *и* $D_2 = x_{i_1}^{a_1} \vee \ldots \vee x_{i_k}^{a_k} \vee x_{m_1}^{c_1} \vee \ldots \vee x_{m_q}^{c_q}$ *са булеви функции на* n *общи променливи,* $0 \leq k \leq n$, $0 \leq p, q \leq n-k$ *и* $\{x_{j_1}, \ldots, x_{j_p}\} \cap \{x_{m_1}, \ldots, x_{m_q}\} = \emptyset$. *Тогава:*

1) $|Z_{D_1} \cap Z_{D_2}| = 2^{n-(k+p+q)}$

2) $|Z_{D_1 \wedge D_2}| = 2^{n-(k+p+q)}(2^p + 2^q - 1)$

Доказателство. Полагаме

$$D = D_1 \vee D_2 = \mathrm{x}_{i_1}^{a_1} \vee \ldots \vee \mathrm{x}_{i_k}^{a_k} \vee \mathrm{x}_{j_1}^{b_1} \vee \ldots \vee \mathrm{x}_{j_p}^{b_p} \vee \mathrm{x}_{m_1}^{c_1} \vee \ldots \vee \mathrm{x}_{m_q}^{c_q}.$$

1) Множеството $Z_{D_1} \cap Z_{D_2}$ съдържа само тези вектори $\alpha \in \{0,1\}^n$, за които $D_1(\alpha) = 0$ и $D_2(\alpha) = 0$. Следователно $D(\alpha) = 0$ и $|Z_{D_1} \cap Z_{D_2}| = |Z_D| = |Z_{D_1 \vee D_2}| = 2^{n-(k+p+q)}$, съгласно Лема 3.8.

2) Множеството $Z_{D_1 \wedge D_2}$ съдържа само тези вектори $\alpha \in \{0,1\}^n$, за които $D_1(\alpha) = 0$ или $D_2(\alpha) = 0$. Следователно $Z_{D_1 \wedge D_2} = Z_{D_1} \cup Z_{D_2}$ и

$$|Z_{D_1 \wedge D_2}| = |Z_{D_1} \cup Z_{D_2}| = |Z_{D_1}| + |Z_{D_2}| - |Z_{D_1} \cap Z_{D_2}| =$$
$$= 2^{n-(k+p)} + 2^{n-(k+q)} - 2^{n-(k+p+q)} = 2^{n-(k+p+q)}(2^p + 2^q - 1). \diamond$$

Лема 3.11 *Нека* C *е множеството от всички 12 клаузи с дължина 2, образувани от буквите на променливи* $\{x_i, x_j, x_k\}$ *или техни отрицания. Нека*

векторът $\alpha \in \{0,1\}^n$ *се покрива едновременно от три клаузи от* C. *Тогава останалите непокрити от тези три клаузи вектори от* $\{0,1\}^n$ *могат да бъдат покрити чрез поне още три клаузи от* C.

Доказателство. Нека $\alpha = (\ldots, a_i, \ldots, a_j, \ldots, a_k, \ldots)$. Той се покрива едновременно от клаузите $D_1 = x_i^{\bar{a}_i} \vee x_j^{\bar{a}_j}$, $D_2 = x_i^{\bar{a}_i} \vee x_k^{\bar{a}_k}$ и $D_3 = x_j^{\bar{a}_j} \vee x_k^{\bar{a}_k}$. Прилагаме принципа за включване и изключване и Лема 3.10 и получаваме $|Z_{D_1 \wedge D_2 \wedge D_3}| = |Z_{D_1} \cup Z_{D_2} \cup Z_{D_3}| = \ldots = 2^{n-1}$. Тоест D_1, D_2 и D_3 покриват общо половината от векторите в $\{0,1\}^n$ и това са точно векторите, съседни на α по i-та, или по j-та, или по k-та позиция. В останалата половина вектори в $\{0,1\}^n$ е векторът $\bar{\alpha}$, противоположен на α. Без ограничение на общността да приемем, че той се покрива от клаузата $D_4 = x_i^{a_i} \vee x_j^{a_j}$. Тогава D_4 покрива и векторите, съседни на $\bar{\alpha}$ по k-та позиция, тоест от вида $(\ldots, \bar{a}_i, \ldots, \bar{a}_j, \ldots, a_k, \ldots)$. Останалите две множества от вектори – съседните на $\bar{\alpha}$ по j-та позиция, или от вида $(\ldots, \bar{a}_i, \ldots, a_j, \ldots, \bar{a}_k, \ldots)$ и съседните на $\bar{\alpha}$ по i-та позиция, или от вида $(\ldots, a_i, \ldots, \bar{a}_j, \ldots, \bar{a}_k, \ldots)$ – се различават помежду си в i-та и в j-та позиции и следователно те не могат да бъдат покрити от една и съща клауза от C. Тъй като векторите от втората половина на $\{0,1\}^n$ са противоположните на тези от първата половина, можем да изберем $D_5 = x_i^{a_i} \vee x_k^{a_k}$ и $D_6 = x_j^{a_j} \vee x_k^{a_k}$ (клаузите D_4, D_5 и D_6 можем да разглеждаме като получени съответно от D_1, D_2 и D_3 чрез прилагане на операцията отрицание на променлива). Съгласно Леми 3.8, 3.9 и 3.10 имаме $Z_{D_1 \wedge \ldots \wedge D_6} = \{0,1\}^n$. $\diamond$

В допълнение към твърдението на Лема 3.11 отбелязваме, че клаузите D_1, D_2 и D_3 покриват едновременно не само вектора α, но и целия $(n-3)$-мерен подкуб, образуван от векторите с фиксирани стойности a_i, a_j и a_k съответно в позиции i, j и k. Аналогично твърдение е в сила и за клаузите D_4, D_5 и D_6 и вектора $\bar{\alpha}$. Следователно една осма от векторите в $\{0,1\}^n$ се покриват и от трите клаузи D_1, D_2 и D_3, друга една осма се покриват едновременно от D_4, D_5 и D_6, а всички останали вектори се покриват от по точно една клауза.

Теорема 3.12 *Неприводимите 3-КНФ* $D_1 \wedge D_2 \wedge \ldots \wedge D_r$ *на* $\tilde{0}$ *с до 3 променливи са:*

1) за $r = 1$: *празната клауза или самата* $\tilde{0}$;

2) за $r = 2$: $x_i \bar{x}_i$;

3) за $r = 3$: $x_i x_j (\bar{x}_i \vee \bar{x}_j)$;

4) за $r = 4$:
- $x_i x_j x_k (\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$;
- $x_i x_j (\bar{x}_i \vee x_k)(\bar{x}_j \vee \bar{x}_k)$;
- $(x_i \vee x_j)(x_i \vee \bar{x}_j)(\bar{x}_i \vee x_j)(\bar{x}_i \vee \bar{x}_j)$;

5) за $r = 5$:
- $x_i (x_j \vee x_k)(x_j \vee \bar{x}_k)(\bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$;
- $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee x_k)(\bar{x}_j \vee \bar{x}_k)$;
- $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee \bar{x}_j)(\bar{x}_j \vee \bar{x}_k)$;

6) за $r = 6$:
- $(x_i \vee x_j)(x_i \vee x_k)(x_j \vee x_k)(\bar{x}_i \vee \bar{x}_j)(\bar{x}_i \vee \bar{x}_k)(\bar{x}_j \vee \bar{x}_k)$;

- $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee x_k)(x_i \vee \bar{x}_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$;
- $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee \bar{x}_k)(x_i \vee \bar{x}_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee x_k)$;

7) за $r = 7$:

- $(x_i \vee x_j)(x_i \vee x_k)(x_j \vee x_k)(x_i \vee \bar{x}_j \vee \bar{x}_k)(\bar{x}_i \vee x_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$;

8) за $r = 8$:

- $(x_i \vee x_j \vee x_k)(x_i \vee x_j \vee \bar{x}_k)(x_i \vee \bar{x}_j \vee x_k)(x_i \vee \bar{x}_j \vee \bar{x}_k) \wedge$
  $\wedge(\bar{x}_i \vee x_j \vee x_k)(\bar{x}_i \vee x_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$.

Доказателство.

1) Празната клауза или $\tilde{0}$ сама по себе си е единствената неприводима форма на $\tilde{0}$ с една клауза.

2) Очевидно, $x_i\bar{x}_i$ е неприводима – нито x_i, нито $\bar{x}_i$ е форма на $\tilde{0}$. Тази форма на $\tilde{0}$ е единствената с две клаузи. Действително, ако във форма на $\tilde{0}$ с две клаузи едната е с дължина 1 – например x_i, втората не може да бъде празната клауза (това противоречи на неприводимостта) и остава тя също да бъде с дължина 1, съгласно Лема 3.8. Според Леми 3.9 и 3.10 втората клауза може да бъде единствено $\bar{x}_i$. Лема 3.8 обуславя невъзможността клаузите да имат дължини по-големи или равни на 2.

3) Неприводима форма на $\tilde{0}$, състояща се от три еднобуквени клаузи не съществува – очевидно $\varphi = x_i^{a_1}x_j^{a_2}x_k^{a_3} \neq \tilde{0}$, а $\varphi = x_i\bar{x}_ix_j^a$ не е неприводима.

Ако приемем $D_1 = x_i$ и $D_2 = x_j$, тогава $|Z_{D_1 \wedge D_2}| = 3.2^{n-2}$ и останалите 2^{n-2} вектора трябва да бъдат покрити от третата клауза. Според Леми 3.8 и 3.9, единствено клаузата $D_3 = \bar{x}_i \vee \bar{x}_j$ покрива точно тези вектори. Получаваме неприводимата форма $x_ix_j(\bar{x}_i \vee \bar{x}_j)$ на $\tilde{0}$, която се оказва единствена в този случай – съгласно Леми 3.8, 3.9 и 3.10, за останалите форми на $\tilde{0}$ с три клаузи е в сила:

- те не могат да съдържат клауза с дължина 3, нито три клаузи с дължина 2 и следователно в тях могат да участват точно две букви на променливи (с или без отрицания);

- ако те съдържат точно една клауза с дължина 1, те не са неприводими при отрязване на буква.

4) Доказателството, че не съществува 3-КНФ на $\tilde{0}$ от четири клаузи с дължина 1 е аналогично на това в случая 3).

Разглеждаме подслучая, в който три клаузи са с дължина 1, а четвъртата е по-дълга. Такава форма на $\tilde{0}$, в която четвъртата клауза е с дължина 2, има вида $x_ix_jx_k(\bar{x}_i \vee \bar{x}_j)$ и тя не е неприводима при отрязване на клаузата x_k. Остава четвъртата клауза да бъде с дължина 3 – получаваме единствената неприводима форма на $\tilde{0}$ в този подслучай $x_ix_jx_k(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$.

Да разгледаме подслучая, в който точно две клаузи са с дължина 1 и нека това са $D_1 = x_i$ и $D_2 = x_j$. Остават 2^{n-2} вектора, които трябва да бъдат покрити от D_3 и D_4. Съгласно Лема 3.8, това може да стане по три начина и получаваме следните НОКФ:

- $x_ix_j(\bar{x}_i \vee x_k)(\bar{x}_j \vee \bar{x}_k)$;
- $x_ix_j(\bar{x}_i \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$;
- $x_ix_j(\bar{x}_i \vee \bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$.

Втората форма се получава от третата чрез отрязване на буква, а първата - чрез отрязване на буква от втората и следователно не са неп-

риводими. Неприводима е само първата от тях.

В следващия подслучай имаме само една клауза с дължина 1 и нека $D_1 = x_i$. Тя покрива половината от векторите, а останалата половина трябва да се покрива от клаузите D_2, D_3 и D_4, които да са с дължина 2 или 3. Възможните НОКФ са:

- $x_i(x_j \vee x_k)(x_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j)$;
- $x_i(x_j \vee x_k)(\bar{x}_i \vee \bar{x}_j)(\bar{x}_i \vee \bar{x}_k)$;
- $x_i(x_j \vee x_k)(\bar{x}_i \vee \bar{x}_j)(\bar{x}_i \vee x_j \vee \bar{x}_k)$;
- $x_i(\bar{x}_i \vee x_j)(\bar{x}_i \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$;
- $x_i(\bar{x}_i \vee x_j)(\bar{x}_i \vee \bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$.

Очевидно, нито една от тях не е неприводима при отрязване на буква.

В следващия подслучай всички клаузи са с дължина 2. Всяка от тях покрива точно 2^{n-2} вектора и следователно $Z_{D_i} \cap Z_{D_j} = \emptyset$, $1 \leq i < j \leq 4$. Възможните две форми на $\tilde{0}$, конструирани от 2 и от 3 променливи, са:

- $(x_i \vee x_j)(x_i \vee \bar{x}_j)(\bar{x}_i \vee x_j)(\bar{x}_i \vee \bar{x}_j)$;
- $(x_i \vee x_j)(x_i \vee \bar{x}_j)(\bar{x}_i \vee x_k)(\bar{x}_i \vee \bar{x}_k)$.

Първата форма е неприводима, а втората не е, понеже можем (дори по няколко начина) да изрежем буква от нейна клауза и да получим НОКФ, която е pn-еквивалентна на първата или втората форма от предишния подслучай.

3-КНФ с четири клаузи, всяка с дължина по-голяма или равна на 2 и поне една от тях с дължина 3, не могат да покрият всички вектори и следователно не са форми на $\tilde{0}$.

5) Не съществуват НОКФ с пет клаузи, в които да има две или три клаузи с дължина 1 – в противен случай поне една от клаузите е излишна и формата не е минимална.

Първата от останалите възможности за получаване на неприводими форми на $\tilde{0}$ с пет клаузи е да имаме една клауза с дължина 1 – например $D_1 = x_i$. Тя покрива половината от векторите, а останалата половина може да бъде покрита от клаузите: $D_2 = \bar{x}_i \vee x_j \vee x_k$, $D_3 = \bar{x}_i \vee x_j \vee \bar{x}_k$, $D_4 = \bar{x}_i \vee \bar{x}_j \vee x_k$ и $D_5 = \bar{x}_i \vee \bar{x}_j \vee \bar{x}_k$. Тогава $D_1 \wedge \ldots \wedge D_5$ е форма на $\tilde{0}$ (съгласно Леми 3.8, 3.9 и 3.10), но не е неприводима. Изрязваме единствената обща променлива $\bar{x}_i$ от D_2, D_3 и D_4. D_5 трябва да остане същата – в противен случай D_1 става излишна и формата не е неприводима при отрязване на клауза. Така получаваме първата неприводима форма $x_i(x_j \vee x_k)(x_j \vee \bar{x}_k)(\bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$.

Втората възможност е да конструираме форма, в която всяка от петте клаузи е с дължина 2. Те ще покриват общо 5.2^{n-2} вектора, т. е. 2^{n-2} вектора в $\{0,1\}^n$ ще се припокриват от по две клаузи (и няма вектор, който да се покрива едновременно от три клаузи, съгласно Лема 3.11). Нека $D_1 = x_i \vee x_j$, $D_2 = \bar{x}_i \vee x_j$ и $D_3 = \bar{x}_j \vee \bar{x}_k$. Имаме $Z_{D_1} \cap Z_{D_2} = Z_{D_1} \cap Z_{D_3} = Z_{D_2} \cap Z_{D_3} = \emptyset$ поради Лема 3.9. За да покрием останалите 2^{n-2} вектора, можем да изберем $D_4 = x_i \vee x_k$ и $D_5 = \bar{x}_i \vee x_k$ – тогава $Z_{D_1} \cap Z_{D_4} \neq \emptyset$, а $Z_{D_2} \cap Z_{D_4} = Z_{D_3} \cap Z_{D_4} = Z_{D_4} \cap Z_{D_5} = \emptyset$ и $Z_{D_2} \cap Z_{D_5} \neq \emptyset$, а $Z_{D_1} \cap Z_{D_5} = Z_{D_3} \cap Z_{D_5} = \emptyset$. Получаваме неприводимата форма $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee x_k)(\bar{x}_j \vee \bar{x}_k)$. Възможен е и втори начин за избор на D_5 – $D_5 = \bar{x}_i \vee \bar{x}_j$. Аналогично,

$Z_{D_3} \cap Z_{D_5} \neq \emptyset$, а $Z_{D_1} \cap Z_{D_5} = Z_{D_2} \cap Z_{D_5} = Z_{D_4} \cap Z_{D_5} = \emptyset$ и получаваме третата неприводима форма на $\tilde{0}$ – $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee \bar{x}_j)(\bar{x}_j \vee \bar{x}_k)$.

Накрая отбелязваме, че всяка форма на $\tilde{0}$ от пет клаузи с дължина поне 2 и поне една клауза с дължина 3, не е неприводима при отрязване на буква.

6) Отбелязваме, че в този случай всички клаузи трябва да са с дължина не по-малка от 2 – в противен случай формата няма да бъде неприводима при отрязване на клауза.

Първо ще разгледаме случая, в който всички клаузи са с дължина 2. Те покриват общо $6.2^{n-2} = 2^n + 2^{n-1}$ вектора в $\{0,1\}^n$ и очевидно някои вектори се припокриват от повече от една клаузи. Ако допуснем, че половината вектори се припокриват от точно по две клаузи, тогава поне една клауза става излишна и формата не е неприводима. Остава възможността три клаузи да покриват един и същи вектор. Съгласно Лема 3.11 и поясненията след нея, това автоматично означава една четвърт от векторите да се припокриват от точно по три клаузи. Получаваме формата $(x_i \vee x_j)(x_i \vee x_k)(x_j \vee x_k)(\bar{x}_i \vee \bar{x}_j)(\bar{x}_i \vee \bar{x}_k)(\bar{x}_j \vee \bar{x}_k)$, която е единствената 3-КНФ на $\tilde{0}$ от шест клаузи с дължина две.

Всяка форма на $\tilde{0}$ от пет клаузи с дължина 2 и една клауза с дължина 3 можем да разглеждаме като получена от неприводимата форма от предишния подслучай, като в една от нейните клаузи сме добавили неучастващата променлива или нейно отрицание. Следователно формите от този тип не са неприводими (при отрязване на буква).

Следващият възможен подслучай е форма от 4 клаузи с дължина 2 и две клаузи с дължина 3. Нека първо изберем $D_1 = x_i \vee x_j$, $D_2 = x_i \vee x_k$ и $D_3 = x_j \vee x_k$. Те покриват половината вектори и тогава D_4 може да бъде $\bar{x}_i \vee \bar{x}_j$, или $\bar{x}_i \vee \bar{x}_k$, или $\bar{x}_j \vee \bar{x}_k$, покривайки една четвърт от останалите вектори. Тогава D_5 и D_6 могат да бъдат избрани по единствен начин, така че да покриват останалите 2^{n-2} вектори и $Z_{D_5} \cap Z_{D_6} = \emptyset$. Получаваме три форми на $\tilde{0}$, които не са неприводими – отрязване на буква от D_5 или от D_6 дава НОКФ от предишния подслучай (с пет клаузи с дължина 2 и една клауза с дължина 3). Да изберем първите четири клаузи по друг начин: $D_1 = x_i \vee x_j$, $D_2 = x_i \vee x_k$ и $D_3 = \bar{x}_i \vee x_j$. Тогава $Z_{D_1} \cap Z_{D_2} \neq \emptyset$ и $Z_{D_1} \cap Z_{D_3} = Z_{D_2} \cap Z_{D_3} = \emptyset$, според Лема 3.9. D_4 трябва да бъде $D'_4 = \bar{x}_i \vee x_k$ или $D''_4 = \bar{x}_i \vee \bar{x}_k$, така че $Z_{D_1} \cap Z_{D_4} = Z_{D_2} \cap Z_{D_4} = \emptyset$ и $Z_{D_3} \cap Z_{D_4} \neq \emptyset$. Следователно $Z_{D_1 \wedge D_2} \cap Z_{D_3 \wedge D_4} = \emptyset$, $|Z_{D_1 \wedge D_2}| = |Z_{D_3 \wedge D_4}| = 2^{n-3}(2+2-1) = 3.2^{n-3}$ и $|Z_{D_1 \wedge D_2 \wedge D_3 \wedge D_4}| = |Z_{D_1 \wedge D_2}| + |Z_{D_3 \wedge D_4}| = 6.2^{n-3}$ поради Лема 3.10. Останалите 2^{n-2} вектора трябва да бъдат покрити чрез клаузите D_5 и D_6 и да няма припокривания както помежду им, така и с останалите клаузи. Тогава D_5 трябва да бъде $D_5 = x_i \vee \bar{x}_j \vee \bar{x}_k$. Когато $D_4 = D'_4$ имаме $D_6 = \bar{x}_i \vee \bar{x}_j \vee \bar{x}_k$, а когато $D_4 = D''_4$, имаме $D_6 = \bar{x}_i \vee \bar{x}_j \vee x_k$. Така получаваме две неприводими форми на $\tilde{0}$, съставени от четири клаузи с дължина 2 и две клаузи с дължина 3, а именно:

- $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee x_k)(x_i \vee \bar{x}_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$;
- $(x_i \vee x_j)(x_i \vee x_k)(\bar{x}_i \vee x_j)(\bar{x}_i \vee \bar{x}_k)(x_i \vee \bar{x}_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee x_k)$.

Лесно може да се провери, че формите от шест клаузи, които съдържат повече от 2 клаузи с дължина 3, не са неприводими (при отрязване на буква).

7) Най-простият начин за получаване на 3-КНФ на $\tilde{0}$ от седем клаузи $D_1, D_2, \ldots, D_7$ е да ги изберем така, че $Z_{D_i} \cap Z_{D_j} = \emptyset$, за $1 \leq i < j \leq 7$. Тогава D_1 трябва да бъде с дължина 2 и да покрива 2^{n-2} вектора, а останалите клаузи да бъдат с дължина 3 и да покрият останалите 6.2^{n-3} вектора. Да изберем $D_1 = x_i \vee x_j$, $D_2 = x_i \vee \bar{x}_j \vee x_k$, $D_3 = x_i \vee \bar{x}_j \vee \bar{x}_k$, $D_4 = \bar{x}_i \vee x_j \vee x_k$, $D_5 = \bar{x}_i \vee x_j \vee \bar{x}_k$, $D_6 = \bar{x}_i \vee \bar{x}_j \vee x_k$ и $D_7 = \bar{x}_i \vee \bar{x}_j \vee \bar{x}_k$. Възможни са различни отрязвания на букви в $D_2, D_3, \ldots, D_7$, които дават отново форми на $\tilde{0}$. Оказва се, че в тях също са възможни отрязвания на букви. Тоест, форми от седем клаузи, сред които една или две клаузи са с дължина 2, не са неприводими при отрязване на буква. Подслучаят от три клаузи с дължина 2 е подобен на случая 6) и произтича от Лема 3.11. Трите клаузи с дължина 2 покриват половината вектори, а останалата половина се покрива от четирите клаузи с дължина 3. Получаваме формата

$$(x_i \vee x_j)(x_i \vee x_k)(x_j \vee x_k)(x_i \vee \bar{x}_j \vee \bar{x}_k)(\bar{x}_i \vee x_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k),$$

в която повече отрязвания на букви са невъзможни – в противен случай ще получим форма, която не е НОКФ. Това е единствената неприводима форма на $\tilde{0}$ със седем клаузи.

8) Формите на $\tilde{0}$ с осем клаузи не могат да съдържат клауза с дължина 2 – в противен случай формата съдържа излишна клауза и не е НОКФ. Всяка клауза трябва да е с дължина 3 (т. е. да покрива 2^{n-3} вектора) и да няма припокриване на нито един вектор (т. е. $Z_{D_i} \cap Z_{D_j} = \emptyset$, за $1 \leq i < j \leq 8$). Осемте клаузи могат да бъдат избрани по единствен начин и единствената неприводима форма на $\tilde{0}$ в този случай е:

$$(x_i \vee x_j \vee x_k)(x_i \vee x_j \vee \bar{x}_k)(x_i \vee \bar{x}_j \vee x_k)(x_i \vee \bar{x}_j \vee \bar{x}_k) \wedge$$
$$\wedge(\bar{x}_i \vee x_j \vee x_k)(\bar{x}_i \vee x_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k).$$

Отбелязваме, че всяка форма на $\tilde{0}$ на не повече от три променливи, съдържаща повече от осем клаузи, не е НОКФ и с това теоремата е доказана. $\diamond$

В заключение ще кажем, че дадената в Теорема 3.12 класификация е подпомогната и проверена чрез разработени специално за целта компютърни програми. Споменатите вече 63 367 025 форми на $\tilde{0}$ (елементи на множеството C_3^0), след отрязване на клаузи се редуцират до 869 минимални неудовлетворими форми (без празната клауза). След премахване на pn-еквивалентните и на приводимите при отрязване на буква се получават неприводимите форми от теоремата.

3.3 Необходими или достатъчни условия при задачата 3-УБФ

В този раздел се изследва оригиналната задача 3-УБФ, както и съответната ѝ задача за неудовлетворимост. Разглеждаме всички клаузи

в дадена 3-КНФ като булеви функции на толкова променливи, колкото участват в самата 3-КНФ, именно на n променливи в общия случай.

Да означим с S множеството от всички възможни трибуквени клаузи над множеството от букви на променливи $X = \{x_1, x_2, \ldots, x_n\}$. Ще разбием S на четири непресичащи се помежду си подмножества в зависимост от броя на отрицанията във всяка клауза. Означаваме:

$S_0 = \{x_i \vee x_j \vee x_k |\ x_i, x_j, x_k \in X\}$. Следователно $|S_0| = \binom{3}{0}\binom{n}{3} = \binom{n}{3}$;

$S_1 = \{\bar{x}_i \vee x_j \vee x_k |\ x_i, x_j, x_k \in X\}$. Следователно $|S_1| = \binom{3}{1}\binom{n}{3} = 3\binom{n}{3}$;

$S_2 = \{\bar{x}_i \vee \bar{x}_j \vee x_k |\ x_i, x_j, x_k \in X\}$. Следователно $|S_2| = \binom{3}{2}\binom{n}{3} = 3\binom{n}{3}$;

$S_3 = \{\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k |\ x_i, x_j, x_k \in X\}$. Следователно $|S_3| = \binom{3}{3}\binom{n}{3} = \binom{n}{3}$.

Тогава $S = S_0 \cup S_1 \cup S_2 \cup S_3$ и съдържа общо $8\binom{n}{3}$ клаузи. Следователно $8\binom{n}{3}$, умножено по някаква константа $c > 0$ (в зависимост от избрания начин на представяне на формите), е максималната дължина на входа при задачата 3-УБФ. Полагаме $m = \binom{n}{3}$ и ще ползваме m само за това по-кратко означение в този раздел.

В съответствие с Теорема 3.12 дефинираме:

Дефиниция 3.13 Всяка 3-КНФ от вида $(x_i \vee x_j \vee x_k)(x_i \vee x_j \vee \bar{x}_k)(x_i \vee \bar{x}_j \vee x_k)$ $(x_i \vee \bar{x}_j \vee \vee \bar{x}_k)(\bar{x}_i \vee x_j \vee x_k)(\bar{x}_i \vee x_j \vee \bar{x}_k)(\bar{x}_i \vee \bar{x}_j \vee x_k)(\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$ ще наричаме *минимална основна форма* (МОФ) на $\tilde{0}$ и ще я бележим с $M_{i,j,k}$.

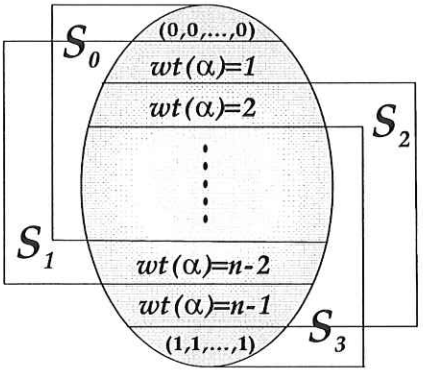
В следващата таблица са дадени елементите на множеството S във вида $D = x_i^{\sigma_1} \vee x_j^{\sigma_2} \vee x_k^{\sigma_3}$ за различните стойности на (i, j, k) и в комбинация с възможните стойности на $(\sigma_1, \sigma_2, \sigma_3)$. Представено е разбиването на S на подмножества $S_0, \ldots, S_3$, вида на възможните МОФ и разбиването на S по МОФ, разглеждани като множества от по осем клаузи.

Стойности на $(\sigma_1, \sigma_2, \sigma_3)$	Възможни тройки променливи и съотв. им клаузи					Подмножества на S
	(1, 2, 3)	$\cdots$	(i, j, k)	$\cdots$	(n-2, n-1, n)	
(0,0,0)	$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$		$\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k$		$\bar{x}_{n-2} \vee \bar{x}_{n-1} \vee \bar{x}_n$	S_3
(0,0,1)	$\bar{x}_1 \vee \bar{x}_2 \vee x_3$		$\bar{x}_i \vee \bar{x}_j \vee x_k$		$\bar{x}_{n-2} \vee \bar{x}_{n-1} \vee x_n$	
(0,1,0)	$\bar{x}_1 \vee x_2 \vee \bar{x}_3$	$\cdots$	$\bar{x}_i \vee x_j \vee \bar{x}_k$	$\cdots$	$\bar{x}_{n-2} \vee x_{n-1} \vee \bar{x}_n$	S_2
(1,0,0)	$x_1 \vee \bar{x}_2 \vee \bar{x}_3$		$x_i \vee \bar{x}_j \vee \bar{x}_k$		$x_{n-2} \vee \bar{x}_{n-1} \vee \bar{x}_n$	
(0,1,1)	$\bar{x}_1 \vee x_2 \vee x_3$		$\bar{x}_i \vee x_j \vee x_k$		$\bar{x}_{n-2} \vee x_{n-1} \vee x_n$	
(1,0,1)	$x_1 \vee \bar{x}_2 \vee x_3$	$\cdots$	$x_i \vee \bar{x}_j \vee x_k$	$\cdots$	$x_{n-2} \vee \bar{x}_{n-1} \vee x_n$	S_1
(1,1,0)	$x_1 \vee x_2 \vee \bar{x}_3$		$x_i \vee x_j \vee \bar{x}_k$		$x_{n-2} \vee x_{n-1} \vee \bar{x}_n$	
(1,1,1)	$x_1 \vee x_2 \vee x_3$		$x_i \vee x_j \vee x_k$		$x_{n-2} \vee x_{n-1} \vee x_n$	S_0
МОФ	$M_{1,2,3}$	$\cdots$	$M_{i,j,k}$	$\cdots$	$M_{n-2,n-1,n}$	Подмн-ва на S

Таблица 3.1: Множеството S и неговите разбивания.

Да определим по колко вектора $\alpha \in \{0,1\}^n$ покриват общо всички клаузи от множествата $S_0, \ldots, S_3$. Означаваме с $cov(S_i)$ броя на векторите от $\{0,1\}^n$, които се покриват от клаузите на множеството S_i, за $i = 0, \ldots, 4$. Клаузите от множеството S_0 покриват всички вектори α, такива че $0 \leq$

$wt(\alpha) \leq n-3$, т. е. всички вектори, които съдържат поне три нулеви компоненти. Техният брой получаваме като от 2^n извадим броя на векторите с нула, с една и с две нулеви компоненти и той е $cov(S_0) = 2^n - 1 - \frac{n(n+1)}{2}$. Аналогично, клаузите от S_3 покриват всички вектори α, такива че $3 \leq wt(\alpha) \leq n$, т. е. векторите, съдържащи поне три единични компоненти. Техният брой отново е $2^n - 1 - \frac{n(n+1)}{2}$. Клаузите от множеството S_1 покриват всички вектори α, такива че $1 \leq wt(\alpha) \leq n-2$, т. е. всички вектори с поне две нулеви и с поне една единична компоненти. Получаваме, че броят им е $cov(S_1) = 2^n - 2 - n$. И аналогично, клаузите от S_2 покриват всички вектори α, такива че $2 \leq wt(\alpha) \leq n-1$, т. е. векторите с поне една нулева и с поне две единични компоненти. Броят им отново е $2^n - 2 - n$. Всичките тези покрития са илюстрирани на Фиг. 3.1, в която $\{0,1\}^n$ е представен чрез разбиване по слоеве.



Фиг. 3.1: Покриване на слоевете в $\{0,1\}^n$ чрез множествата $S_0, \ldots, S_3$.

В Таблица 3.2 са дадени стойностите на $cov(S_i)$, $i = 0, \ldots, 3$, колко вектора покриват сумарно и мощността на $\{0,1\}^n$ за $3 \leq n \leq 7$.

Формула за:	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$
$cov(S_0) = cov(S_3) = 2^n - 1 - \frac{n(n+1)}{2}$	1	5	16	42	99
$cov(S_1) = cov(S_2) = 2^n - 2 - n$	3	10	25	56	119
$\sum_{i=0}^{3} cov(S_i) = 2^{n+2} - 6 - n(n+3)$	8	30	82	196	436
$\lvert\{0,1\}^n\rvert = 2^n$	8	16	32	64	128

Таблица 3.2: Стойностите на $cov(S_i)$, $i = 0, \ldots, 3$ и на $|\{0,1\}^n|$, за $n = 3, \ldots, 7$.

Теорема 3.14 (Необходими условия за неудовлетворимост) *Нека φ е 3-КНФ на $\tilde{0}$ на n ($n \geq 3$) променливи. Тогава за нея са в сила условията:*

1) φ съдържа поне осем клаузи;

2) φ съдържа поне една клауза от множеството S_0 и поне една клауза от множеството S_3

3) в зависимост от броя на променливите n, във φ участват:

– при $n = 3$ – всички клаузи от S;

– *при* $n = 4$ – *клаузи от поне три от множества* $S_0, \ldots, S_3$, *съобразно 2);*
– *при* $n \geq 5$ – *клаузи от поне две от множества* $S_0, \ldots, S_3$, *съобразно 2).*

Твърдението от Теорема 3.14 е пряко следствие от Теорема 3.12, Таблици 3.1 и 3.2, и Фиг. 3.1.

И така, 3-КНФ с по-малко от осем клаузи не могат да бъдат форми на $\tilde{0}$. Броят на всевъзможните такива форми е $\sum_{i=0}^{7} \binom{8m}{i}$.

Да пресметнем колко форми не отговарят на второто условие (т. е. не покриват нулевия или единичния вектор на $\{0,1\}^n$) и следователно не са форми на $\tilde{0}$. Тези, които не съдържат нито една клауза от S_0 (т. е. съдържат клаузи само от $S_1 \cup S_2 \cup S_3$), са 2^{7m}. Толкова са и формите, които не съдържат клаузи от S_3. Формите, които съдържат клаузи само от $S_1 \cup S_2$, са 2^{6m}. Чрез принципа за включване и изключване получаваме, че общо $2^{7m} + 2^{7m} - 2^{6m} = 2^{7m+1} - 2^{6m}$ форми не съдържат клауза от S_0, или не съдържат клауза от S_3. Сред тях има форми с по-малко от осем клаузи. Аналогично, те са $\sum_{i=0}^{7} \binom{7m}{i} + \sum_{i=0}^{7} \binom{7m}{i} - \sum_{i=0}^{7} \binom{6m}{i}$ на брой. Следователно броят на удовлетворимите 3-КНФ е не по-малко от

$$2^{7m+1} - 2^{6m} + \sum_{i=0}^{7} \binom{8m}{i} - \sum_{i=0}^{7} \left(2\binom{7m}{i} - \binom{6m}{i}\right) = \tag{3.1}$$
$$= 2^{7m+1} - 2^{6m} + \sum_{i=0}^{7} \left(\binom{8m}{i} - 2\binom{7m}{i} + \binom{6m}{i}\right).$$

Теорема 3.15 (Достатъчни условия за неудовлетворимост) *Нека* φ *е 3-КНФ на* n $(n \geq 3)$ *променливи, за която е изпълнено някое от условията:*

1) множеството от клаузите ѝ съдържа като подмножество клаузите на произволна МОФ;

2) съдържа повече от $7m$ *клаузи.*

Тогава φ *е форма на* $\tilde{0}$.

Доказателство.

1) Твърдението е пряко следствие от Теорема 3.12.

2) Очевидно 3-КНФ, съдържаща всевъзможните $8m$ на брой клаузи, е форма на $\tilde{0}$. Да пресметнем колко е максималния брой клаузи, които могат да бъдат премахнати от нея така, че получената форма да остане пак форма на $\tilde{0}$. За целта ще определим колко клаузи от S покриват един и същи вектор $\alpha \in \{0,1\}^n$. Нека α съдържа единици в позициите $\{i_1, i_2, \ldots, i_k\}$, $0 \leq k \leq n$. Образуваме множеството от променливи $Y = \{\bar{x}_{i_1}, \bar{x}_{i_2}, \ldots, \bar{x}_{i_k}, x_{j_1}, x_{j_2}, \ldots, x_{j_{n-k}}\}$ и разглеждаме всевъзможните клаузи $D = y_p \vee y_q \vee y_r$, в които $y_p, y_q, y_r \in Y$. Очевидно $D(\alpha) = 0$ и броят на тези клаузи е равен на броя на начините за избор на трите променливи y_p, y_q, y_r от Y, т. е. равен на m. Тогава можем да премахнем до $m-1$ от тези клаузи и векторът α пак ще остане покрит. Следователно при отрязване на не повече от $m-1$ произволни клаузи от 3-КНФ, съдържаща всички клаузи от S, тя остава отново форма на $\tilde{0}$.

Да оценим броя на 3-КНФ на $\tilde{0}$ отдолу, като използваме първото условие от Теорема 3.15. Нека A е множеството от всички 3-КНФ, такива че

всяка от тях съдържа в себе си (като подмножество) клаузите на МОФ $M_{1,2,3}$, а B – множеството от тези 3-КНФ, всяка от които съдържа в себе си (като подмножество) клаузите на МОФ $M_{1,2,4}$. Тогава $|A| = |B| = 2^{8m-8}$. Обединението $A \cup B$ съдържа всевъзможните 3-КНФ, които съдържат в себе си (като подмножество) клаузите на произволен брой минимални основни форми. Следователно броят на тези 3-КНФ е:

$$|A \cup B| = |A| + |B| - |A \cap B| = 2^{8m-8} + 2^{8m-8} - 2^{8m-16} = 2^{8m}(2^{-7} - 2^{-16}) \quad (3.2)$$

Това представлява $\approx \frac{1}{128}$ част от броя на всички възможни 3-КНФ и това отношение не зависи от n. Тази оценка е груба, понеже отчита само формите, в които участват задължително клаузите на $M_{1,2,3}$ или на $M_{1,2,4}$. Може да бъде направена и по-прецизна оценка (като изключим $M_{1,2,3}$ и $M_{1,2,4}$ от S, изберем следващите две МОФ и преброим формите, в които участва поне една от тях, после изключим тези МОФ и т. н.), която няма добър затворен вид и не променя принципно даденото съотношение.

За съжаление, нито едно от условията, които формулирахме, не се явява необходимо и достатъчно едновременно. Затова ще разгледаме случаите, в които са налице само необходимите условия. Нека функцията $f \in \mathcal{F}_2^n$ е зададена чрез нейната 3-КНФ:

$$f(x_1, x_2, \ldots, x_n) = D_1 \wedge D_2 \wedge \ldots \wedge D_r, \ 8 \leq r \leq 7m$$

и нека тя съдържа поне една клауза от S_0 и поне една клауза от S_3. Не ни е известен критерий, който да проверява за полиномиално време дали $f = \tilde{0}$ или не. За да проверим дали клаузите на f покриват целия n-мерен булев куб, можем да използваме принципа за включване и изключване за множествата $Z_{D_1}, Z_{D_2}, \ldots, Z_{D_r}$ (както в Лема 3.10, която е частен случай за $r = 2$). Получаваме:

$$|Z_f| = |Z_{D_1 \wedge D_2 \wedge \ldots \wedge D_r}| = |Z_{D_1} \cup Z_{D_2} \cup \ldots \cup Z_{D_r}| = \quad (3.3)$$

$$= \sum_{i=1}^{r} |Z_{D_i}| - \sum_{1 \leq i < j \leq r} |Z_{D_i} \cap Z_{D_j}| + \ldots + (-1)^{k-1} \sum_{1 \leq i_1 < \ldots < i_k \leq r} |Z_{D_{i_1}} \cap Z_{D_{i_2}} \cap \ldots \cap Z_{D_{i_k}}| +$$

$$+ \ldots + (-1)^{r-1} |Z_{D_1} \cap Z_{D_2} \cap \ldots \cap Z_{D_r}|.$$

Да разгледаме събираемите от вида $|Z_{D_{i_1}} \cap Z_{D_{i_2}} \cap \ldots \cap Z_{D_{i_k}}|$ при $m < k \leq r$, т. е. когато имаме повече клаузи, отколкото могат да бъдат образувани от n фиксирани букви на променливи или техни отрицания. Тогава някоя от клаузите ще съдържа отрицание на променлива, която се съдържа в друга клауза, т. е. всяко от тези събираеми съдържа поне две клаузи D_{i_p} и D_{i_q}, такива, че ако x_j участва в D_{i_p}, то $\bar{x}_j$ участва в D_{i_q}. Съгласно Лема 3.9, $|Z_{D_{i_p}} \cap Z_{D_{i_q}}| = 0$ и следователно $|Z_{D_{i_1}} \cap Z_{D_{i_2}} \cap \ldots \cap Z_{D_{i_k}}| = 0$. Тогава във формула (3.3) от определено място до края всички събираеми ще бъдат нули и формулата се редуцира до

$$|Z_f| = |Z_{D_1 \wedge D_2 \wedge \ldots \wedge D_r}| = |Z_{D_1} \cup Z_{D_2} \cup \ldots \cup Z_{D_r}| = \quad (3.4)$$

$$= \sum_{i=1}^{r} |Z_{D_i}| - \sum_{1 \leq i < j \leq r} |Z_{D_i} \cap Z_{D_j}| + \ldots + (-1)^{m-1} \sum_{1 \leq i_1 < \ldots < i_m \leq r} |Z_{D_{i_1}} \cap Z_{D_{i_2}} \cap \ldots \cap Z_{D_{i_m}}|.$$

След отпадането на толкова (евентуално много) събираеми е редно да се запитаме дали това е достатъчно, за да може да изчислим формулата (3.4) за полиномиално време. Отговорът е "не" (поне засега) – на базата

на биномиалните оценки, дадени в [37] и по-конкретно $\binom{n}{k} \geq (\frac{n}{k})^k$ ще дадем една груба оценка за броя на събиранията и изважданията във формула (3.4). Полагаме $r = 7m$ и за броя им получаваме:

$$\binom{7m}{1} + \binom{7m}{2} + \ldots + \binom{7m}{m} \geq \left(\frac{7m}{1}\right)^1 + \left(\frac{7m}{2}\right)^2 + \ldots + \left(\frac{7m}{m}\right)^m > 7^m =$$

$$= (\sqrt[8]{7})^{8m} \approx (1,275)^{8m}.$$

В заключение ще отбележим, че всяко от необходимите и всяко от достатъчните условия, които формулирахме, може да бъде проверено чрез алгоритъм (в повечето случаи тривиален) за полиномиално време спрямо големината на входа. Но оценките, които изведохме (за броя на удовлетворимите и на неудовлетворимите 3-КНФ – съответно (3.1) и (3.2)), показват, че за полиномиално време могат да бъдат решени твърде малка част от екземплярите на задачата 3-УБФ. Специално за тази задача може да бъде направен още един извод – при нея е в сила моделът "лесни – трудни – лесни", описан в раздел 3.1.2. Стойностите 8 и $7m$ за броя на клаузите в 3-КНФ на входа можем да считаме за критични, те маркират началото и края на фазов преход с много широк диапазон, в който попадат мнозинството от екземплярите на задачата 3-УБФ и това са точно трудните екземпляри.

Изборът на подход за изследване в тази глава и някои от резултатите се потвърдиха впоследствие в [34, 63, 51]. Така например композицията от преименуване на променлива и замяна на променлива с нейно отрицание е еквивалент на понятието "преименуване на литерал", НОКФ на $\tilde{0}$ е минимална по отношение на брой клаузи и е еквивалент на понятието "минимална неудовлетворима формула", операцията отрязване на буква, разглеждана като изображение на C_n^0 в себе си, е еквивалент на понятието "ендоморфизъм", а неприводима форма – на "ядро на формула". Твърдения, аналогични на Лема 3.9 и Лема 3.10 са формулирани (пак като леми) и използвани в [51].

Глава 4

Преброяване на m-ични разбивания от определен вид

При изследванията в предишната глава често ставаше дума за покрития и разбивания на n-мерния булев куб. В опитите за покриване на куба с възможно минимален брой клаузи достигнахме до задачата за разбиване на $\{0,1\}^n$ на подкубове, покривани от клаузите и до преброяване на възможните разбивания с отчитане на pn-еквивалентностите. Оказа се, че тази задача е свързана с разбиванията на числото 2^n по степените на двойката и преброяването им. Първите резултати по тази задача са публикувани в [3]. При последвалите проучвания достигнахме до по-общата задача за разбиванията на числото m^n по степените на самото число m (т. нар. m-ични разбивания) и тяхното преброяване, или още по-общо до преброяване на m-ичните разбивания на суми от k събираеми, всяко от които е равно на m^n. В тази глава представяме изследванията ни по тази задача. Обзорът в първи раздел включва необходимите понятия, постановки и постигнатите резултати. Във втори раздел предлагаме два алгоритъма за определяне броя на m-ичните разбивания на такива суми, основани на стратегията динамично програмиране. С тяхна помощ в трети раздел извеждаме някои алгебрични и комбинаторни свойства на функцията на разбиване. Четвърти раздел е посветен на съществуването и получаването на полиномиални форми, представящи броя на тези разбивания. Свойствата от трети раздел и полиномиалните форми от четвърти раздел, се явяват основа за построяване на друг алгоритъм, с времева сложност $\Theta(n^3)$, който първо определя коефициентите на полиномиалното представяне, а после пресмята броя на разглежданите разбивания. В пети раздел са дадени две приложения, свързани с изброяване на: (1) определени видове m-ични дървета и (2) разбиванията на n-мерния булев куб.

Част от представените изследвания, са докладвани на Националния годишен семинар по теория на кодирането и приложения (2000 г.), организиран от Секция МОИ на ИМИ при БАН. Всички изследвания и резултати, включени в тази глава, са приети за публикуване в [27].

4.1 Основни понятия и постигнати резултати

Чърчхаус, изследвайки задачата за представяне на естественото число n като сума от степени на числото 2, означава с $b(n)$ броя на начините за такова представяне и нарича $b(n)$ *функция на двоичното разбиване* [36]. Той посочва Ойлер, Тантури, Малер и др., които са изследвали тази функция (следващите формули (4.1), (4.2) и (4.3), за $m = 2$, са открити от Ойлер). Чърчхаус е един от първите, който използва компютър за пресмятане на $b(n)$ за $0 \leq n \leq 200$ и за да докаже някои тъждества и сравнимости за функцията на двоично разбиване. Те са доразвити, обобщени, както и някои нови свойства (най-вече за сравнимост) са получени от Андрюс [22, 23], Рьодсет [59] и Гупта [49]. Рьодсет означава с $t_m(n)$ броя на разбиванията на естественото число n в нестрого растяща редица от събираеми, всяко от които представлява неотрицателна степен на дадено естествено число $m > 1$ и нарича $t_m(n)$ *функция на m-ичното разбиване.* Той полага $t_m(0) = 1$ и $F_m(x) = \sum_{n=0}^{\infty} t_m(n)x^n$ за $|x| < 1$. В сила са следващите четири равенства ([36, 49, 59]):

$$F_m(x) = \prod_{k=0}^{\infty}(1 - x^{m^k})^{-1} \tag{4.1}$$

и следователно $F_m(x)$ удовлетворява функционалното уравнение

$$(1 - x)F_m(x) = F_m(x^m). \tag{4.2}$$

Тогава

$$t_m(n) = t_m(n-1) + t_m\left(\left\lfloor \frac{n}{m} \right\rfloor\right) \quad \text{и} \tag{4.3}$$

$$t_m(mn + k) = t_m(mn) \text{ за } 0 \leq k < m. \tag{4.4}$$

В тази глава винаги **ще считаме, че m, n и k са естествени числа, такива че** $m > 1, n > 0$. Ще разглеждаме разбивания от вида

$$m^n = m^{i_1} + m^{i_2} + \ldots + m^{i_r}, \quad i_j \geq 0 \text{ за } j = 1, 2, \ldots r. \tag{4.5}$$

Както в [19, 59], всяко разбиване на числото m^n от вида (4.5) ще разглеждаме като комбинация с повторения, чиито елементи (събираемите) образуват монотонно растяща редица:

$$m^n = m^{i_1} + m^{i_2} + \ldots + m^{i_r}, \quad 0 \leq i_1 \leq i_2 \leq \ldots i_r < n, \text{ при } r > 1,$$

или $i_1 = n$ при $r = 1$.

Дефиниция 4.1 Разбиване от вида (4.5), което съдържа точно k ($k > 0$) събираеми, се нарича *k-разбиване* ([22, 15, 19]). При $k = 1$, разбиването $m^n = m^n$ се нарича *тривиално разбиване.* Всички останали k-разбивания (при $k > 1$) се наричат *нетривиални разбивания.*

Записваме равенство (4.4) във вида

$$t_m(mn - 1) = t_m(mn - 2) = \ldots = t_m(m(n-1)), \tag{4.6}$$

и го заместваме в (4.3):

$$t_m(mn) = t_m(m(n-1)) + t_m(n). \tag{4.7}$$

Чрез повтаряне на заместванията на (4.3) и (4.6) в (4.7) (както Чърчхаус – за $m = 2$ в [36]) получаваме:

$$t_m(mn) = \sum_{i=0}^{n} t_m(i). \tag{4.8}$$

Въпреки усилията и търсенията, не можахме да решим или да намерим готово решение на рекурентното уравнение (4.8). Това уравнение, заедно с равенствата (4.3), (4.6), (4.7) и началното условие $t_m(0) = 1$ определят алгоритъм за пресмятане стойността на $t_m(mn)$ за дадени m и n. Този алгоритъм ще наречем кратко *Алгоритъм А*. Неговият рекурсивен *Maple*-код, основан на формула (4.3), е даден в [72] – за получаване на редиците *A*000123 (разбивания на $2n$ по степени на 2) и *A*002577 (разбивания на 2^n по степени на 2). А за редиците *A*005704 (разбивания на $3n$ по степени на 3), *A*005705 (разбивания на $4n$ по степени на 4) и *A*005706 (разбивания на $5n$ по степени на 5) пораждащи функции, рекурентни уравнения, или *Maple*-код не са представени.

Както е показано в [43, 72] и както ще забележим по-късно, функцията на m-ично разбиване нараства изключително бързо. Една точна оценка на времевата сложност на Алгоритъм А и на останалите алгоритми трябва да се основава на критерия за логаритмична цена (вж. раздел 2.3), понеже междинните резултати на всяка стъпка и крайният резултат (изходът) имат експоненциално нарастваща големина спрямо големината на входа m и n. Това изисква вграждане на алгоритъм за работа с достатъчно големи числа, както и да бъде направена прецизна оценка на времевата сложност на аритметичните операции, които той реализира (като функция на размера на данните на всяка стъпка). Тази задача е встрани от основната идея в тази глава. Вместо това ще акцентираме само върху броя и вида на аритметичните операции, които даден алгоритъм изпълнява. По този начин възприемаме критерия за еднаква цена (вж. раздел 2.3), който е реален при малки стойности на входните данни, за които една компютърна дума е достатъчна за получаване и съхраняване на изхода. Имаме основания да считаме, че изводите, които ще направим при сравняване ефективността на алгоритмите, ще бъдат валидни и в останалите случаи. Основният ни аргумент е, че в различните алгоритми, които ще изследваме, можем да вградим един и същи алгоритъм за работа с големи числа, който ще изпълнява различен брой основни аритметични операции с предварително зададени цени. Тогава по-ефективен ще бъде този алгоритъм, който изпълнява по-малко операции и такива с по-ниска цена.

Входни данни за Алгоритъм А са числата n и m, а негов изход е стойността на $t_m(mn)$. За реализиране на Алгоритъм А написахме няколко версии на нерекурсивна програма, подобрявайки ефективността на всяка следваща версия. Последната от тях пресмята стойността на $t_m(mn)$ като изпълнява $\left\lfloor \frac{n}{m} \right\rfloor$ събирания и същия брой целочислени деления (за пресмятане на индекси). Приемаме, че това е минималният брой аритметични операции за програма, основана на споменатите вече равенства. Нека събирането, целочисленото деление и съхраняването на резултат се изпълняват за единица време: съответно $c_1 = const$, $c_2 = const$ и $c_3 = const$,

Доказателство.

1) За $n = 1$ и за произволно k твърдението следва пряко от Лема 4.2. За $n \geq 1$ единственото представяне $k.m^n = (km).m^{n-1}$ е вярно за всяко k и формално за $k = 0$ – ето защо полагаме $\tilde{t}(n, 0) = 1$.

2) Нека $n > 1$ и $k > 0$. От Лема 4.2 следва

$$k.m^n = (k-1).m^n + m^n = (k-1).m^n + m.m^{n-1}.$$

Да разгледаме последните m събираеми. Фиксираме част от тях и разбиваме останалите – последователно, отляво надясно, така че всеки път да получаваме монотонно растяща редица от събираеми. Броят на фиксираните събираеми варира от m до 0 и в зависимост от този брой разпределяме всички разбивания на сумата $k.m^n$ в $m+1$ непресичащи се помежду си групи.

Първата група съдържа всички разбивания, в които последните m събираеми са равни на m^{n-1}, а всяко друго събираемо е по-малко или равно на m^{n-1}. Понеже $k.m^n = (k-1).m^n + m.m^{n-1}$, първа група съдържа общо $\tilde{t}(n, k-1)$ разбивания (можем да ги разглеждаме като получени от разбиванията на сумата $(k-1).m^n$, като в края на всяко нейно разбиване сме добавили фиксираната последователност от събираеми). Във втора група участват всички разбивания, в които последните $m-1$ събираеми са равни на m^{n-1}, а всяко друго събираемо е по-малко от m^{n-1}. Тъй като $k.m^n = (k-1).m^n + m.m^{n-1} = ((k-1)m+1).m^{n-1} + (m-1).m^{n-1}$, втора група съдържа общо $\tilde{t}(n-1, (k-1)m+1)$ разбивания (могат да бъдат получени от разбиванията на сумата $((k-1)m+1).m^{n-1}$, като в края на всяко нейно разбиване добавим фиксираната редица $(m-1).m^{n-1}$) и т. н. Разбиванията от последната група не съдържат нито едно фиксирано събираемо, всички събираеми са по-малки от m^{n-1}. От равенството $k.m^n = ((k-1)m + m - 1).m^{n-1} + m^{n-1} = (km).m^{n-1}$ следва, че броят им е $\tilde{t}(n-1, km)$.

Следователно броят на всички нетривиални разбивания на сумата $k.m^n$ по степени на m е сума от броя на разбиванията във всички групи, т. е.

$$\tilde{t}(n,k) = \tilde{t}(n,k-1) + \tilde{t}(n-1,(k-1)m+1) + \tilde{t}(n-1,(k-1)m+2) + \ldots + \tilde{t}(n-1,km). \diamond$$

3) Пресмятане стойността на решението по метода отдолу-нагоре.

Не е трудно да се напише рекурсивна програма, основана на равенство (4.9), която да пресмята стойността на $\tilde{t}(n,k)$ за дадени входни данни m, n и k. Тя ще пресметне първо $\tilde{t}(n,k-1)$, след което $\tilde{t}(n-1,(k-1)m+1)$, и т. н., накрая $\tilde{t}(n-1,km)$. Съгласно равенство (4.9), пресмятането на $\tilde{t}(n-1,km)$ ще предизвика преизчисляване на $t(n-1,km-1)$, след това преизчисляване на $\tilde{t}(n-1,km-2)$ и т. н. Забелязваме, че рекурсивният алгоритъм решава едни и същи подзадачи повече от веднъж[1], т. е. оригиналната задача притежава свойството *припокриване на подзадачите* – вторият ключов елемент за прилагане на стратегията динамично програмиране. Ето защо ще разработим следващия алгоритъм по метода отдолу-нагоре. За целта използваме таблица (масив) $\tilde{T}$ с n реда, номерирани с $1, 2, \ldots, n$.

[1]Това важи и за подобна рекурсивна програма, реализираща Алгоритъм А.

Всяко число $\tilde{t}(i,j)$ запомняме в клетката $\tilde{T}[i,j]$ от таблицата. Да определим броя на колоните ѝ. Понеже $\tilde{t}(i,0) = 1$ за $1 \leq i \leq n$, нулевата колона ще съдържа единици. В последния (n-тия) ред остава да се пресметнат и попълнят стойностите на $\tilde{t}(n,1), \tilde{t}(n,2), \ldots, \tilde{t}(n,k)$. Останалите клетки в реда са несъществени и могат да останат празни (непопълнени). За да попълним клетките в последния ред трябва да са известни $1+km$ числа от съответните клетки на $(n-1)$-вия ред (останалите негови клетки отново са несъществени и могат да останат непопълнени) и т. н. По този начин можем индуктивно да покажем, че в i-тия ред ($1 \leq i \leq n$) трябва да се пресметнат и попълнят стойностите на $\tilde{t}(i,j)$, $0 \leq j \leq km^{n-i}$, или общо $1+km^{n-i}$ числа. Тогава първият ред на таблицата ще съдържа $1+km^{n-1}$ числа, които са единици – понеже $\tilde{t}(1,j) = 1$ за $j \geq 0$. Следователно в таблицата $\tilde{T}$ имаме $1+km^{n-1}$ колони, които номерираме с $0, 1, \ldots, km^{n-1}$ в съответствие със стойностите на j.

Следващият алгоритъм, който пресмята стойността на решението по метода отдолу-нагоре, ще наречем *Алгоритъм B.*

Алгоритъм B: пресмята броя на всички нетривиални разбивания на сумата $k.m^n$ по степени на m.

Вход: m, n и k.

Изход: стойността на $\tilde{t}(n,k)$.

Процедура:

1) Дефинираме таблица $\tilde{T}$ с n реда и $km^{n-1}+1$ колони.

2) Попълваме всички клетки от първия ред и от нулевата колона на $\tilde{T}$ с единици.

3) В цикъл по редовете, от втория до n-тия, обхождаме клетките от текущия ред – от първата (с номер едно) до последната съществена клетка, която трябва да бъде попълнена (тя има номер km^{n-i} в i-тия ред). За всяка текуща клетка $\tilde{T}[i,j]$ пресмятаме поредната стойност на $\tilde{t}(i,j)$ (съгласно равенство (4.9)) и запомняме резултата в нея.

4) Връщаме като резултат числото, записано в клетката $\tilde{T}[n,k]$. Край.

Когато m, n и k са произволни естествени числа, съответната им таблица означаваме с $\tilde{T}_{m,n,k}$. Ето как изглеждат таблица $\tilde{T}_{2,5,1}$ и част от таблица $\tilde{T}_{3,5,1}$.

$i\backslash j$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	3	5	7	9	11	13	15	17								
3	1	9	25	49	81												
4	1	35	165														
5	1	201															

Таблица 4.1: $\tilde{T}_{2,5,1}$.

$i\backslash j$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	...
2	1	4	7	10	13	16	19	22	25	28	31	34	37	40	43	...
3	1	22	70	145	247	376	532	715	925	1162						...
4	1	238	1393	4195												...
5	1	5827														...

Таблица 4.2: Част от таблица $\tilde{T}_{3,5,1}$.

4.2.2 Втори случай: всички разбивания на сумата $k.m^n$ по степени на m

Този случай е подобен на първия, но е по-общ. Разглеждаме всички разбивания на сумата $k.m^n$ по степени на m, в които всяко събираемо е по-малко или равно на m^n, т. е. позволени са тривиални разбивания на събираемите в сумата. Означаваме броя на тези разбивания с $t_m(n,k)$, или само с $t(n,k)$, когато говорим за дадено число m.

1) Характеризиране на броя на разбиванията.

Както в първия случай, Лема 4.2 обуславя наличието на свойството оптимална подструктура.

2) Рекурсивно решение.

Рекурсивното решение на задачата е дадено в следващата теорема.

Теорема 4.4 *Нека са дадени естествените числа n, m и k. Тогава е валидна рекурентната зависимост:*

$$t(n,k) = \begin{cases} k+1 & \text{ако } n = 1, \\ 1 & \text{ако } k = 0, \\ t(n,k-1) + t(n-1,km) & \text{ако } n > 1 \text{ и } k > 0. \end{cases} \tag{4.10}$$

Доказателство.

1) При $n = 1$, за сумата $k.m^1$ имаме едно тривиално разбиване плюс още k разбивания, понеже всяко нейно събираемо може да бъде разбито само веднъж (Лема 4.2). Тогава $t(1,k) = 1 + k$.

2) Сумата $k.m^n$ има единствено тривиално разбиване $k.m^n = k.m^n$ за всяко n и k. Формално, това равенство е валидно при $k = 0$ и полагаме $t(n,0) = 1$.

3) Нека $n > 1$ и $k > 0$. Разделяме всички разбивания на сумата $k.m^n$ на две непресичащи се групи.

В първа група са разбиванията, които съдържат поне едно тривиално разбиване, т. е. поне едно от събираемите в сумата е равно на m^n. Разглеждаме всички разбивания на сумата $(k-1).m^n$ по степени на m и добавяме m^n в края на всяко от тях. Така получаваме всички разбивания от първа група и следователно броят им е $t(n,k-1)$.

Разбиванията от втора група не съдържат в себе си нито едно тривиално разбиване, т. е. групата се състои от всички разбивания от първия случай. Според Лема 4.2 имаме $k.m^n = (km).m^{n-1}$ и тогава втора група включва общо $t(n-1,km)$ разбивания.

Следователно $t(n,k) = t(n,k-1) + t(n-1,km)$. $\diamond$

3) Пресмятане стойността на решението по метода отдолу-нагоре.

Очевидно е, че задачата притежава свойството припокриване на подзадачите. Следващия алгоритъм ще разработим на базата на рекурентното уравнение (4.10), като пак прилагаме метода отдолу-нагоре. Използваме таблица T с n реда, номерирани с $1, 2, \ldots, n$. Всяко число $t(i,j)$ записваме в клетката $T[i,j]$. Аналогично на предишния случай можем да докажем индуктивно, че в клетките от i-тия ред $(1 \leq i \leq n)$ на T трябва да

се пресметнат и запишат стойностите на $t(i,0), t(i,1), \ldots, t(i, km^{n-i})$. Тогава таблица T има $1 + km^{n-1}$ колони, които номерираме с $0, 1, \ldots, km^{n-1}$.

Следващият алгоритъм пресмята стойността на решението по метода отдолу-нагоре. Наричаме го кратко *Алгоритъм С.*

Алгоритъм С: пресмята броя на всички разбивания на сумата $k.m^n$ по степени на m.

Вход: m, n и k.

Изход: стойността на $t(n,k)$.

Процедура:

1) Дефинираме таблица T с n реда и $km^{n-1}+1$ колони.

2) Попълване на първия ред: за всяко j, $1 \leq j \leq km^{n-1}$, пресмятаме числото $j+1$ и го записваме в клетката $T[1,j]$.

3) Попълване на нулевата колона: записваме единици във всички нейни клетки.

4) Попълване на останалите съществени клетки: в цикъл по редовете, от втория до n-тия, обхождаме клетките от текущия ред – от първата (с номер едно) до последната съществена клетка, която трябва да бъде попълнена (тя има номер km^{n-i} в i-тия ред). За всяка текуща клетка $T[i,j]$ пресмятаме поредната стойност на $t(i,j)$ (според равенство (4.10)) и запомняме резултата в нея.

4) Връщаме като резултат числото, записано в клетката $T[n,k]$. Край.

Когато говорим за произволни m, n и k, съответната им таблица означавме с $T_{m,n,k}$. По-долу са представени части от таблиците $T_{2,5,2}$ и $T_{3,5,2}$.

$i\backslash j$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
2	1	4	9	16	25	36	49	64	81	100	121	144	169	196	225	...
3	1	10	35	84	165	286	455	680	969							...
4	1	36	201	656	1625											...
5	1	202	1827													...

Таблица 4.3: Част от $T_{2,5,2}$.

$i\backslash j$	0	1	2	3	4	5	6	7	8	9	10	11	12	...
1	1	2	3	4	5	6	7	8	9	10	11	12	13	...
2	1	5	12	22	35	51	70	92	117	145	176	210	247	...
3	1	23	93	238	485	861	1393	2108	3033	4195	5621	7338	9373	...
4	1	239	1632	5827	15200	32856	62629							...
5	1	5828	68457											...

Таблица 4.4: Част от $T_{3,5,2}$.

Бележка 4.5 *При таблиците $T_{m,n,k}$ и $\tilde{T}_{m,n,k}$, функционалната зависимост между номера на реда i, $1 \leq i \leq n$ и броя на попълнените клетки в него r, е $r = km^{n-i}$, т. е. тя е от експоненциално намаляващ тип. С други думи, редовете в таблиците имат експоненциално намаляваща дължина и за попълването им е необходимо експоненциално намаляващо време.*

Да пресметнем времевите сложности на Алгоритмите B и C (според възприетия критерий за еднаква цена) за дадени входни данни m, n и k.

Алгоритъм B попълва общо $km^{n-1}+n$ единици в първия ред и нулевата колона на таблицата $\tilde{T}$. Алгоритъмът пресмята и попълва общо $k(m^{n-2}+m^{n-3}+\ldots+m^1+m^0)=k(m^{n-1}-1)/(m-1)$ числа в останалите клетки на $\tilde{T}$. За пресмятане на всяко от тези числа се изпълняват m събирания и следователно времевата сложност на Алгоритъм B е $c_3(km^{n-1}+n)+(c_1+c_3)km(m^{n-1}-1)/(m-1)=\Theta(km^{n-1})$.

Алгоритъм C попълва n единици в нулевата колона на таблицата T, а също пресмята и попълва $km^{n-1}, km^{n-2}, \ldots, km^0$ числа съответно в първия, втория, $\ldots, n$-тия ред. Пресмятането на всяко от тези числа става само чрез едно събиране. Получаваме, че алгоритъм C изпълнява общо $k(m^{n-1}+m^{n-2}+\ldots+m^0)=k(m^n-1)/(m-1)$ събирания и следователно времевата му сложност е $c_3n+(c_1+c_3)km^{n-1}=\Theta(km^{n-1})$.

4.3 Свойства на числата от таблиците

Попълването на таблиците $\tilde{T}_{m,n,k}$ и $T_{m,n,k}$ много прилича на попълването на други таблици, основани на рекурентни зависимости – например триъгълниците на Паскал, на Стирлинг, на Ойлер. Подобни таблици са т. нар. *аритметични триъгълници, аритметични квадрати* и техните обобщения, резглеждани в [4]. В аритметичните квадрати нулевият ред и нулевата колона съдържат единици, а всички останали клетки съдържат числа, които са сума на числата, записани в клетката вляво и в определени клетки над текущата. Числата от аритметичните триъгълници и квадрати са свързани с:

- различни комбинаторни задачи за движение на шахматни фигури върху шахматна дъска;
- т. нар. *фигурни числа* (триъгълни, квадратни и т. н.; пирамидални). Основите на тяхното изучаване са поставени от Питагор и Никомах;
- триъгълника на Паскал в различните му разновидности.

Числата, попълнени в таблиците $\tilde{T}_{m,n,k}$ и $T_{m,n,k}$, притежават редица интересни свойства. Те са подобни на свойствата на числата на Паскал, на Стирлинг, на Ойлер [7] и на свойствата на числата от аритметичните триъгълници и квадрати [4], както може да се очаква. Тези свойства произтичат от дадените дефиниции и доказаните твърдения и ще ги формулираме като следствия. Разглеждаме таблиците $\tilde{T}$ и T, получени чрез изпълнение съответно на Алгоритъм B и Алгоритъм C с входни данни естествените числа m, n и k.

От разгледаните вече случаи пряко следва:

Следствие 4.6 *Броят на нетривиалните разбивания на числото m^n от вида (4.5) е равен на $\tilde{t}(n,1)$, определено чрез равенство (4.9). Броят на всички разбивания на същото число и от същия вид е равен на $t(n,1)$, определено чрез равенство (4.10).*

Отбелязваме, че $t(n,1)=\tilde{t}(n,1)+1$ – в случая $k=1$ и разликата между първия и втория вид разбивания е само в тривиалното разбиване. Тогава Алгоритъм B и Алгоритъм C пресмятат стойността на $t_m(m^n)=t(n,1)=$

$T[n,1] = \tilde{t}(n,1) + 1 = \tilde{T}[n,1] + 1$ с една и съща времева сложност – $\Theta(m^{n-1})$. Алгоритъм А пресмята стойността на $t_m(m^n)$ с по-добра времева сложност – $\Theta(m^{n-2})$.

Таблица 4.5 е получена чрез изпълнение на Алгоритъм С (в който събирането е над 12-байтови компютърни думи) четири пъти: за $n = 9$ и за $m = 2, 3, 5, 7$. В нея са представени числата от първите колони на съответните таблици $T_{m,9,1}$, т. е. стойностите на $t_m(n, 1)$ за $1 \leq n \leq 9$.

$n \backslash m$	2	3	5	7
1	2	2	2	2
2	4	5	7	9
3	10	23	82	205
4	36	239	3707	24901
5	202	5828	642457	16077987
6	1828	342383	446020582	58169810617
7	27338	50110484	1288155051832	1226373476385199
8	692004	18757984046	15905066118254957	154912862345527456431
9	30251722	18318289003448	856874264098480364332	11967977955077323244243580

Таблица 4.5: Стойностите на $t_m(n, 1)$ за $1 \leq n \leq 9$ и за $m = 2, 3, 5, 7$.

Както дефинирахме, стойността в клетката $T[i, j]$ представя броя на всички разбивания на сумата $j.m^i$ по степени на m, за $1 \leq i \leq n$. Към това добавяме:

Следствие 4.7 *Числото, записано в клетката $T[i, j]$ е равно на броя на всички разбивания на сумата $j.m^l$ по степени на m, в които всяко събираемо е по-голямо или равно на m^{l-i}, за $1 \leq i \leq l \leq n$.*

Доказателство. Разглеждаме всички разбивания на сумата $j.m^i$ (по степени на m) и съпоставяме на всяко от тях разбиване на сумата $j.m^l$ (по степени на m), в което всяко събираемо е по-голямо или равно на m^{l-i}. Втория вид разбивания получаваме, като умножим двете страни на всяко равенство, представящо разбиване от първия вид, по m^{l-i}. Очевидно, това съпоставяне е биективно и следователно броят на тези два вида разбивания е един и същи, а именно $T[i, j]$. $\diamond$

Аналогично, Следствие 4.7 може да бъде формулирано и за броя на нетривиалните разбивания, записан в клетката $\tilde{T}[i, j]$.

От равенство (4.9) имаме:

$\tilde{T}[1, j] = 1 \quad$ за $\quad j = 0, 1, 2, \ldots, km^{n-1}$,

$\tilde{T}[i, 1] = 1 \quad$ за $\quad i = 1, 2, \ldots, n$,

$$\tilde{T}[i, j] = \tilde{T}[i, j-1] + \sum_{l=1}^{m} \tilde{T}[i-1, (j-1)m + l] \quad \text{за} \quad 1 < i \leq n, \quad 1 \leq j \leq km^{n-i}.$$

Таблици, съдържащи числа, свързани помежду си със зависимости като току-що посочените, са наречени *рекурентни таблици* в [4]. Ако $j > 1$, заместваме $\tilde{T}[i, j-1]$ с $\tilde{T}[i, j-2] + \sum_{l=1}^{m} \tilde{T}[i-1, (j-2)m + l]$, след това заместваме $\tilde{T}[i, j-2]$ и т. н., докато достигнем до $\tilde{T}[i, 0] = 1$. Накрая получаваме:

Следствие 4.8 $\tilde{T}[i,j] = 1 + \sum_{l=1}^{mj} \tilde{T}[i-1,l] = \sum_{l=0}^{mj} \tilde{T}[i-1,l]$ *за* $1 < i \leq n$.

Бележка 4.9 *Твърдението от Следствие 4.8 е доказано от Чърчхаус [36] за* $m = 2$. *Той представя правоъгълна таблица, която напълно съответства на таблицата* $\tilde{T}_{2,5,1}$, *дадена по-горе. Като обобщава Теорема 1 в [36], Гупта [49] показва, че за* $m > 2$, $t_m(m^r.n) = \sum_{j=0}^{n} C_r(j) t_m(n-j)$, *където коефициентите* $C_{r+1}(j) = \sum_{i=0}^{mj} C_r(i)$ *и* $C_1(j) = 1$ *за всяко* $j \geq 0$. *Това е друго свойство на числата от таблиците* $\tilde{T}_{m,n,k}$, *получено чрез аналитични техники. Изяснявайки смисъла на коефициентите* $C_i(j)$ *отбелязваме, че това са точно числата от таблиците* $\tilde{T}_{m,n,k}$, *представящи броя на съответните разбивания.*

От равенство (4.10) имаме:
$T[1,j] = j+1$ за $j = 0,1,2,\ldots,km^{n-1}$,
$T[i,1] = 1$ за $i = 1,2,\ldots,n$,
$T[i,j] = T[i,j-1] + T[i-1,jm]$ за $1 < i \leq n$, $1 \leq j \leq km^{n-i}$.
Ако $j > 1$, заместваме $T[i,j-1]$ с $T[i,j-2] + T[i-1,(j-1)m]$, след което заместваме $T[i,j-2]$ и т. н., докато достигнем до $T[i,0] = 1$. Така получаваме:

Следствие 4.10 $T[i,j] = 1 + \sum_{l=1}^{j} T[i-1,lm] = \sum_{l=0}^{j} T[i-1,lm]$ *за* $1 < i \leq n$.

Съществува връзка между таблиците T и $\tilde{T}$ за дадени m,n и k, тъй като разбиванията от първия случай се явяват част от разбиванията от втория случай. От доказателството на Теорема 4.4 директно следва:

Следствие 4.11 (Връзка между таблиците $\tilde{T}$ и T)
а) $\tilde{T}[i,j] = T[i-1,jm]$ *и* $T[i,j] = T[i,j-1] + \tilde{T}[i,j]$ *за* $1 < i \leq n$;
б) ако игнорираме първия ред на таблицата $\tilde{T}_{m,n,k}$, *тогава всяка нейна колона с номер* j $(0 \leq j \leq km^{n-2})$ *съвпада с колоната с номер* jm *на таблицата* $T_{m,n-1,k}$.

Аритметичен ред от степен m наричаме редицата от стойности на полином от степен m: $p(x) = a_0 + a_1 x + \ldots + a_m x^m$, получени за $x = 0,1,2,\ldots$. Числата от първия ред на $\tilde{T}$ образуват аритметичен ред от степен 0. Числата от първия ред на T, както и тези от втория ред на $\tilde{T}$, образуват аритметични редове от степен 1 (т. е. аритметични прогресии). Фигурните (по-точно многоъгълните) числа образуват аритметични редове от степен 2. Ако $q \geq 3$ е фиксирано цяло число, то n-то q-ъгълно число се изразява чрез известната формула: $P_n^q = n + (q-2)n(n-1)/2$, $n = 1,2,\ldots$.

Следствие 4.12 (Връзка между таблиците и многоъгълните числа)
а) Ако $n > 2$, *първите* $km^{n-2}+1$ *на брой* $(m+2)$*-ъгълни числа са разположени във втория ред на таблицата* $T_{m,n,k}$, *т. е.* $T_{m,n,k}[2,j] = P_{j+1}^{m+2}$ *за* $j = 0,1,\ldots,km^{n-2}$;
б) Ако $n > 3$, *определен вид* $(m+2)$*-ъгълни числа са разположени в третия ред на таблицата* $\tilde{T}_{m,n,k}$, *а именно* $\tilde{T}_{m,n,k}[3,j] = P_{mj+1}^{m+2}$ *за* $j = 0,1,\ldots,km^{n-3}$.

Доказателство. а) $T_{m,n,k}[1,j] = 1 + j$ за $j = 0, 1, \ldots, km^{n-1}$. За втория ред на таблицата имаме $T_{m,n,k}[2,0] = 1$ и за $j = 1, 2, \ldots, km^{n-2}$ чрез Следствие 4.10 получаваме:

$$T_{m,n,k}[2,j] = \sum_{l=0}^{j} T_{m,n,k}[1,lm] = \sum_{l=0}^{j}(1 + lm) = j + 1 + mj(j+1)/2.$$

Следователно $T_{m,n,k}[2,j] = P_{j+1}^{m+2}$ за $j = 0, 1, \ldots, km^{n-2}$.

б) Твърдението следва от Следствие 4.11 и от а). $\diamond$

4.4 Полиномиално представяне на броя на разбиванията

В този раздел ще докажем съществуването на полиномиално представяне на броя на разбиванията на сумата $k.m^n$ по степени на m. След това ще представим алгоритъм за получаване на съответния полином и за пресмятане броя на разглежданите разбивания.

4.4.1 Съществуване на полиномиалното представяне

Лема 4.13 *Нека m и n са положителни естествени числа. Тогава сумата $0^n + 1^n + \ldots + m^n = \sum_{k=0}^{m} k^n$ е полином на m от степен $n+1$, а коефициентът му пред m^{n+1} е $1/(n+1)$.*

Доказателство. Полагаме $s_m = 0^n + 1^n + \ldots + m^n$. Тогава $s_m - s_{m-1} = m^n$. Това нехомогенно рекурентно уравнение (с начално условие $s_0 = 0$) може да бъде решено по метода на неопределените коефициенти [48]. Общото решение има вида $s_m = s_m^{(h)} + s_m^{(p)}$, където $s_m^{(h)}$ е общото решение на съответното хомогенно рекурентно уравнение $s_m - s_{m-1} = 0$, а $s_m^{(p)}$ е частно решение на самото нехомогенно уравнение. Следователно $s_m^{(h)} = c$, $c = const$, а $s_m^{(p)}$ има следния полиномиален вид: $s_m^{(p)} = m(a_0 + a_1 m + \ldots + a_n m^n)$. Заместваме $s_m^{(p)}$ в нехомогенното уравнение и като сравняваме коефициентите пред еднаквите степени на m от двете страни на равенството получаваме: $a_i = a_i(n)$, $i = 0, 1, \ldots, n$ и $a_n = 1/(n+1)$. След определяне на константата c от началното условие получаваме, че решението s_m е полином на m от степен $n+1$ и коефициентът пред m^{n+1} е $a_n = 1/(n+1)$. $\diamond$

Бележка 4.14 *Общата формула, която представя такива суми е формулата на Бернули $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1}\sum_{k=0}^{n}\binom{n+1}{k} B_k m^{n+1-k}$, където коефициентите B_k са числата на Бернули [7]. Те се дефинират чрез рекурентната зависимост $B_0 = 1$, $\sum_{j=0}^{n}\binom{n+1}{j} B_j = 0$ за всяко $n > 0$ и не могат да бъдат представени в затворен вид. Тогава $B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}$, $B_3 = 0 = B_{2k+1}$ за $k > 0$, $B_4 = -\frac{1}{30}, B_6 = \frac{1}{42}$ и т. н. Най-общо, сумата $S(m,n) = \sum_{k=1}^{m} k^n$ може да бъде изразена чрез ζ-функцията на Риман.*

Теорема 4.15 *Нека са дадени естествените числа m, n и k и нека i е фиксирано цяло, $1 \leq i \leq n$. Тогава стойността на всяка клетка $T[i,j]$, за $j = 0, 1, \ldots, km^{n-i}$, може да бъде представена чрез полином на две променливи $P_i(m,j)$, в който едночленът от най-висока степен е $\frac{1}{i!}m^{\frac{i(i-1)}{2}}j^i$, и в него променливите m и j са от максимални степени.*

Доказателство. Ще докажем твърдението на теоремата чрез математическа индукция по i, като използваме равенството от Следствие 4.10: $T[i,j] = \sum_{l=0}^{j} T[i-1, lm]$.

1) Знаем, че $T[1,j] = j+1$ за всяко $j = 0, 1 \ldots, km^{n-1}$. В доказателството на Следствие 4.12 а) показахме, че $T[2,j] = 1 + \frac{1}{2}(2+m)j + \frac{1}{2}mj^2$ за всяко $j = 0, 1 \ldots, km^{n-2}$. Следователно, за $i = 1$ и за $i = 2$ теоремата е вярна.

2) Да предположим, че за $i = s$, $i < n$ и за $\forall\, j = 0, 1, \ldots, km^{n-s}$, $P_s(m,j)$ са полиноми от споменатия вид, представящи стойността на съответните клетки $T[s,j]$.

3) Нека $i = s+1$ и да разгледаме стойността на клетката $T[s+1,j]$ за произволно j, $0 \leq j \leq km^{n-s-1}$. Тогава, в съответствие с индуктивното предположение, имаме:

$$T[s+1,j] = \sum_{l=0}^{j} T[s, lm] = \sum_{l=0}^{j} P_s(m, lm).$$

Това е сума от полиноми и следователно тя също е полином $P_{s+1}(m,j)$.

а) Съгласно индуктивното предположение, разглеждаме всяко $P_s(m,j)$, $0 \leq j \leq km^{n-s}$, като полином на j от степен s, т. е. $P_s(m,j) = p_0 + p_1 j + p_2 j^2 + \ldots + p_s j^s$. Коефициентите му $p_l = p_l(m)$ са полиноми на m за $l = 0, 1, \ldots, s$ и $p_s(m) = \frac{1}{s!}m^{\frac{s(s-1)}{2}}$. Тогава:

$$\begin{aligned}
P_{s+1}(m,j) &= \sum_{l=0}^{j} P_s(m, lm) = \sum_{l=0}^{j} (p_0 + p_1.(lm) + \ldots + p_s.(lm)^s) = \\
&= p_0 \sum_{l=0}^{j} 1 + p_1 m \sum_{l=0}^{j} l + \ldots + p_s m^s \sum_{l=0}^{j} l^s = \\
&= \ldots + p_s m^s \sum_{l=0}^{j} l^s = \ldots + \frac{1}{s!}m^{\frac{s(s-1)}{2}} m^s \sum_{l=0}^{j} l^s \quad \overset{\text{Лема 4.13}}{=} \\
&= \ldots + \frac{1}{s!}m^{\frac{s(s+1)}{2}} (a_0 + a_1 j + \ldots + a_s j^s + \frac{1}{s+1} j^{s+1}) = \\
&= \ldots + \frac{1}{(s+1)!}m^{\frac{s(s+1)}{2}} j^{s+1} \ ,
\end{aligned}$$

където $a_l = a_l(s)$, $l = 0, 1, \ldots s$, и следователно твърдението е вярно в разглеждания случай.

б) Сега, съгласно индуктивното предположение, разглеждаме всяко $P_s(m,j)$, $0 \leq j \leq km^{n-s}$, като полином на m от степен $r = s(s-1)/2$, т. е. $P_s(m,j) = q_0 + q_1 m + q_2 m^2 + \ldots + q_r m^r$. Коефициентите му $q_l = q_l(j)$ са

полиноми на j за $l = 1, 2, \ldots, r$ и $q_r(j) = \frac{1}{s!}j^s$. Тогава:

$$\begin{aligned} P_{s+1}(m,j) &= \sum_{l=0}^{j} P_s(m,lm) = \sum_{l=0}^{j}(q_0(lm) + q_1(lm).m + \ldots + q_r(lm).m^r) = \\ &= \sum_{l=0}^{j} q_0(lm) + m\sum_{l=0}^{j} q_1(lm) + \ldots + m^r \sum_{l=0}^{j} q_r(lm) = \\ &= \ldots + m^r \sum_{l=0}^{j} \frac{1}{s!}(lm)^s = \ldots + \frac{1}{s!}m^r m^s \sum_{l=0}^{j} l^s \overset{\text{Лема 4.13}}{=} \\ &= \ldots + \frac{1}{s!}m^{r+s}(a_0 + a_1 j + \ldots + a_s j^s + \frac{1}{s+1}j^{s+1}) = \\ &= \ldots + \frac{1}{(s+1)!}m^{\frac{s(s+1)}{2}} j^{s+1}\ , \end{aligned}$$

където $a_l = a_l(s)$, $l = 0, 1, \ldots, s$ и следователно твърдението е вярно и във втория случай, с което теоремата е доказана. $\diamond$

Теорема 4.16 *Нека са дадени естествените числа m, n и k и нека i е фиксирано цяло, $1 \leq i \leq n$. Тогава стойността на всяка клетка $\tilde{T}[i,j]$, за $j = 0, 1, \ldots, km^{n-i}$, може да бъде представена чрез полином на две променливи $\tilde{P}_i(m,j)$, в който едночленът от най-висока степен е $\frac{1}{(i-1)!}m^{\frac{i(i-1)}{2}} j^{i-1}$ и в него променливите m и j са от максимални степени.*

Вместо доказателство отбелязваме, че твърдението на Теорема 4.16 следва пряко от Следствие 4.11. Теоремата може да бъде доказана и по начин, аналогичен на доказателството на Теорема 4.15, като се използва Следствие 4.8.

Бележка 4.17

а) От алгебрична гледна точка, Теорема 4.15 установява съществуването на фамилия от полиноми на две променливи $P_i(m,j)$, $i = 1, \ldots, n$, представящи редовете на таблицата $T_{m,n,k}$.

б) От геометрична гледна точка, същата теорема установява съществуването на фамилия от алгебрични повърхнини $S_i : z = P_i(m,j)$, $i = 1, \ldots, n$. Техните сечения с равнината $m = const$ са криви, съдържащи (като точки) стойностите на съответните редове от таблицата $T_{m,n,k}$. Аналогично, сеченията им с равнината $j = const$ представят дискретно стойностите на една и съща клетка $T[i,j]$ от различни таблици $T_{m,n,k}$ – когато i е фиксирано, а m се мени.

Теорема 4.16 установява валидността на а) и б) за таблиците $\tilde{T}_{m,n,k}$ и за съответните им полиноми $\tilde{P}_i(m,j)$.

4.4.2 Получаване на полиномите, преставящи броя на разбиванията

Теорема 4.15 и Теорема 4.16 показват някои характеристики на полиномите, представящи броя на разбиванията на сумата $k.m^n$ по степени на m, но те не дават явния вид на самите полиноми – понеже полиномите от Лема 4.13 нямат представяне в затворен вид [7]. Като използваме Лема

4.13 и доказателството на Теорема 4.15, ще покажем как се получават полиномите $P_n(m,j)$ за първите няколко стойности на n. Вече ни е известно, че $P_1(m,j) = j+1$ и $P_2(m,j) = 1 + \frac{1}{2}(2+m)j + \frac{1}{2}mj^2$. Чрез Лема 4.13 получаваме: $\sum_{l=0}^{j} l^2 = \frac{j}{6} + \frac{j^2}{2} + \frac{j^3}{3}$, и следователно:

$$P_3(m,j) = \sum_{l=0}^{j} P_2(m,ml) = \sum_{l=0}^{j}(1 + \frac{2+m}{2}ml + \frac{m}{2}(ml)^2) = \sum_{l=0}^{j} 1 + \frac{2m+m^2}{2}\sum_{l=0}^{j} l +$$

$$+\frac{m^3}{2}\sum_{l=0}^{j} l^2 = \ldots = 1 + \frac{1}{12}(12+6m+3m^2+m^3)j + \frac{1}{4}m(2+m+m^2)j^2 + \frac{1}{6}m^3j^3 \ .$$

Полиномите $\tilde{P}_n(m,j)$ можем да получим по същия начин, или още по-лесно – само чрез Следствие 4.11. Например, ако в последния израз (за $P_3(m,j)$) заместим j с mj, получаваме полинома, представящ четвъртия ред на таблицата $\tilde{T}_{m,n,k}$:

$$\tilde{P}_4(m,j) = 1 + \frac{1}{12}m(12+6m+3m^2+m^3)j + \frac{1}{4}m^3(2+m+m^2)j^2 + \frac{1}{6}m^6j^3.$$

За $m=2$ получаваме

$$P_3(2,j) = 1 + \frac{11}{3}j + 4j^2 + \frac{4}{3}j^3 = \frac{(2j+1)(2j+2)(2j+3)}{6} \ ,$$

$$\tilde{P}_4(2,j) = 1 + \frac{22}{3}j + 16j^2 + \frac{32}{3}j^3 = \frac{(4j+1)(4j+2)(4j+3)}{6}.$$

Тетраедралното число с пореден номер l, $l = 1,2,\ldots$, се представя чрез формулата $l(l+1)(l+2)/6$. Към връзката между таблиците и фигурните числа (Следствие 4.12) добавяме:

Следствие 4.18 (Връзка между таблиците и тетраедралните числа)
а) Ако $n > 3$, в третия ред на таблицата $T_{2,n,k}$ са разположени тетраедралните числа с нечетни поредни номера.

б) Ако $n > 4$, в четвъртия ред на таблицата $\tilde{T}_{2,n,k}$ са разположени тетраедралните числа с поредни номера от вида $l = 4j+1$, $j = 0,1,\ldots,2^{n-4}$.

Следващите два полинома (за $n=4$ и за $n=5$) са получени чрез пакета Mathematica 2.2:

$$\begin{aligned} P_4(m,j) = 1 &+ \frac{1}{24}(24+12m+6m^2+5m^3+2m^4+m^5)j + \frac{1}{24}m(12+6m+9m^2+4m^3+3m^4+m^5)j^2 + \\ &+ \frac{1}{12}m^3(2+m+m^2+m^3)j^3 + \frac{1}{24}m^6j^4 \ , \end{aligned}$$

$$\begin{aligned} P_5(m,j) = 1 &+ \frac{1}{720}(720+360m+180m^2+150m^3+105m^4+75m^5+36m^6+15m^7+5m^8-m^{10})j + \\ &+ \frac{1}{48}m(24+12m+18m^2+11m^3+11m^4+7m^5+4m^6+2m^7+m^8)j^2 + \\ &+ \frac{1}{72}m^3(12+6m+9m^2+10m^3+6m^4+4m^5+3m^6+m^7)j^3 + \\ &+ \frac{1}{48}m^6(2+m+m^2+m^3+m^4)j^4 + \frac{1}{120}m^{10}j^5 \ . \end{aligned}$$

Полагаме $j=1$ в по-горните полиноми и получаваме съответно полиноми на m, представящи броя на разбиванията на числото m^n от вида (4.5), за $n = 1,2,\ldots,5$:

$P_1(m,1) = 2$,

$P_2(m,1) = 2+m$,

$P_3(m,1) = \frac{1}{2}(4+2m+m^2+m^3)$,

$$P_4(m,1) = \frac{1}{12}(24 + 12m + 6m^2 + 9m^3 + 4m^4 + 3m^5 + 2m^6) ,$$
$$P_5(m,1) = \frac{1}{24}(48 + 24m + 12m^2 + 18m^3 + 11m^4 + 11m^5 + 9m^6 + 5m^7 + 3m^8 + 2m^9 + m^{10}).$$

Можем да получим следващия полином $P_6(m,1)$ без да имаме полинома $P_6(m,j)$. От равенство (4.10) имаме $T_{m,n,k}[6,1] = 1 + T_{m,n,k}[5,m]$. Заместваме j с m в $P_5(m,j)$, добавяме 1 и получаваме:

$$\begin{aligned} P_6(m,1) &= \frac{1}{720}(1440 + 720m + 360m^2 + 540m^3 + 330m^4 + 375m^5 + 360m^6 + 260m^7 + \\ &+ 210m^8 + 165m^9 + 120m^{10} + 69m^{11} + 45m^{12} + 25m^{13} + 15m^{14} + 6m^{15}). \end{aligned}$$

Бележка 4.19 *Според Теорема 4.15 съществува полином $P_n(m,k)$, чрез който се изразява стойността на $t_m(n,k)$ за фиксирано n. Всеки такъв полином, чиито явен вид е известен, може да бъде пресметнат (за дадени m и k) чрез подходящо прилагане на правилото на Хорнер*

$$Q(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + xa_n))\ldots)$$

към всяка от променливите m и k на полинома. Тогава изчисляването на $P_n(m,k)$ се изпълнява за полиномиално време $\Theta(n.n(n-1)/2) = \Theta(n^3)$, докато алгоритмите B и C решават същата задача за експоненциално време. Общата задача, която решават алгоритмите A, B и C, е пресмятането на броя на всичките разбивания на числото m^n от вида (4.5) при дадени входни данни m и n. Съществува полином на m от степен $n(n-1)/2$, който представя този брой. Когато явният му вид е известен, пресмятането на $t_m(m^n) = t_m(n,1) = P_n(m,1)$ се изпълнява за полиномиално време $\Theta(n^2)$, за разлика от експоненциалната времева сложност $\Theta(m^{n-2})$ на Алгоритъм A за пресмятане на същата стойност.

Алгоритмите B и C могат да бъдат ускорени чрез прилагане на смесена стратегия. Например, ако $n > 5$ използваме полинома $P_5(m,j)$ за да пресметнем и попълним клетките от петия ред на таблицата T. Предполагаме, че коефициентите пред степените на j в $P_5(m,j)$ са известни за даденото m (в противен случай ги пресмятаме за време $\Theta(n^3)$). Прилагаме правилото на Хорнер и пресмятаме стойността на всяка клетка $T[5,j]$, $j = 1,2\ldots,km^{n-5}$ – така попълването на петия ред се извършва за време $\Theta(km^{n-5})$. Редовете от шестия до края на таблицата T попълваме по начина, описан в процедурата на Алгоритъм C и общата цена за това е $\Theta(km^{n-6})$ единици от време. Следователно, при дадени m,n $(n > 5)$ и k на входа, модифицираният Алгоритъм C пресмята стойността на $t_n(n,k)$ с времева сложност $\Theta(km^{n-5})$. Той пресмята стойността на $t_m(m^n)$ за време $\Theta(m^{n-5})$, което е по-добро от времето, необходимо на Алгоритъм A за пресмятане същата стойност.

Отбелязваме, че в общия случай, предложената модификация на Алгоритъм C свежда експоненциалната му времева сложност до по-добра, но пак експоненциална. Можем да предложим и промени по отношение на използваната памет, които (доколкото се основават на таблица с експоненциален брой стълбове) свеждат сложността по памет към по-добра, но пак експоненциална. Вместо това ще пресметнем броя на разглежданите разбивания (в общия случай) за полиномиално време и с полиномиален

разход на памет, като за целта предлагаме следващия алгоритъм:

Алгоритъм D: определя полинома, който представя броя на всички разбивания на сумата $k.m^n$ по степени на m и пресмята този брой.
Вход: m, n и k.
Изход: коефициентите на съответния полином $P_n(m, j)$, получени за дадените m и n и стойността му за $j = k$.
Процедура:

1) Получаване на тригълника на Паскал: пресмятаме и попълваме биномните коефициенти в масива c, така че $c[i, j] = \binom{i}{j}$, за $0 \leq i \leq n+1$, $0 \leq j \leq i$.

2) Получаване на числата на Бернули: пресмятаме (съгласно Бележка 4.14) и попълваме числата на Бернули в масива b, така че $b[i] = B_i$, $0 \leq i \leq n$.

3) В съответствие с Лема 4.13 и Бележка 4.14, пресмятаме коефициентите на полиномите $Q_i(j) = q_{i,0} + q_{i,1}j + \ldots + q_{i,i+1}j^{i+1}$, представящи сумата $0^i + 1^i + \ldots + j^i$ и ги запомняме в i-тия ред на масива q, за $i = 0, 1, \ldots, n-1$.

4) Следвайки доказателството на Теорема 4.15, пресмятаме коефициентите на полиномите $P_i(m, j) = p_{i,0} + p_{i,1}j + \ldots + p_{i,i}j^i$ (където $p_{i,l} = p_{i,l}(m)$, $0 \leq l \leq i$) и ги запомняме в i-тия ред на масива p, за $i = 1, 2, \ldots, n$.

5) Прилагаме правилото на Хорнер и пресмятаме стойността на полинома $P_n(m, j)$ за $j = k$ и я връщаме като резултат (при нужда се връщат и желаните коефициенти от масива p). Край.

Сега да поясним и анализираме изпълнението на стъпките на Алгоритъм D.

Стъпка 1 може да се реализира чрез известната формула: $\binom{i}{j} = 1$, ако $j = 0$ или $j = i$, или $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$, ако $0 < j < i$. Тогава пресмятането и попълването на биномните коефициенти в масива c има цена (по време и по памет) $\Theta(n^2)$.

В стъпка 2 (според Бележка 4.14) попълваме $b[0] = 1$, $b[1] = -1/2$ и $b[i] = 0$ за всички нечетни i, $3 \leq i \leq n$. За всички четни i, $2 \leq i \leq n$, пресмятаме $B_i = \dfrac{-1}{i+1} \displaystyle\sum_{j=0}^{i-1} \binom{i+1}{j} B_j$ и ги попълваме в клетката $b[i]$. Всички биномни коефициенти и числа на Бернули, необходими за пресмятането на B_i, са вече попълнени в масивите c и b, съответно. Цялото изпълнение на стъпка 2 има времева цена $\Theta(n^2)$ и цена по памет $\Theta(n)$.

При изпълнение на стъпка 3 използваме формулата на Бернули

$$0^i + 1^i + \ldots + j^i = \sum_{r=0}^{j} r^i = \frac{1}{i+1} \sum_{r=0}^{i} \binom{i+1}{r} B_r (j+1)^{i+1-r}$$

за да получим явния вид на полиномите от Лема 4.13, за $i = 0, 1, \ldots, n-1$. Резултатът, т. е. коефициентите пред степените на j, запомняме в масива q с n реда (номерирани според стойностите на i) и $n+1$ колони (номерирани според степените на j от 0 да n). В клетката $q[i, l]$ ще записваме коефициента пред j^l в полинома, който представя сумата $0^i + 1^i + \ldots + j^i$. В частност, при $i = 0$, от формулата на Бернули получаваме $0^0 + 1^0 + \ldots + j^0 = \ldots = 1 + j$ и затова инициализираме $q[0, 0] = 1, q[0, 1] = 1$ и $q[0, l] = 0$ за

$l = 2, 3, \ldots, n$. Дефинираме помощен масив w с n реда (номерирани от 0 да $n-1$) и $n+2$ колони (номерирани от 0 до $n+1$), като последната колона е спомагателна. В цикъл (за $i = 1, 2, \ldots, n-1$) изпълняваме следното:

а) инициализираме $w[r, l] = 0$ за $0 \leq r \leq i,\ 0 \leq l \leq i+1$;

б) за всяко $r = 0, 1, \ldots, i$ пресмятаме стойността на $\binom{i+1}{r} B_r$ (като използваме съответните числа от масивите c и b) и я записваме в клетката $w[r, n+1]$ от последния стълб ;

в) за всяко $r = 0, 1, \ldots, i$ проверяваме стойността на $w[r, n+1]$. Ако тя е различна от нула, вземаме биномните коефициенти в развитието на $(1+j)^{i+1-r}$ от $(i+1-r)$-тия ред на масива c. Умножаваме всеки от тях по $w[r, n+1]$ и запомняме резултата в съответната клетка от r-тия ред на масива w;

г) за всяко $l = 0, 1, \ldots, i+1$ сумираме числата от l-тата колона на масива w (т. е. $w[0, l] + w[1, l] + \ldots + w[i, l]$) и разделяме тази сума на $(i+1)$ – получаваме коефициента $q_{i,l}$ в полинома $Q_i(j)$ и го запомняме в клетката $q[i, l]$.

Очевидно, всички операции от а) до г) можем да изпълним за време $\Theta(i^2)$ за дадено i. Алгоритъмът ги повтаря за всяко $i = 1, \ldots, n-1$ и следователно времевата сложност на стъпка 3 е $\Theta(1^2) + \Theta(2^2) + \ldots + \Theta((n-1)^2) = \Theta(1^2 + 2^2 + \ldots + (n-1)^2) = \Theta(n^3)$. Броят на клетките в масивите q и w има порядък $\Theta(n^2)$ и това е оценката за сложността по памет.

В стъпка 4 използваме доказателството на Теорема 4.15 и разгледаните примери. Стъпка 4 е аналогична на стъпка 3 и се основава на факта, че ако са известни коефициентите на полинома $P_i(m, j) = a_0 + a_1 j + \ldots + a_i j^i$, тогава коефициентите на следващия полином $P_{i+1}(m, j)$ могат да бъдат пресметнати по формулата

$$P_{i+1}(m, j) = \sum_{l=0}^{j} P_i(m, ml) = \sum_{l=0}^{j} (a_0 + a_1(ml) + \ldots + a_i(ml)^i) =$$

$$= \sum_{l=0}^{j} a_0 + a_1 m \sum_{l=0}^{j} l + \ldots + a_i m^i \sum_{l=0}^{j} l^i.$$

Използвайки полиномите, получени в стъпка 3, развиваме последното равенство и пресмятаме коефициентите на $P_{i+1}(m, j)$, за $i = 1, \ldots, n-1$. Запомняме ги в масива p с n реда (номерирани според стойностите на i) и $n+1$ колони (номерирани според степените на j), така че клетката $p[i, l]$ да съдържа коефициента пред j^l в полинома $P_i(m, j)$. Понеже $P_1(m, j) = 1 + j$, инициализираме $p[1, 0] = 1, p[1, 1] = 1$ и $p[1, l] = 0$ за $l = 2, 3, \ldots, n$. В помощен масив u пресмятаме и попълваме стойностите на степените на m, така че $u[i] = m^i$, за $i = 0, \ldots, n-1$. Отново ползваме помощния масив w от стъпка 3. В цикъл (за $i = 1, 2, \ldots, n-1$) определяме коефициентите на $P_{i+1}(m, j)$ като изпълняваме:

а) инициализираме $w[r, l] = 0$, за $0 \leq r \leq i,\ 0 \leq l \leq i+1$;

б) за всяко $r = 0, 1, \ldots, i$ умножаваме a_r по m^r и запомняме резултата в клетката $w[r, n+1]$ (като използваме коефициентите от i-тия ред на масива p и стойностите от масива u);

в) за всяко $r = 0, 1, \ldots, i$ вземаме коефициентите от r-тия ред на масива q, умножаваме всеки от тях по $w[r, n+1]$ и запомняме резултата в съответната клетка от r-тия ред на масива w;

г) за всяко $l = 0, 1, \ldots, i+1$ сумираме числата от l-тата колона на масива w (т. е. $w[0,l]+w[1,l]+\ldots+w[i,l]$) и полученият резултат е коефициентът $p_{i+1,l}$ в полинома $P_{i+1}(m,j)$ – запомняме го в клетката $p[i+1,l]$.

Анализът на сложността в стъпка 4 е аналогичен на този в стъпка 3. Отново получаваме оценките: $\Theta(n^3)$ за времевата сложност и $\Theta(n^2)$ за сложността по памет.

В стъпка 5 прилагаме правилото на Хорнер – като използваме коефициентите от n-тия ред на масива p пресмятаме стойността на полинома $P_n(m,j)$ за $j = k$.

Стъпка 5 се изпълнява за време $\Theta(n)$, като използва константно количество памет.

Коректността на Алгоритъм D следва пряко от твърденията и поясненията, които вече дадохме. Времевата сложноста на алгоритъма е сума от времевите сложности за изпълнение на петте му стъпки и следователно тя е $\Theta(n^3)$, според възприетия критерий за еднаква цена. Аналогично, сложността по памет е $\Theta(n^2)$. Получихме същия тип времева сложност, не зависеща от m, както в случаите когато е известен явният вид на полиномите $P_n(m,j)$. Пресмятането на стойността на $t_m(m^n) = t_m(n,1) = P_n(m,1)$ чрез Алгоритъм D се изпълнява отново за време $\Theta(n^3)$. Накрая отбелязваме, че в описанието на алгоритъма бяха пропуснати някои детайли, свързани с аритметиката над рационални числа. Те са включени в наша програма, реализираща Алгоритъм D, понеже коефициентите на полиномите са рационални числа и използване на данни от тип *real* би довело до грешки от закръгляне и тяхното натрупване. Оперирането с рационални числа и най-вече съкращаването на дроби забавят изпълнението на алгоритъма. При събиране, изваждане, умножение на дроби и при умножение на дроб с цяло число времевата сложност нараства с константен множител, но порядъкът ѝ не се променя. При съкращаването на дроби времевата цена зависи от размера на данните, представящи числителите и знаменателите и избраният критерий не е подходящ за нейната оценка.

4.5 Приложения

Съществуват различни комбинаторни задачи, чиито решения са свързани с разгледаните разбивания. Например, задачи за двоични разбивания възникват при разделянето и разпределянето на оперативната памет или на пространството на твърдите дискове. Задачи за m-ични разбивания възникват в редица други случаи – например при верижни реакции (химични, ядрени, радиоактивно разпадане при космичните лъчи, деления на клетки и др.) през даден период от време. Техните решения могат да бъдат илюстрирани чрез дървета, притежаващи определени свойства.

4.5.1 Преброяване на пълните m-ични дървета от определен вид

Доналд Кнут [12] обръща специално внимание на дърветата като информационни структури и поставя и/или решава редица изброителни задачи за определени видове дървета. Той отбелязва, че едно от най-непосредствените приложения на математическата теория за дърветата при изследване на алгоритми е свързано с формули за изброяване на различни типове дървета.

Това ни мотивира да приложим разгледаните разбивания за преброяване пълните m-ични дървета от определен вид. За целта започваме с една индуктивна (и конструктивна) дефиниция на понятието *пълно m-ично кореново дърво*, подобна на дефинициите за коренови дървета в [16].

Дефиниция 4.20

0) Графът $G_0 = (\{v_0\}, \emptyset)$ е *пълно m-ично кореново дърво* с корен v_0 и без ребра. G_0 наричаме *тривиално дърво.*

1) Графът $G_1 = (V_1, E_1)$, в който $V_1 = \{v_0, v_1, \ldots, v_m\}$ и $E_1 = \{(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_m)\}$, е *пълно m-ично кореново дърво* с корен v_0.

2) Нека $G_i = (V_i, E_i)$ е пълно m-ично дърво с корен $r \in V_i$ и $V_i \cap V_1 = \emptyset$. Тогава графът $G_{i+1} = (V_{i+1}, E_{i+1})$, в който $V_{i+1} = V_1 \cup V_i \backslash \{v_j\}$, $v_j \in V_i$ и $E_{i+1} = E_1 \cup (E_i \backslash \{(v_k, v_j)\}) \cup \{(v_k, v_0)\}$, също е *пълно m-ично кореново дърво* с корен r. Казваме, че G_{i+1} е получено от G_i чрез операцията "заместване на произволно листо v_j в G_i с дървото G_1".

3) Няма други пълни m-ични коренови дървета, освен G_0, G_1 и дърветата, получени чрез прилагане на операцията 2).

Добре известно е как се представят аритметични изрази чрез двоични коренови дървета. Разгледаните разбивания са също аритметични изрази с една единствена операция събиране, а събираемите са степени на m. Вместо двоични дървета избираме пълни m-ични коренови дървета, чрез които да представим m-ичните разбивания на числото m^n. С други думи, съпоставяме такова дърво на всяко разбиване на m^n от вида (4.5). Всяко листо на дървото надписваме със събираемо, а всеки вътрешен възел надписваме със знака "+", който разглеждаме като знак за m-аргументно събиране, понеже при разбиване на някое събираемо то винаги се замества от сума на m нови събираеми.

Първото разбиване на числото m^n е тривиалното разбиване. Според Лема 4.2, второто разбиване е $m^n = \underbrace{m^{n-1} + \ldots + m^{n-1}}_{m}$. Всяко следващо разбиване можем да получим последователно, като прилагаме отново Лема 4.2. На всяка стъпка (след второто разбиване) продължаваме с разбиване на някое събираемо, а ако имаме равни събираеми разбиваме най-лявото от тях. Така всеки път получаваме монотонно растяща редица от събираеми. Като следваме дадените дефиниции и пояснения, ще дефинираме процедура, с която съпоставяме пълно m-ично кореново дърво на всяко разбиване на числото m^n от вида (4.5).

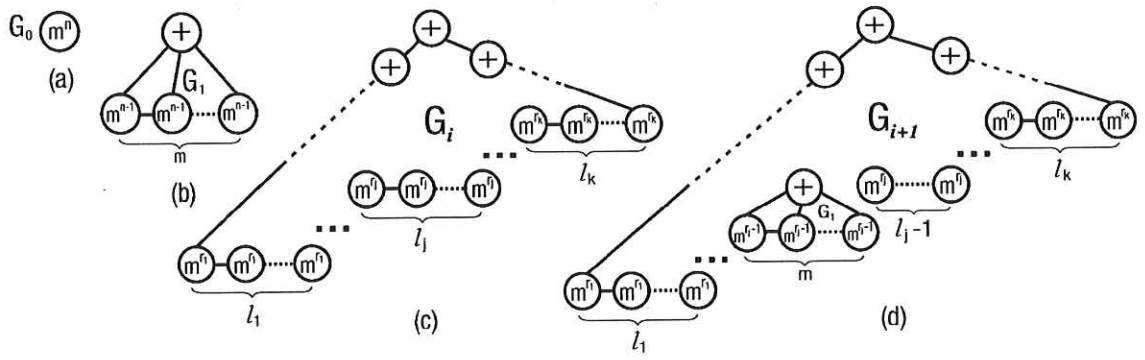
Процедура за съпоставяне А.

0) Тривиалното дърво G_0 съпоставяме на първото (тривиалното) разбиване $m^n = m^n$. Надписваме корена v_0 с m^n, както на Фиг. 4.1 (a).

1) Дървото G_1 съпоставяме на разбиването $m^n = \underbrace{m^{n-1} + \ldots + m^{n-1}}_{m}$. Надписваме корена v_0 с "+", а листата му с m^{n-1} – Фиг. 4.1 (b).

2) Нека $G_i = (V_i, E_i)$ е пълно m-ично кореново дърво, което е съпоставено на поредното разбиване $m^n = \underbrace{m^{r_1} + \ldots + m^{r_1}}_{l_1} + \ldots + \underbrace{m^{r_j} + \ldots + m^{r_j}}_{l_j} + \ldots + \underbrace{m^{r_k} + \ldots + m^{r_k}}_{l_k}$, т. е. вътрешните възли на G_i са надписани с "+", а листата му, отляво надясно, са надписани с $m^{r_1}, \ldots, m^{r_1}, \ldots, m^{r_j}, \ldots, m^{r_j}, \ldots, m^{r_k}, \ldots, m^{r_k}$, в съответствие с последователността им в разбиването – както на Фиг. 4.1 (c).

3) Дървото G_{i+1} съпоставяме на разбиването $m^n = \underbrace{m^{r_1} + \ldots + m^{r_1}}_{l_1} + \ldots + \underbrace{m^{r_j-1} + \ldots + m^{r_j-1}}_{m} + \underbrace{m^{r_j} + \ldots + m^{r_j}}_{l_j - 1} + \ldots + \underbrace{m^{r_k} + \ldots + m^{r_k}}_{l_k}$, което е получено от разбиването в 2) чрез заместване на най-лявото събираемо m^{r_j} със сумата $\underbrace{m^{r_j-1} + \ldots + m^{r_j-1}}_{m}$. Дървото G_{i+1} е получено от G_i чрез заместване на най-лявото му листо, съответстващо на m^{r_j}, с дървото G_1, в което коренът е надписан отново с "+", но листата му сега са надписани са m^{r_j-1} – Фиг. 4.1 (d).



Фиг. 4.1: Илюстрация на Процедура за съпоставяне А.

Означаваме с $\mathcal{T}$ множеството от всички дървета, които са резултат от Процедура за съпоставяне А при дадени m и n. Те се характеризират със следните три ограничения:

1) височината на всяко дърво не надминава n, понеже всяко събираемо (включително m^n) може да бъде разбивано до n пъти;

2) броят на листата (респ. събираемите) е от вида $k(m-1)+1$, за $k = 0, 1, \ldots, \frac{m^n-1}{m-1}$, понеже всеки път някое листо се трансформира във вътрешен възел и се появяват нови m листа. Дървото нараства общо с $m-1$ листа, минималният им брой е 1, а максималният е m^n;

3) разстоянията от корена до листата, разглеждани от ляво надясно, образуват монотонно намаляваща редица от числа (между n и 0), понеже събираемите във всяко разбиване формират монотонно растяща редица.

Теорема 4.21 *Ако m и n са естествени числа, $m > 1, n > 0$, то $t_m(m^n) = |\mathcal{T}|$.*
Доказателство. Означаваме с $\mathcal{P}$ множеството от всички разбивания на m^n от вида (4.5) и тогава $|\mathcal{P}| = t_m(m^n)$, а с $f : \mathcal{P} \to \mathcal{T}$ – функцията, дефинирана чрез Процедура за съпоставяне А. Да разгледаме две различни разбивания от $\mathcal{P}$ и да предположим, че пораждането на първото от тях (във веригата от последователни разбивания) приключва преди пораждането на второто. Тогава дървото, съпоставено на първото разбиване, е вече построено, докато дървото, съпоставено на второто разбиване, продължава да нараства. Следователно тези две дървета са различни и f е инекция. Всяко дърво от $\mathcal{T}$ е резултат от Процедура за съпоставяне А и значи съществува разбиване от $\mathcal{P}$, съответстващо на това дърво, т. е. f е сюрекция. Следователно f е биекция и $|\mathcal{P}| = |\mathcal{T}|$. $\diamond$

Така наречените *дървета на рекурсията* в [37] са дървета от $\mathcal{T}$. Там те се използват за визуализиране на итерациите, които изпълнява рекурсивен алгоритъм от вида "разделяй и владей". Пълни m-ични коренови дървета, притежаващи само първите две ограничения, могат да представят композициите (т. е. наредените разбивания; разбиванията, в които редът на събираемите е съществен [23]) на m^n по степени на m. Не е трудно да се докаже равенството между броя на тези композиции и броя на всички тези дървета.

4.5.2 Преброяване на разбиванията на n-мерния булев куб

Задачата за преброяване разбиванията на n-мерния булев куб беше първоначалната задача, чиито обобщения разгледахме в предишните части на тази глава. Повечето понятия и дефиниции, свързани с булевия куб, бяха дадени в раздел 1.2. Остава да дефинираме разбиванията му.

Дефиниция 4.22 *Разбиване* (или *k-разбиване*) на n-мерния булев куб на подкубове наричаме представянето $\{0,1\}^n = \{0,1\}^{r_1} \cup \{0,1\}^{r_2} \cup \ldots \cup \{0,1\}^{r_k}$, $0 \leq r_i < n$, $i = 1, 2, \ldots, k$, такова че $\{0,1\}^{r_i} \cap \{0,1\}^{r_j} = \emptyset$ за $1 \leq i < j \leq n$. Разбиването $\{0,1\}^n = \{0,1\}^n$ наричаме *тривиално разбиване*.

Очевидно, подкубовете във фиксирано k-разбиване могат да бъдат избирани по много различни начини, без да се променят размерностите им.

Дефиниция 4.23 Две k-разбивания на булевия куб наричаме *еквивалентни* (или *k-еквивалентни*), ако размерностите на подкубовете от едното разбиване съвпадат с тези от другото разбиване (тоест различават се едно от друго само по векторите, които образуват подкубовете им, но не и по броя им). И обратно: две k-разбивания *не са еквивалентни*, ако размерностите на техните подкубове, разглеждани като ненаредени k-орки, са различни.

Според казаното в предишната глава, различни подкубове се покриват от различни клаузи. При разбиване на n-мерния булев куб на подкубове и покриването им чрез клаузи, клаузите са такива, че $Z_{D_i} \cap Z_{D_j} = \emptyset$, $i \neq j$

за всеки два подкуба $\{0,1\}^{r_i} = Z_{D_i}$ и $\{0,1\}^{r_j} = Z_{D_j}$ от разбиването. Конюнктивната нормална форма, образувана от тези клаузи, е форма на $\tilde{0}$, по-точно тя е НОКФ. Ако имаме две еквивалентни k-разбивания, съответните им НОКФ са pn-еквивалентни. И обратно: ако разбиванията не са еквивалентни, то ненаредените k-орки от дължините на клаузите в съответните им НОКФ са различни и следователно тези НОКФ не са pn-еквивалентни. В предишната глава разглеждахме само такива НОКФ, които не са pn-еквивалентни една на друга и затова сега *ще разгледаме само неекивалентните помежду си разбивания на* $\{0,1\}^n$.

Лема 4.24 *За всяко естествено число* $n > 0$ *съществува единствено* 2*-разбиване на* $\{0,1\}^n$ *на два подкуба, всеки с размерност* $n-1$.
Доказателство. Дефиниция 1.13 на n-мерния булев куб дава съществуването на такова разбиване. Единствеността следва от дефинициите за разбиване 4.22, 4.23 и Лема 4.2 за $m = 2$. $\diamond$

Означаваме 2-разбиването от Лема 4.24 с $\{0,1\}^n = \{0,1\}_0^{n-1} \cup \{0,1\}_1^{n-1}$, където $\{0,1\}_0^{n-1}$ и $\{0,1\}_1^{n-1}$ са двата $(n-1)$-мерни подкуба, първият е образуван от всички вектори, започващи с нула, а вторият – от всички вектори, започващи с единица. Когато имаме 2-разбиване на подкуб с размерност r, $\{0,1\}^r = \{0,1\}_0^{r-1} \cup \{0,1\}_1^{r-1}$, $1 \leq r < n$, тогава $\{0,1\}_0^{r-1}$ означава всички вектори от $\{0,1\}^r$, в които първата нефиксирана позиция е 0, а $\{0,1\}_1^{r-1}$ – всички вектори от $\{0,1\}^r$, в които тази позиция е 1.

Следвайки дадените дефиниции и твърдения ще дефинираме процедура, с която съпоставяме k-разбиване на $\{0,1\}^n$ на всяко k-разбиване на числото 2^n от вида (4.5). Процедурата е аналогична на предишната.

Процедура за съпоставяне В.

0) Тривиалното разбиване на $\{0,1\}^n$ съпоставяме на тривиалното разбиване на числото 2^n.

1) 2-разбиването $\{0,1\}^n = \{0,1\}_0^{n-1} \cup \{0,1\}_1^{n-1}$ (от Лема 4.24) съпоставяме на 2-разбиването $2^n = 2^{n-1} + 2^{n-1}$ (от Лема 4.2).

2) Нека $\{0,1\}^n = \{0,1\}^{r_1} \cup \ldots \cup \{0,1\}^{r_i} \cup \ldots \cup \{0,1\}^{r_k}$ е k-разбиването на $\{0,1\}^n$, съпоставено на поредното k-разбиване $2^n = 2^{r_1} + \ldots + 2^{r_i} + \ldots + 2^{r_k}$ на числото 2^n, така че всеки r_i-мерен подкуб съответства на събираемото 2^{r_i}, $i = 1, 2, \ldots, k$. Размерностите на подкубовете, както и събираемите, образуват мнонотнно растяща редица.

3) $(k+1)$-разбиването $\{0,1\}^n = \{0,1\}^{r_1} \cup \ldots \cup \{0,1\}_0^{r_i-1} \cup \{0,1\}_1^{r_i-1} \cup \ldots \cup \{0,1\}^{r_k}$ съпоставяме на следващото $(k+1)$-разбиване $2^n = 2^{r_1} + \ldots + 2^{r_i-1} + 2^{r_i-1} + \ldots + 2^{r_k}$, получено чрез прилагане на Лема 4.2 към най-лявото събираемо 2^{r_i} в т. 2). $(k+1)$-разбиването на $\{0,1\}^n$ е получено чрез прилагане на Лема 4.24 към най-левия подкуб $\{0,1\}^{r_i}$ в т. 2).

Процедурата за съпоставяне В е конструктивна – дава начин за получаване на всички разбивания на $\{0,1\}^n$. Тя може да бъде илюстрирана чрез двоични дървета, така както и разбиванията на числото 2^n.

Теорема 4.25 *Нека* n *е естествено число,* $n > 0$. *Тогава броят на всички разбивания на* n*-мерния булев куб е равен на* $t_2(2^n)$.

Доказателство. Означаваме с g функцията, дефинирана чрез Процедура за съпоставяне B. Дефиниционна област на g е множеството от всички разбивания на числото 2^n от вида (4.5), а нейна кообласт е множеството от всички разбивания на $\{0,1\}^n$. Доказателството, че g е биекция е идентично на това, че f е биекция в Теорема 4.21 и следователно твърдението на теоремата е вярно. $\diamond$

В края на тази глава ще кажем, че предстои разглеждането на по-общи задачи от представените. А именно:

1) Да се определи броят на всички разбивания на суми от вида $a_1m^{n_1} + a_2m^{n_2} + \ldots + a_km^{n_k}$ (a_i и n_i са естествени числа, $i = 1, \ldots, k$) по степени на m.

2) Да се потърси връзка между разбиванията на такива суми и разгледаните вече разбивания. По-точно, ако $n = c_1m^{i_1} + c_2m^{i_2} + \ldots + c_rm^{i_r}$ е представянето на естественото число n в m-ична бройна система и разгледаме всички негови разбивания, дали има връзка между тях и разбиванията на m^n от вида (4.5) или с тези от зад. 1).

Надяваме се разгледаните в тази част разбивания да намерят приложения и в други изброителни задачи.

Глава 5

Изследвания на монотонните булеви функции

Задачите в областта на монотонните булеви функции (МБФ) са важни не само за булевата алгебра. Много от тях са свързани с (или имат пряка интерпретация в) задачи от теория на графите (хиперграфите), теория на веригите, изкуствения интелект, праговата логика, теория на изчислителното изучаване (computational learning theory (COLT)), теория на игрите и др. [24, 39, 42, 55]. Редица задачи за МБФ остават все още нерешени, а при други от тях все още остава открит въпросът за сложността им. При някои от тях възможностите на методите и средствата, прилагани за решаването им, са до голяма степен изчерпани и известни изследователи препоръчват търсенето и прилагането на нови структури, средства и техники [33, 55].

В тази глава са представени нашите изследвания върху три от най-важните задачи в областта на МБФ. В първи раздел са дадени основните понятия и означения, свързани с МБФ. Във втори раздел е предложена една матрична структура, дефинирана над векторите от n-мерния булев куб, извеждат се нейни комбинаторни и алгебрични свойства. Тези свойства са в основата на трите алгоритъма, представени в тази глава. В трети раздел е описан алгоритъм за генериране на всички МБФ на n променливи в лексикографски ред – във връзка със задачата на Дедекинд за определяне броя на МБФ на n променливи. В четвърти раздел е даден алгоритъм за определяне на минимален истинен и максимален неистинен вектор на неизвестна монотонна функциия чрез т. нар. "въпроси за членство", използвани в теория на изчислителното изучаване. Съпоставяме го на популярния и често използван алгоритъм на Гайнанов [6, 33, 55]. В последния раздел се разглежда задачата за идентифициране на неизвестна монотонна функция чрез въпроси за членство, която е обект на засилен интерес от много изследователи. Предлага се алгоритъм за решаването на тази задача, представени са експерименталните резултати от работата му. Оказва се, че за МБФ на не повече от шест променливи алгоритъмът разпознава неизвестна монотонна функция, като използва полиномиален (спрямо общата големина на входа и изхода) брой въпроси.

Изследванията в тази глава са докладвани на IV конференция "Приложения на математиката в икономиката и проблеми на обучението по математика и информатика" (1995 г.), на Международната конференция по групи и графи (2002 г.), на Националния годишен семинар по теория на кодирането и приложения (2002 г.), организиран от Секция МОИ на ИМИ при БАН и на XXXII Пролетна конференция на СМБ (2003 г.). Получените резултати са публикувани в [2, 28, 29].

5.1 Основни понятия и означения

В този раздел допълваме понятията от първа глава и ги конкретизираме спрямо МБФ. В литературата по темата не всички понятия и означения са общоприети и затова ще приемем най-често използваните от тях.

Дефиниция 5.1 Булевата функция $f \in \mathcal{F}_2^n$ наричаме *монотонна*, ако за всяка двойка $\alpha, \beta \in \{0,1\}^n$, такива че $\alpha \preceq \beta$, е изпълнено $f(\alpha) \leq f(\beta)$.

Означаваме с M^n множеството от всички монотонни функции на n променливи, а с $M = M^0 \cup M^1 \cup \ldots \cup M^n \cup \ldots$ – множеството от всички МБФ. Например, $\tilde{0}$ и $\tilde{1}$, разглеждани като функции на нула променливи, са единствените функции в M^0; $f(x) = x \in M^1$, но $f(x) = \bar{x} \notin M^1$; $f(x,y) = xy \in M^2$, $f(x,y) = x \vee y \in M^2$, но $f(x,y) = x \oplus y \notin M^2$ и т. н. Множеството M е един от петте основни класа на Пост [16, 9, 20], т. е. за него са в сила:

1) $f(x) = x \in M$;

2) ако $f, g_1, g_2, \ldots, g_n \in M$, то $f(g_1, g_2, \ldots, g_n) \in M$, или суперпозиция на монотонни функции е пак монотонна функция.

Дефиниция 5.2 Нека $f \in \mathcal{F}_2^n$ и $\alpha \in \{0,1\}^n$. Векторът α наричаме *истинен вектор* на f, ако $f(\alpha) = 1$, или *неистинен вектор* на f, ако $f(\alpha) = 0$.

За дадена функция $f \in \mathcal{F}_2^n$, означаваме с $T(f) = \{\alpha | \alpha \in \{0,1\}^n, f(\alpha) = 1\}$ и с $F(f) = \{\alpha | \alpha \in \{0,1\}^n, f(\alpha) = 0\}$ съответно множествата от всички истинни и от всички неистинни вектори на f. Ако $T(f) = \emptyset$, то $f = \tilde{0}$, а ако $F(f) = \emptyset$, то $f = \tilde{1}$.

Дефиниция 5.3 Векторът $\alpha \in T(f)$ наричаме *минимален истинен вектор* (МИВ), ако не съществува вектор $\alpha' \in T(f)$, такъв че $\alpha' \preceq \alpha$ и $\alpha' \neq \alpha$. Векторът $\beta \in F(f)$ наричаме *максимален неистинен вектор* (МНВ), ако не съществува вектор $\beta' \in F(f)$, такъв че $\beta \preceq \beta'$ и $\beta \neq \beta'$.

Означаваме с $min\,T(f)$ множеството от всички МИВ, а с $max\,F(f)$ – множеството от всички МНВ на дадена функция f. От Дефиниция 5.1 пряко следва, че *когато f е монотонна функция, тя се определя еднозначно чрез едното от множествата си $min\,T(f)$ или $max\,F(f)$.*

Отбелязваме, че в литературата на руски и на български език има известни разминавания между понятията, аналогични на истинен вектор, неистинен вектор, МИВ и МНВ. Понятията "горна нула" в [6, 18] и "максимална горна нула" в [11] означават едно и също – МНВ. В [6, 18] понятието "долна единица" е аналог на МИВ, а в [16] съответното понятие

е "горна единица". За да избегнем терминологични недоразумения ще се придържаме към понятията от Дефиниции 5.2 и 5.3. Напомняме, че под *вектор на булева функция разбираме нейния вектор с функционални стойности*, независимо дали е записан вертикално или хоризонтално.

Дефиниция 5.4

1) Елементарната конюнкция $K = x_{i_1}^{\sigma_1} \dots x_{i_k}^{\sigma_k}$ наричаме *импликанта* на функцията $f \in \mathcal{F}_2$, ако $T(K) \subseteq T(f)$. Импликантата K на f се нарича *проста*, ако не съществува друга импликанта K' на f, такава че $T(K) \subset T(K')$.

2) Клаузата (елементарната дизюнкция) $D = x_{j_1}^{\tau_1} \vee \dots \vee x_{j_r}^{\tau_r}$ наричаме *импликата* на функцията $g \in \mathcal{F}_2$, ако $F(D) \subseteq F(g)$. Импликатата D на g се нарича *проста*, ако не съществува друга импликата D' на g, такава че $F(D) \subset F(D')$.

Пример 5.5 Да разгледаме функцията $f(x,y,z) = (0,0,1,1,0,1,1,1)$. Както в Пример 1.32, построяваме нейните СъвДНФ и СъвКНФ.

1) СъвДНФ: $f(x,y,z) = \bar{x}y\bar{z} \vee \bar{x}yz \vee x\bar{y}z \vee xy\bar{z} \vee xyz$. Правилото за слепване (вж. раздел 1.4, в края), приложено към първите две пълни елементарни конюнкции (те са импликанти на f) дава импликантата $K = \bar{x}y$. Очевидно $T(\bar{x}y\bar{z}) \subset T(K)$ и $T(\bar{x}yz) \subset T(K)$, т. е. K поглъща (вж. пак раздел 1.4, в края) всяка от първите две импликанти. Следователно те не са прости.

2) СъвКНФ: $f(x,y,z) = (x \vee y \vee z)(x \vee y \vee \bar{z})(\bar{x} \vee y \vee z)$. Очевидно е, че например клаузата $D = x \vee y$ е също импликата на f. За нея имаме $F(x \vee y \vee z) \subset F(D)$ и затова импликатата $x \vee y \vee z$ не е проста.

Дефиниция 5.6 *Сложност* на формулата φ над $F \subseteq \mathcal{F}_2$ наричаме броя на срещанията на букви на променливи във φ. КНФ на $f \in \mathcal{F}_2$, чиято сложност е минимална спрямо сложностите на всевъзможните КНФ на f, наричаме *минимална* КНФ (МКНФ) на f. Аналогично, ДНФ на f, чиято сложност е минимална спрямо сложностите на всевъзможните ДНФ на f, наричаме *минимална* ДНФ (МДНФ) на f.

Няма да се спираме на методите за получаване на МДНФ и МКНФ на дадена функция $f \in \mathcal{F}_2^n$ – за справка те могат да бъдат намерени в учебниците [16, 9, 20]. Пак там е доказана следната

Теорема 5.7 *МДНФ на $f \in \mathcal{F}_2$ се състои само от прости импликанти.*

Аналогично твърдение е в сила и за МКНФ.

Теорема 5.8 *МКНФ на $f \in \mathcal{F}_2$ се състои само от прости импликати.*

Теоремата може да бъде доказана лесно чрез допускане на противното, доказателството ѝ е аналогично на това на Теорема 5.7.

Известно е, че ако f е МБФ, тя има единствена МДНФ и единствена МКНФ, в които всички букви на променливи са без отрицания [32, 33, 50]. С други думи, МДНФ и МКНФ на произволна $f \in M$ са суперпозиции над множеството $\{xy, x \vee y, \tilde{0}, \tilde{1}\}$. Отново там, както и в [16] (Лема 5.6.1), е показано съществуването на биекция между множеството от всички прости импликанти на $f \in M$ и множеството $minT(f)$, т. е.

Теорема 5.9 *На всяка проста импликанта $K_i = x_{i_1}x_{i_2}\ldots x_{i_k}$ в МДНФ на функцията $f \in M^n$ биективно съответства вектора $\alpha \in min\,T(f)$, който има единици в позициите $i_1, i_2, \ldots, i_k$ и нули във всички останали позиции (т. е. α е характеристичен вектор на K_i).*

Аналогично, за множеството от простите импликати на $f \in M$ и множеството $max\,F(f)$ е в сила [54, 50, 60]:

Теорема 5.10 *На всяка проста импликата $D_j = x_{j_1} \vee x_{j_2} \vee \ldots \vee x_{j_r}$ в МКНФ на функцията $f \in M^n$ биективно съответства вектора $\beta \in max\,F(f)$, който има нули в позициите $j_1, j_2, \ldots, j_r$ и единици във всички останали позиции (т. е. β е антихарактеристичен вектор на D_j).*

Пример 5.11 Да вземем отново функцията $f(x,y,z) = (0,0,1,1,0,1,1,1)$ от Пример 5.5. Отбелязваме, че f е монотонна и за нея имаме:

1) МДНФ на f е $f(x,y,z) = y \vee xz$, а y и xz са нейните прости импликанти. Биективните еквиваленти на y и xz са съответно векторите $(0,1,0)$ и $(1,0,1)$. Тогава $min\,T(f) = \{(0,1,0),(1,0,1)\}$.

2) МКНФ на f е $f(x,y,z) = (x \vee y)(y \vee z)$, а $(x \vee y)$ и $(y \vee z)$ са простите импликати на f. Биективните еквиваленти на $(x \vee y)$ и $(y \vee z)$ са съответно векторите $(0,0,1)$ и $(1,0,0)$. Тогава $max\,F(f) = \{(0,0,1),(1,0,0)\}$.

5.2 Една матрична структура и нейните свойства

В раздел 1.2 дефинирахме релацията "$\preceq$" над $\{0,1\}^n \times \{0,1\}^n$ и казахме, че тя е рефлексивна, антисиметрична и транзитивна, и че спрямо нея n-мерният булев куб е частично наредено множество (ЧНМ). Ще представим матрично предхожданията между двойките вектори на $\{0,1\}^n$.

Дефиниция 5.12 Дефинираме *матрица на предхожданията* $P_n = ||p_{i,j}||$ с размерност $2^n \times 2^n$ както следва: за всяка двойка вектори $\alpha, \beta \in \{0,1\}^n$, такива че $\#(\alpha) = i$, $\#(\beta) = j$, полагаме $p_{i,j} = 1$, ако $\alpha \preceq \beta$ или $p_{i,j} = 0$ в противен случай.

Понеже релацията "$\preceq$" е рефлексивна и антисиметрична и понеже векторите на $\{0,1\}^n$ са подредени лексикографски, следва че P_n е триъгълна матрица с единици по главния си диагонал и нули под него. Според дефиницията, редовете и стълбовете на P_n имат номера от 0 до $2^n - 1$, в съответствие с номерата на векторите от $\{0,1\}^n$.

Теорема 5.13 *За $n = 1$ матрицата $P_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. За всяко цяло число $n > 1$ матрицата P_n е блокова матрица от вида*

$$P_n = \begin{pmatrix} P_{n-1} & P_{n-1} \\ O_{n-1} & P_{n-1} \end{pmatrix},$$

където P_{n-1} означава същата матрица с размерност $2^{n-1} \times 2^{n-1}$, а O_{n-1} е нулевата матрица с размерност $2^{n-1} \times 2^{n-1}$.

Доказателство. Ще докажем теоремата чрез индукция по n.

1) Очевидно, за $n = 1$ матрицата P_1 представя предхожданията между векторите (0) и (1) в едномерния булев куб.

2) Да допуснем, че твърдението е вярно за матрицата P_{n-1}, представяща прехожданията на векторите от $\{0,1\}^{n-1}$ съгласно Дефиниция 5.12.

3) Съгласно Дефиниция 1.13, разглеждаме векторите на $\{0,1\}^n$ като получени от векторите на $\{0,1\}^{n-1}$ – веднъж чрез добавяне на 0 и втори път чрез добавяне на 1 в началото на всеки от тях, т. е. $\forall\ \alpha \in \{0,1\}^n \Rightarrow \alpha = (0,\gamma)$ или $\alpha = (1,\gamma)$, $\gamma \in \{0,1\}^{n-1}$. За произволни вектори $\alpha, \beta \in \{0,1\}^n$, според това дали започват с 0 или с 1, разглеждаме четирите възможни случая:

а) $\alpha = (0,\gamma), \beta = (0,\delta)$, където $\gamma, \delta \in \{0,1\}^{n-1}$. Нека $\#(\alpha) = i$, $\#(\beta) = j$. Тогава $i = \#(\gamma)$, $j = \#(\delta)$, $0 \leq i, j \leq 2^{n-1} - 1$ и още: $\alpha \preceq \beta \iff \gamma \preceq \delta$. За $0 \leq i, j \leq 2^{n-1} - 1$ елементите $p_{i,j}$ на матрицата P_n имат стойности, които (съгласно индуктивното предположение) съвпадат със стойностите на елементите със същите индекси от матрицата P_{n-1}. Следователно P_{n-1} е разположена в горния ляв блок на P_n.

б) $\alpha = (0,\gamma), \beta = (1,\delta)$, където $\gamma, \delta \in \{0,1\}^{n-1}$. Аналогично, нека $\#(\alpha) = i$, $\#(\beta) = j$. Тогава $i = \#(\gamma)$, $0 \leq i \leq 2^{n-1} - 1$ и $j = 2^{n-1} + \#(\delta)$, $2^{n-1} \leq j \leq 2^n - 1$. Отново $\alpha \preceq \beta \iff \gamma \preceq \delta$. За тези стойности на i и j елементите $p_{i,j}$ на матрицата P_n имат същите стойности, както елементите $p_{i,k}$, $k = j - 2^{n-1}$, на матрицата P_{n-1}, съгласно индуктивното предположение. Следователно P_{n-1} е разположена в горния десен блок на P_n.

в) $\alpha = (1,\gamma), \beta = (0,\delta)$, където $\gamma, \delta \in \{0,1\}^{n-1}$. Нито един вектор, който започва с 1, не предхожда вектор, започващ с 0. За номерата на векторите имаме: $2^{n-1} \leq \#(\alpha) \leq 2^n - 1$, $0 \leq \#(\beta) \leq 2^{n-1} - 1$ и следователно в долния ляв блок на P_n е разположена нулевата матрица O_{n-1}.

г) $\alpha = (1,\gamma), \beta = (1,\delta)$, където $\gamma, \delta \in \{0,1\}^{n-1}$. Случаят е аналогичен на случая а), с тази разлика, че номерата на векторите i и j са в друг диапазон: $2^{n-1} \leq i, j \leq 2^n - 1$. Аналогично получаваме, че матрицата P_{n-1} е разположена в долния десен блок на P_n.

С това теоремата е доказана. $\diamond$

Бележка 5.14 *Триъгълникът от числа, разположен върху и над главния диагонал на матрицата P_n е свързан с други познати структури:*

а) той се явява дискретен аналог на фракталната структура, известна като "Триъгълник на Серпински" (тази бележка дължим на доц. д-р Красимир Манев);

б) той съвпада с триъгълника на Паскал с 2^n реда, в който числата са взети по модул 2, т. е. над $GF(2)$ (това лесно се проверява по индукция).

Означаваме с $R_n = \{r_0, \ldots, r_{2^n-1}\}$ множеството от всички редове на P_n, разглеждани като двоични вектори.

Теорема 5.15 *Нека $\alpha = (a_1, a_2, \ldots, a_n) \in \{0,1\}^n$, $\#(\alpha) = i$, $1 \leq i \leq 2^n - 1$ и α съдържа единици в позициите $(i_1, i_2, \ldots, i_r)$, $1 \leq r \leq n$. Тогава елементарната конюнкция $c_i = x_{i_1} x_{i_2} \ldots x_{i_r}$ (т. е. с характеристичен вектор α) е монотонна функция, чиито вектор съвпада с i-тия ред r_i на матрицата P_n. Ако $\#(\alpha) = 0$, нулевият ред r_0 на P_n съответства на $\tilde{1}$.*

Доказателство. Първо отбелязваме, че досега номерирахме елементите на векторите от $\{0,1\}^n$ отляво надясно и означавахме с $x_1, x_2, \ldots, x_n$ променливите, съответстващи на тези елементи. Според Дефиниция 1.13, първо добавяме нули, после добавяме единици в началото на всеки вектор от $\{0,1\}^{n-1}$ и така получаваме векторите от $\{0,1\}^n$. Това означава, че добавяме в началото променливата x_1, а останалите променливи преномераме, увеличавайки индекса на всяка от тях с единица.

Това, че нулевият ред на P_n съответства на $\tilde{1}$, следва от Дефиниция 5.12 и Теорема 5.13. Елементарните конюнкции от твърдението на теоремата не съдържат отрицания и следователно са монотонни функции. Останалата част от твърдението ще докажем чрез индукция по n.

1) За $n = 1$, т. е. за матрицата P_1, твърдението очевидно е вярно.

2) Да допуснем, че твърдението е в сила за матрицата P_{n-1}, т. е., че за всяко i, $1 \leq i \leq 2^{n-1}-1$, елементарната конюнкция c_i (с характеристичен вектор $\alpha \in \{0,1\}^{n-1}$, $\#(\alpha) = i$), има вектор, който съвпада с i-тия ред r_i на матрицата P_{n-1}.

3) Нека $\alpha \in \{0,1\}^{n-1}$, $\#(\alpha) = i$, $1 \leq i \leq 2^{n-1}-1$ и α съдържа единици в позиции $i_1, i_2, \ldots, i_m$, $1 \leq m \leq n-1$. Съгласно индуктивното предположение, редът r_i на P_{n-1} съвпада (като вектор) с вектора на функцията $c_i = x_{i_1} x_{i_2} \ldots x_{i_m}$. Разглеждаме матрицата P_n.

а) Нека $\beta \in \{0,1\}^n$ е векторът, получен от α чрез добавяне на 0 в началото му. Тогава $\#(\beta) = i$, β съдържа единици в позициите с номера $i_1+1, i_2+1, \ldots, i_m+1$ и β е характеристичен вектор на елементарната конюнкция $c'_i = x_{i_1+1} x_{i_2+1} \ldots x_{i_m+1}$. Съгласно Теорема 5.13, редът r'_i на P_n е получен от реда r_i на матрицата P_{n-1}, записан два пъти един след друг. Тогава r'_i съвпада с вектора на булева функция на n променливи, получена чрез добавяне на фиктивната променлива x_1 към функцията c_i. Следователно редът r'_i на P_n съдържа функционалните стойности на елементарната конюнкция c'_i.

б) Нека $\beta \in \{0,1\}^n$ е векторът, получен от α чрез добавяне на 1 в началото му. Тогава $\#(\beta) = 2^{n-1} + i = k$, β съдържа единици в позициите с номера $1, i_1+1, \ldots, i_m+1$ и β е характеристичен вектор на елементарната конюнкция $c_k = x_1 x_{i_1+1} \ldots x_{i_m+1}$. Съгласно Теорема 5.13, редът r_k на P_n е образуван от ред на нулевата матрица O_{n-1} (т. е. започва с 2^{n-1} на брой нули), след който е записан редът r_i на матрицата P_{n-1}. Разглеждаме r_k като вектор на булева функция на n променливи, получена чрез добавяне на съществената променлива x_1 – в конюнкция с функция на $n-1$ променливи. Върху първата половина от векторите на $\{0,1\}^n$ имаме $x_1 = 0$ и тогава първата половина от стойностите на r_k са нули. Върху втората половина от векторите на $\{0,1\}^n$ имаме $x_1 = 1$ и тогава втората половина от стойностите на r_k съвпадат съответно със стойностите на r_i. Следователно в реда r_k на P_n са записани функционалните стойности на елементарната конюнкция c_k.

С това теоремата е доказана. $\diamond$

Означаваме с C_n множеството от всички елементарни конюнкции на n променливи, несъдържащи отрицания.

Бележка 5.16 *Теорема 5.15 установява биекциите:* $\varphi : \{0,1\}^n \to C_n$ *(биекцията между вектор* $\alpha \in \{0,1\}^n$, $\#(\alpha) = i$ *и елементарната конюнкция* c_i*) и* $\psi : C_n \to R_n$ *(биекцията между представянето чрез формула* c_i *и представянето чрез вектор* r_i *на всяка елементарна конюнкция). Дизюнкцията и конюнкцията над двоични вектори разглеждаме съответно като побитова дизюнкция и конюнкция. Векторът на произволна* $f \in M^n$ *може да бъде представен като линейна комбинация* $f(x_1, x_2, \ldots, x_n) = a_0 r_0 \vee a_1 r_1 \vee \ldots \vee a_{2^n-1} r_{2^n-1}$, *в която коефициентите* $a_0, a_1, \ldots, a_{2^n-1} \in \{0,1\}$, *а тривиалната комбинация съответства на* $\tilde{0}$. *Когато* $f(x_1, x_2, \ldots, x_n) = r_{i_1} \vee r_{i_2} \vee \ldots \vee r_{i_k}$ *е МДНФ на* f, *тогава* r_{i_j} *и* $r_{i_l}, 1 \leq j < l \leq k$, *са несравними помежду си, а съответните им елементарни конюнкции* $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ *са прости импликанти на* f.

Следващата таблица илюстрира твърдението на Теорема 5.15 за $n = 3$.

$\alpha = (x_1, x_2, x_3)$	$i = \#(\alpha)$	P_3	c_i
(0 0 0)	0	1 1 1 1 1 1 1 1	$\tilde{1}$
(0 0 1)	1	0 1 0 1 0 1 0 1	x_3
(0 1 0)	2	0 0 1 1 0 0 1 1	x_2
(0 1 1)	3	0 0 0 1 0 0 0 1	x_2x_3
(1 0 0)	4	0 0 0 0 1 1 1 1	x_1
(1 0 1)	5	0 0 0 0 0 1 0 1	x_1x_3
(1 1 0)	6	0 0 0 0 0 0 1 1	x_1x_2
(1 1 1)	7	0 0 0 0 0 0 0 1	$x_1x_2x_3$

Таблица 5.1: Илюстрация на твърдението на Теорема 5.15 за $n = 3$.

Да разгледаме произволен ред r_i на матрицата P_n и нека той съдържа единици в позициите $i_1, i_2, \ldots, i_k$ $(i_1 = i < i_2 < \ldots < i_k)$. Тогава множеството от вектори $\{\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_k}\} \subseteq \{0,1\}^n$ е всъщност множеството $T(c_i)$. Векторът $\alpha_{i_1} = \alpha_i$ предхожда останалите вектори от $T(c_i)$ и следователно $min\, T(c_i) = \{\alpha_i\}$. Нека j е номерът на позицията на най-дясната нула в реда r_i, т. е. след нея са само единици. Да разгледаме стълба s_j на P_n. Нека той съдържа единици в позициите $j_1, j_2, \ldots, j_m$ $(j_1 < j_2 < \ldots < j_m = j)$. Това означава, че всеки вектор от множеството $\{\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_m}\} \subseteq \{0,1\}^n$ предхожда вектора $\alpha_{j_m} = \alpha_j$. Нулата в позиция j на r_i означава нули и в позициите $j_1, j_2, \ldots, j_m$, т. е. $\{\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_m}\} \subseteq F(c_i)$ и $\{\alpha_j\} \subseteq max\, F(c_i)$. Съгласно Теорема 5.10, на вектора α_j биективно съответства клауза d_j, на която α_j е антихарактеристичен вектор.

Пример 5.17 Да вземем реда r_3 на матрицата P_3 (вж. Таблица 5.1). Той съдържа единици в позициите с номера 3 и 7, на r_3 съответства елементарната конюнкция $c_3 = x_2x_3$. Следователно $T(c_3) = \{(0,1,1), (1,1,1)\}$ и $min\, T(c_3) = \{(0,1,1)\}$. Най-дясната нула в r_3 е в позиция номер 6. Стълбът с номер 6 съдържа единици в позициите $\{0,2,4,6\}$. Следователно $\{(0,0,0), (0,1,0), (1,0,0), (1,1,0)\} \subset F(c_3)$ и $(1,1,0) \in max\, F(c_3)$. Всъщност $max\, F(c_3) = \{(1,0,1), (1,1,0)\}$, а на $\alpha_6 = (1,1,0)$ съответства клаузата $d_6 = x_3$.

5.3 Генериране на монотонни булеви функции

Най-старата задача в областта на МБФ, формулирана още в края на XIX век, е *задачата на Дедекинд за определяне на броя на МБФ на n променливи* (или още за определяне броя на антиверигите в ЧНМ) [69]. Многобройните усилия и търсения доведоха до получаване на редица оценки на $|M^n|$ отгоре и отдолу [14, 70], но все още не е известна формула за определяне на $|M^n|$ в общия случай. Една популярна оценка на $|M^n|$ отдолу е дадена в [16, 9, 20]:

Теорема 5.18 $|M^n| > 2^{\binom{n}{\lfloor n/2 \rfloor}}$

Доказателство. Отбелязваме, че при фиксирано n, максималната стойност на биномния коефициент $\binom{n}{k}$ се достига при $k = \lfloor n/2 \rfloor$. Броят на векторите от $\{0,1\}^n$ с тегло $k = \lfloor n/2 \rfloor$ е $\binom{n}{k}$. Определяме множеството M' от всички функции f на n променливи, за които:

$$f(\alpha) = \begin{cases} 0, & \text{ако } wt(\alpha) < k \\ \text{произволна}, & \text{ако } wt(\alpha) = k \\ 1, & \text{ако } wt(\alpha) > k \end{cases}$$

Очевидно, всяка функция от M' е монотонна и затова $|M^n| > |M'| = 2^{\binom{n}{\lfloor n/2 \rfloor}}$.

До скоро броят на МБФ на n променливи беше известен само за $0 \leq n \leq 6$. През 1999 г. Владета Йовович и др. успяха да получат стойностите на $|M^n|$ за $n = 7$ и $n = 8$ [67, 69, 71]. За целта те разбиват множеството от всички МБФ на подмножества според броя на минималните долни единици, т. е. според $|minT(f)| = k$ за $f \in M$. Като означават $M(n,k) = |minT(f)|, \forall f \in M^n$, те успяват да изведат формули за $M(n,k)$ за $0 \leq k \leq 10$ и чрез тях да получат $|M^n|$ за $n = 7, 8$. Известният досега брой на МБФ на n променливи ($0 \leq n \leq 8$) е даден в следващата таблица (вж. [69] и редица A000372 в [72]).

n	$\|M^n\|$	n	$\|M^n\|$	n	$\|M^n\|$
0	2	3	20	6	7 828 354
1	3	4	168	7	2 414 682 040 998
2	6	5	7 581	8	56 130 437 228 687 557 907 788

Таблица 5.2: $|M^n|$ **за** $0 \leq n \leq 8$

Една от важните задачи в комбинаториката е *задачата за генериране елементите на дадено множество в определен ред.* По отношение на МБФ тази задача има и допълнително значение – като подход ("генериране и броене") за частично решаване задачата на Дедекинд. В [68] е даден алгоритъм (реализиран на $C++$) за генериране на МБФ, който се използва за тестване на сортировка на мрежи, както и за решаване задачата на Дедекинд за $0 \leq n \leq 7$. Алгоритъмът е описан в [70] и се основава на факта, че ако $f, g \in M^{n-1}$ (f и g са представени чрез своите вектори) и ако $f \preceq g$ (т. е. $f \vee g = g$), тогава конкатенацията fg на съответните им два вектора (разглеждани като думи), дава вектора на функцията $h = fg$, $h \in M^n$.

Това означава, че за генериране на функциите от M^n е необходимо: (1) да бъдат получени и съхранени всички функции от M^{n-1} и (2) да бъдат извършвани проверки дали $f \vee g = g$ за всевъзможните двойки $f, g \in M^{n-1}$. Тези две характеристики намаляват ефективността на този алгоритъм, но това се компенсира частично чрез използване на редица програмистки техники.

В този раздел предлагаме друг алгоритъм, наречен условно *Gen*, за генериране на МБФ на n променливи, $0 \leq n \leq 7$. Първоначалното му предназначение беше определянето на $|M^7|$, но подходът "генериране и броене" в [2] се оказа недостатъчно мощен спрямо споменатия по-горе подход на Вл. Йововић и др. Gen генерира функциите от M^n при дадено n, $1 \leq n \leq 7$, като вектори, получавани в лексикографски ред. Той се основава на матрицата P_n в смисъла на Бележка 5.16 и поясненията след нея, а именно: ако i-тия ред r_i на P_n съдържа нула в позицията с номер j, $i < j < 2^n - 1$, то редовете r_i и r_j са несравними и тогава $f = r_i \vee r_j = c_i \vee c_j$, $f \in M^n$. Ако векторът на f съдържа нула в позицията с номер k, $i < k < 2^n - 1$, този вектор и r_k са несравними и тогава полагаме $f = r_i \vee r_j \vee r_k = c_i \vee c_j \vee c_k$, $f \in M^n$ и т. н. Нека последователно избираме редовете на P_n от последния към нулевия и полагаме $f = r_i$. Докато векторът на f съдържа нули в позициите след i-та изпълняваме следното: определяме j – позицията на най-дясната нула във вектора на f, извършваме побитово дизюнкцията $f \vee r_j$, присвояваме резултата на f и го извеждаме. Така получаваме векторите на всички функции от M^n, подредени лексикографски, или по-формално:

Алгоритъм Gen: генериране на МБФ на n променливи в лексикографски ред.
Вход: n.
Изход: векторите на функциите от M^n в лексикографски ред.
Процедура:
1) Полагаме $f = \tilde{0}$. Извеждаме f.
2) За всеки ред r_i, $i = 2^n - 1, 2^n - 2, \ldots, 0$, полагаме $f = r_i$ и:
   а) извеждаме f;
   б) за всяка позиция j, $j = 2^n-2, 2^n-3, \ldots, i+1$ проверяваме дали $f[j] = 0$.
Ако "Да", рекурсивно полагаме $f = f \vee r_j$ и преминаваме на т. а).
3) Край.

Фрагмент от реализация на Gen, написан на Паскал, изглежда така:

```
1)  Program Gen;
2)  .....
3)  Procedure Generate (G: BoolFun; i : byte);
4)    var  j : byte;
5)    begin
6)      for j:= i to dim do     { Disjunction between the i-th row }
7)        if P[i,j]=1 then  G[j]:= 1; { and the current function G.}
8)      Output (G);
```

```
9)      for j:= dim-1 downto i+1 do   { Searching zero for the next }
10)       if G[j]=0 then  Generate (G, j);          { disjunction. }
11)   end;   { Generate }

12) Begin   { Main }
13)   ...
14)   readln (n);              { Number of variables }
15)   dim:= 1 shl n - 1;       { dim:= 2^n-1 }
16)   Fill_Array;              { Filling in the matrix P_n }
17)   FillChar (F, dim, 0);    { The constant zero - separately }
18)   Output (F);
19)   for k:= dim downto 0 do  Generate (F, k);
20) End.
```

Коментари към алгоритъма и реализацията му.

1) Процедурата Fill_Array (в ред 16) генерира и съхранява в паметта матрицата P_n – съгласно Бележка 5.14 б) или Теорема 5.13. При $n > 7$ матрицата става твърде голяма – тогава може да се използва побитово представяне на елементите ѝ или по-мощна програмна среда.

2) Gen наистина генерира функциите в лексикографски ред, понеже циклите в редове 9 и 19 са с намаляваща стъпка.

3) Gen може да генерира и само тези функции, които лексикографски следват дадена функция f – за целта, в ред 17, инициализираме $F = f$ вместо $F = \tilde{0}$. Също така цикълът в ред 19 трябва да започва от i – позицията на най-лявата единица във вектора на f. Ако се промени крайната стойност в същия цикъл (от нула на m, $0 < m < dim$, например), Gen ще генерира само тези функции, които предхождат лексикографски функцията с вектор r_{m+1}.

4) Дълбочината на рекурсията в Gen се определя от максималния брой независими редове от P_n, участващи в дизюнкции, т. е. от функцията с максимален брой минимални истинни вектори. Според [32, 33] и в съответствие с Теорема 5.18, това е числото $\max\limits_{f \in M^n} |min\,T(f)| = \binom{n}{\lfloor n/2 \rfloor}$, равно на броя на елементите в максималната антиверига в ЧНМ $\{0,1\}^n$.

5) Gen е типичен представител на експоненциалните алгоритми – поради естеството на самата задача, големината на изхода винаги е експоненциална спрямо големината на входа. Както споменахме в раздел 2.5, сложностите на масови задачи от този тип имат отделна класификация [52]. Gen генерира поредната функция като използва само последната генерирана функция и определен ред от матрицата P_n. Той работи за *нарастващо полиномиално време* [52], понеже поредната функция се генерира за време, което е полином спрямо общата големина на входа, на последната генерирана функция и на определен ред от P_n. Това означава още, че алгоритъмът работи за *полиномиално общо време* [52], т. е. времето за изпълнението му е полином спрямо общата големина на входа и на изхода.

6) Gen може лесно да бъде модифициран, така че:

а) да извежда и простите импликанти на поредната генерирана функция;

б) да генерира всички антивериги в дадено ЧНМ в определен ред.

Бяха направени редица опити за оптимизиране на алгоритъма Gen, касаещи най-вече структурите от данни и представянето им, реализации на *Watcom C* и $C++$. Неговото бързодействие не се подобри значително – на компютър с 300 мегахерцов процесор Gen генерира всички МБФ на 6 променливи за около 6 секунди. При опитите за генериране и преброяване на МБФ на 7 променливи, задачата беше разбита на независими подзадачи, които се изпълняваха в продължение на около 170 часа на няколко (по-слаби) компютъра и така бяха получени около 7.35% от тези МБФ. Асимптотичните оценки на Коршунов [14, 70] показаха, че за решаването на цялата задача ще са необходими около 100 денонощия. След като научихме за резултатите на Йовович и др., опитите бяха прекратени.

5.4 Определяне на минимален истинен и на максимален неистинен вектор на монотонни функции

Задачата за определяне на поне един МИВ и/или поне един МНВ на неизвестна монотонна функция е една от важните задачи в областта на МБФ [6, 11, 13]. Тя е тясно свързана с друга важна задача – за идентифициране (или разпознаване, разшифроване) на неизвестна МБФ, която се разглежда в следващия раздел. Всъщност, определянето на всички МИВ (или МНВ), т. е. на множеството $minT(f)$ (или $maxF(f)$), на практика означава идентифициране на неизвестната МБФ.

В рускоезичната литература задачата за определяне на МИВ и/или МНВ се разглежда при предположението, че произволна функция $f \in M^n$ е зададена чрез някакъв оператор A_f, който за произволен вектор $\alpha \in \{0,1\}^n$ връща стойността на $f(\alpha)$ [6, 11, 18]. В англоезичната литература същата задача се изследва в контекста на теорията на изчислителното изучаване, чиято цел е да определя и изследва полезни модели на явления, свързани с изучаване, от алгоритмична гледна точка [24]. По-конкретно, за неизвестна МБФ изучаващият алгоритъм може да задава два вида въпроси към хипотетична оракулна машина:

– въпроси за членство (membership queries) – например, дали избран вектор α е истинен вектор за f. Машината отговаря с "да" или "не";

– въпроси за еквивалентност (equivalence queries) – например, дали неизвестната функция f съвпада с дадена функция–хипотеза g. Машината отговаря с "да" ако съвпадат, или връща контрапример – произволен вектор, за който двете функции приемат различни стойности.

Задачата за определяне на поне един МИВ и/или поне един МНВ на неизвестна $f \in M^n$ се решава алгоритмично. Търсят се ефективни алгоритми, които да определят съответния вектор (вектори) с минимален брой въпроси за членство (или, еквивалентно, обръщения към оператора A_f) и да имат минимална времева сложност.

В [11] се разглежда задачата за определяне на поне един МНВ при монотонни функции в k-значната логика (МБФ са техен частен случай при $k = 2$). За произволен алгоритъм, който определя един МНВ на $f \in M^n$, е изведена оценка, според която алгоритъмът изисква $O\left(\binom{n}{\lfloor n/2 \rfloor}\right)$ обръщения към оператора A_f. В [6] Гайнанов представя алгоритъм, който за произволна $f \in M^n$ определя вектор α, принадлежащ или на $min\,T(f)$, или на $max\,F(f)$, като използва $O(n)$ обръщения към оператора A_f и има времева сложност $O(n)$. След първото обръщение към оператора A_f (но не и преди това) става ясно какъв вектор ще бъде намерен – МИВ или МНВ. Този алгоритъм е сред най-ефективните досега и затова е особено популярен и често използван в алгоритми за решаване на задачата за идентифициране на неизвестна МБФ [32, 33, 55].

Алгоритъмът на Гайнанов работи както следва. Нека $\alpha \in \{0,1\}^n$ е вектор, който съдържа единици в позиции $j_1, j_2, \ldots, j_k$ и за него A_f връща $f(\alpha) = 1$. Нека $e_i \in \{0,1\}^n$ означава i-тия единичен вектор (само i-та му компонента е единица, а всички останали са нули). Алгоритъмът построява редицата $f(\alpha^i)$, $i = 1, 2, \ldots, k$, за която:

$\alpha^0 = \alpha$,

$\alpha^i = \alpha^0 \oplus f(\alpha^1)e_{j_1} \oplus \ldots \oplus f(\alpha^{i-1})e_{j_{i-1}} \oplus e_{j_i},\ i = 1, 2, \ldots, k.$

Нека p е максималният индекс, за който $f(\alpha^p) = 1$. Тогава векторът $\beta = \alpha^p$ е нов вектор за множеството $min\,T(f)$. Случаят $f(\alpha) = 0$ се третира аналогично – разглеждат се позициите $l_1, l_2, \ldots, l_r$ на нулите в α и алгоритъмът построява редицата $f(\alpha^i)$, $i = 1, 2, \ldots, r$, за която:

$\alpha^0 = \alpha$,

$\alpha^i = \alpha^0 \oplus \bar{f}(\alpha^1)e_{l_1} \oplus \ldots \oplus \bar{f}(\alpha^{i-1})e_{l_{i-1}} \oplus e_{l_i},\ i = 1, 2, \ldots, r.$

Ако q е максималният индекс, за който $f(\alpha^q) = 0$, тогава векторът $\beta = \alpha^q$ е нов вектор за множеството $max\,F(f)$.

Дефиниция 5.19 Нека $f \in M^n$. Векторът $\alpha \in min\,T(f)$ наричаме *лексикографски първи* МИВ (или кратко *първи МИВ*), ако α предхожда лексикографски всеки друг вектор $\beta \in min\,T(f)$. Векторът $\alpha \in max\,F(f)$ наричаме *лексикографски последен* МНВ (или кратко *последен МНВ*), ако всеки друг вектор $\beta \in max\,F(f)$ предхожда лексикографски α.

Предлагаме алгоритъм в две разновидности, наречени *Search_First* и *Search_Last* за определяне съответно на първи МИВ и последен МНВ на произволна $f \in M^n$. Алгоритъмът представлява обикновено двоично търсене и се основава на блоковата структура на матрицата P_n, дадена в Теорема 5.13, на твърдението в Теорема 5.15 и Бележка 5.16. Векторът на произволна $f \in M^n \backslash \{\tilde{0}\}$ представлява побитова дизюнкция на определени (несравними) редове от матрицата P_n. При Search_First задачата е да определим най-малкия номер на ред измежду тези, участващи в дизюнкцията. Неговият номер е всъщност номерът на първия МИВ на f. При Search_Last търсим позицията на нулевата компонента с най-голям номер във вектора на f – това е номерът на последния МНВ на f. В двата варианта на алгоритъма използваме променливите $left$ и $right$, означаващи съответно лявата и дясната граница на интервала за търсене.

Инициализираме ги: $left = 0$ и $right = 2^n - 1$. Във варианта Search_First задаваме въпроси за членство за векторите от $\{0,1\}^n$ с номера от вида $m = (left + right)\ div\ 2$, т. е. дали $f(\alpha_m) = 1$. Ако "да", полагаме $right = m$, иначе $left = m+1$. Пресмятаме отново m, пак задаваме въпрос за членство и т. н. докато $left < right$. Основание за това е фактът, че всички редове от горната половина на матрицата P_n съдържат единица в позицията с номер m, а всички редове от долната ѝ половина съдържат нула в тази позиция (това е в сила и за блоковете P_{n-1} и позициията с номер $2^{n-2} - 1$ и т. н. – вж. Таблица 5.1). Ето как изглежда кодът на Search_First, написан на Паскал.

```
Function Search_First (left, right : word) : word;
  var m : word;
  begin
    while left < right do
      begin
        m:= (left + right) div 2;
        if f[m] = 1          { treated as a membership query }
          then right:= m
          else left:= m+1;
      end;
    Search_First:= left;
  end;
```

Определянето на последния МНВ е аналогично, въпросите за членство са за векторите от $\{0,1\}^n$ с номера от вида $m = ((left + right)\ div\ 2) + 1$, т. е. дали $f(\alpha_m) = 0$. Ако "да", полагаме $left = m$, иначе $right = m - 1$ и т. н. Кодът на Search_Last, написан на Паскал, е аналогичен на предишния:

```
Function Search_Last (left, right : word) : word;
  var m : word;
  begin
    while left < right do
      begin
        m:= (left + right) div 2 + 1;
        if f[m] = 0          { treated as a membership query }
          then left:= m
          else right:= m-1;
      end;
    Search_First:= right;
  end;
```

Като коментар към алгоритъма в двете му разновидности ще отбележим:

1) Search_First определя първия МИВ, а Search_Last – последния МНВ на произволна $f \in M^n$ чрез точно n въпроса за членство и с времева сложност $\Theta(n)$. За определяне на несъществуващия първи МИВ при $\tilde{0}$ и на несъществуващия последен МНВ при $\tilde{1}$ е необходима малка модифика-

ция – например запомняне отговора на последния въпрос за членство и още една проверка в двете функции.

2) Предложеният алгоритъм извършва само операцията целочислено деление (за пресмятане на индекси) и не изпълнява никакви операции над векторите, за разлика от алгоритъма на Гайнанов.

3) Останалите вектори от множествата $minT(f)$ и $maxF(f)$ също могат да бъдат определяни чрез Search_First и Search_Last, но вече включени като елементи в един по-общ алгоритъм за идентифициране на f.

Следващата таблица илюстрира изпълнението на алгоритъма върху примерна функция $f \in M^4$, третирана като непозната. В третия ред с арабски цифри е показана поредността на проверяваните компоненти (чрез въпроси за членство) от функцията Search_First, а с римски цифри – тези от функцията Search_Last.

номера на компонентите	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(x_1, x_2, x_3, x_4)$	0	0	1	1	0	0	1	1	0	1	1	1	0	1	1	1
порденост на тестване		3	4	2				1	*I*				*II*	*IV*	*III*	

Таблица 5.3: Илюстрация на изпълнението на Search_First и Search_Last.

5.5 Идентифициране на монотонни булеви функции

За *задачата за идентифициране на неизвестна МБФ f на n променливи* вече стана дума в началото на предишния раздел. Идентифицирането на f означава изучаващият алгоритъм да определи множествата $minT(f)$ и/или $maxF(f)$ чрез въпроси за членство – в термините на теорията на изчислителното изучаване това е пример за "точно изучаване чрез въпроси за членство" [24, 33, 55]. По отношение на модела "точно изучаване чрез въпроси за членство и въпроси за еквивалентност", в [6, 24] са представени алгоритми, които определят $minT(f)$ на непозната $f \in M^n$, като използват общо $O(n|minT(f)|)$ въпроси за членство и за еквивалентност. Този модел няма да бъде обсъждан по-нататък.

В [33] се разглеждат редица детайли, характеристики и критерии за оценки на изучаващ алгоритъм с въпроси за членство. За един такъв алгоритъм са посочени аргументи, според които:

1) той трябва да определя и двете множества $minT(f)$ и $maxF(f)$, въпреки че едното може да бъде определено от другото, но за експоненциално (спрямо големините на двете множества) време в общия случай. По-късно в [50] е показано, че определянето на $maxF(f)$ от $minT(f)$ е еквивалентно с определянето на двойнствената функция $f^*(x_1, x_2, \ldots, x_n) = \bar{f}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ – тази задача е известна като "дуализация" или "трансверзала на хиперграф". Доказано е, че тази задача може да бъде решена за нарастващо квази-полиномиално време;

2) сложността му да се оценява спрямо общата големина на входа n и изхода $m = |minT(f)| + |maxF(f)|$, като отчита и времето за генерирането на векторите за въпросите. При това условие се оценяват броят на зада-

дени въпроси и сложността по време. Очевидно, полиномиална сложност по време имплицира полиномиален брой въпроси.

Отбелязваме, че според доказателството на Теорема 5.18, големината на m може да достигне $\binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lceil n/2 \rceil}$ и полиномиалност само спрямо n не може да се очаква [32, 33].

Съществуването на полиномиален (спрямо общата големина на входа и изхода) изучаващ алгоритъм чрез въпроси за членство, е еквивалентно на съществуването на полиномиални алгоритми за решаването на много други задачи в теория на хиперграфите, изкуствения интелект, булевата алгебра и др. [55]. Въпросът за сложността на тези задачи е все още отворен, въпреки че според Фридман и Хачиян [42] изглежда малко вероятно те да са $\mathcal{NP}$-трудни (задачата f наричаме *$\mathcal{NP}$-трудна*, ако за $\forall\ g \in \mathcal{NP},\ g \propto f$, но не е задължително $f \in \mathcal{NP}$, т. е. в сила е условието б), но не е задължително условието а) на Дефиниция 2.25).

Макино, Ибараки и др. изследват интензивно сложността на задачата за идентифициране [32, 33, 54, 55]. Сложността ѝ е тясно свързана със сложността на задачата за намиране на нов вектор за някое от множествата $MT \subseteq min\,T(f)$ или $MF \subseteq max\,F(f)$, които представят натрупаните до определен момент частични знания за функцията f, която се идентифицира. В [54, 55] се въвежда понятието "максимална латентност" като мярка на трудността за намиране на нов вектор. Показано е, че ако за неизвестна $f \in M^n$ максималната латентност е константа, тогава нов вектор може да бъде намерен за полиномиално време и тогава съществува алгоритъм за идентифициране, който работи за нарастващо полиномиално време (алгоритъмът в [55] използва време $O(n^2m)$ и задава $O(n^2m)$ въпроса). Това означава, че в тези случаи задачата за идентифициране е решима за полиномиално общо време. За редица подкласове на монотонните булеви функции е доказано, че те имат постоянна максимална латентност. На базата на тези резултати, в [60] е доказано, че почти всички МБФ са изучаеми чрез въпроси за членство за полиномиално общо време.

Предлагаме алгоритъм, наречен *Identify*, който идентифицира непозната МБФ и е принципно различен от останалите. Той отново се основава на свойствата на матрицата P_n, както и на алгоритмите за определяне на първи МИВ и на последен МНВ. Задачата за идентифициране можем да разглеждаме като обратна на задачата за генериране на МБФ в следния смисъл: ако при генерирането на МБФ се извършва дизюнкция на определени (несравними помежду си) редове от P_n, то при идентифициране на такава функция, считана за неизвестна, трябва да бъдат определени кои са тези редове. Трудността при определянето и отделянето на редовете произтича от това, че те имат съвпадащи компоненти.

Identify определя $min\,T(f)$ и $max\,F(f)$ на неизвестна $f \in M^n$ само чрез въпроси за членство. Най-общо идеята, заложена в него, е следната. Identify работи рекурсивно, като на всяка стъпка определя първия МИВ и последния МНВ на f, използвайки Search_First и Search_Last. След това той разделя f на две подфункции $g, h \in M^{n-1}$, такива че $g \preceq h$ и векторът на f се явява конкатенация на векторите (разглеждани като думи) на g

и h – както споменахме в раздел 5.3. Намереният първи МИВ на f е такъв и за g и се търси само последният МНВ на g; намереният последен МНВ на f е такъв и за h и се търси само първият МИВ на h. Всяка от функциите g и h се разделя на подфункции и идентифицира рекурсивно по същия начин, както f. Разделянето на подфункции продължава, докато на поредната стъпка се получи функция, за която позицията на първия МИВ е по-голяма от позицията на последния МНВ, т. е. функция с вектор от вида: $(0,\ldots,0)$, или $(0,\ldots,0,1,\ldots,1)$, или $(1,\ldots,1)$ (в частност може да се окаже вектор на функция на една променлива). При нея започва отделяне на простите импликанти, което се изпълнява лесно. Основната идея на алгоритъма, реализирана на Паскал, е представена в следващата процедура.

```
Procedure Id (left, right, l1, r0  : word);
  { left and right are the boundaries of the function.  }
  { l1 and r0 are the positions of the First MTV and    }
  { of the Last MFV in the function, correspondently.   }
  var m  : word;   { the median's position }
      p0,          { positions of the Last MFV and of  }
      p1 : word;   { the First MTV in the subfunctions }
  begin
    if l1 > r0 then Reg_Prime_Implicants (l1, right)
    else
      begin
        m:= (left+right) div 2;      {The split's position}
        p0:= Search_Last (left, m);
        Reg_Zeros (p0);
        Id (left, m, l1, p0);    {Ident. of the left subfunction}
        p1:= Search_First (m+1, right);
        Id (m+1, right, p1, r0); {Ident. of the right subfunction}
      end;
  end;   {Id}
```

Ще коментираме някои детайли от реализацията на целия алгоритъм:

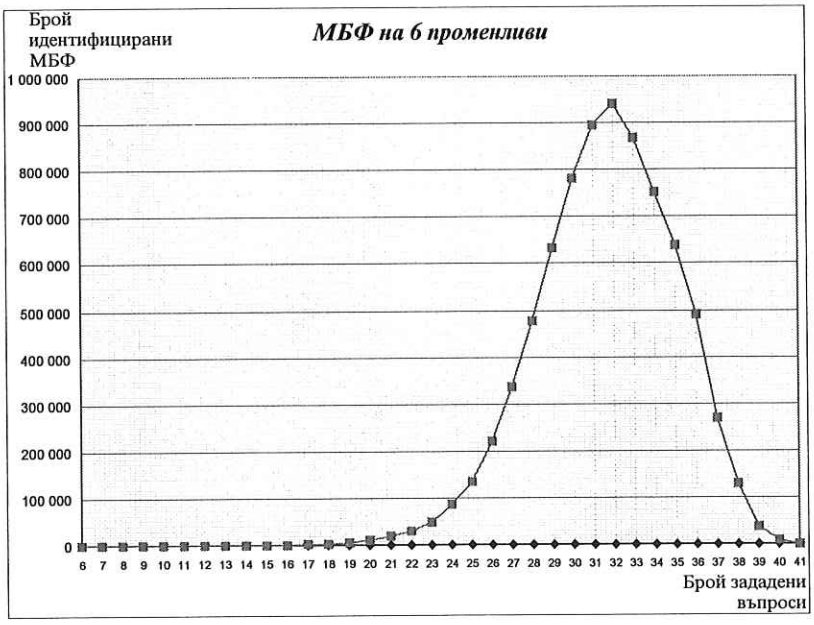
1) той използва функция-догадка h, първоначално всички компоненти в нейния вектор се инициализират като неопределени (например със стойност 2);

2) регистрирането на намерените прости импликанти става в процедурата Reg_Prime_Impl (l1, right). В цикъл от $l1$ до $right$ се проверяват съответните позиции във функцията-догадка h. Ако поредната позиция (например с номер k) е маркирана като неопределена, регистрираме простата импликанта с номер k и записваме единици във всички позиции на h, в които редът r_k на P_n съдържа единици;

3) маркирането на откритите нули във функцията h става чрез процедурата Reg_Zeros (p0). Във функцията h записваме нули във всички позиции, в които стълбът с номер $p0$ на P_n съдържа единици. Поради това, че процедурата Id идентифицира рекурсивно първо лявата подфункция,

а после дясната, намерените прости импликати на подфункциите не винаги са прости импликати на f. Затова определянето на самите прости импликати става в края на алгоритъма, след определянето на $minT(f)$. Отбелязваме, че h вече съдържа само нули и единици и съвпада с f. За да определим простите импликати, обхождаме всички позиции във функцията h от последната към първата. Ако поредната позиция (например с номер j) съдържа нула, тогава регистрираме простата импликата с номер j и записваме стойност 3 във всички позиции на h, в които стълбът с номер j на P_n съдържа единици.

Алгоритъмът Identify включва като елемент в себе си стратегиията "разделяй и владей" – при изпълнение на Search_First и Search_Last, но като цяло се подчинява на стратегията "динамично програмиране" – понеже подфункциите може да имат общи МИВ и МНВ, оригиналната задача притежава свойството "припокриване на подзадачи" и решението ѝ се конструира от решенията на подзадачи. Тези решения се регистират във функцията-догадка h, при това върху цялата h – поради монотонността на f. По този начин при рекурсивните обръщения се избягват въпроси, чиито отговори са вече известни от монотонността. Основен критерий при разработване на алгоритъма беше идентифицирането да бъде с минимален брой въпроси. Понеже определяните първи МИВ и последен МНВ на подфункциите не винаги са МИВ и МНВ на f, оценка на броя използвани въпроси не беше изведена. Резултатите от проведените експерименти показват, че за идентифициране на неизвестна $f \in M^n$, $1 \leq n \leq 6$, са нужни $O(nm)$ въпроса за членство. На Фигура 5.1 са представени графично резултатите от идентифицирането на всички МБФ на 6 променливи. Диаграмата показва колко функции чрез колко въпроса се идентифицират.



Фигура 5.1: Резултати от идентифицирането на всички МБФ на 6 променливи.

Други резултати относно максималния и средния брой въпроси, необходими за идентифициране на функциите от M^n, $1 \leq n \leq 6$, са представени в Таблица 5.4, където q_n означава съответния брой въпроси за n променливи.

n	Максимално q_n	Средно q_n	n	Максимално q_n	Средно q_n
1	2	1.66	4	12	8.95
2	3	2.66	5	22	16.94
3	6	4.70	6	41	31.23

Таблица 5.4: Брой въпроси, необходими за идентифициране на функциите от M^n, за $1 \leq n \leq 6$.

Времевата сложност на Identify е експоненциална и това се дължи на използването на функция-догадка. Предстоят опити за подобряване на тази сложност чрез използване на други начини на представяне на данните (например двоични диаграми на решенията) с цел отхвърляне на функцията-догадка. Възникнаха идеи за по-различен вариант на алгоритъма, реализирането на които е все още в начална фаза.

Заключение

Заключителните бележки, свързани с конкретните разгледани задачи, са дадени в края на съответните глави и раздели на дисертацията, където се посочват и някои възможни насоки за бъдещи изследвания, за усъвършенстване и приложения. Тук резюмираме получените резултати и публикациите по тях.

В раздел 3.2 са въведени еквивалентности над множеството C_n от всички КНФ на не повече от n променливи, в Леми 3.8 – 3.11 се извеждат техни свойства. В Теорема 3.12 са класифицирани (спрямо тези еквивалентности) неприводимите КНФ на $\tilde{0}$, съдържащи не повече от три променливи. Резултатите от този раздел са публикувани в [56, 25].

В раздел 3.3, в Теорема 3.14 са получени необходими условия, а в Теорема 3.15 – достатъчни условия за неудовлетворимост при задачата 3-УБФ, които са полиномиално проверими. Извеждат се оценки за броя на удовлетворимите и на неудовлетворимите 3-КНФ. Резултатите са публикувани в [26].

В четвърта глава са изследвани m-ичните разбивания на суми от k събираеми, всяко от които е равно на m^n. Разглеждат се два случая: когато разбиванията не съдържат и когато е допустимо да съдържат тривиални разбивания на някои от събираемите. В раздел 4.2, чрез стратегията динамично програмиране, са построени два алгоритъма за преброяване на m-ичните разбивания на сумата $k.m^n$ в двата случая. Те са основани на рекурентни зависимости за броя на тези разбивания, получени съответно в Теореми 4.3 и 4.4. В Следствия 4.6 – 4.12 и Следствие 4.18 са изведени комбинаторни, алгебрични и геометрични свойства на функциите на m-ично разбиване. Теореми 4.15 и 4.16 в раздел 4.4 доказват съществуването на полиномиално представяне на броя на разглежданите разбивания. С тяхна помощ е построен алгоритъм (с времева сложност $\Theta(n^3)$ и сложност по памет $\Theta(n^2)$), който определя полинома, представящ броя на всички m-ични разбивания на сумата $k.m^n$ и чрез него пресмята този брой. Представени са две приложения на разгледаните разбивания: чрез Теорема 4.21 е определен броят на пълните m-ични дървета от определен вид, а чрез Теорема 4.25 – броят на неекивалентните помежду си разбивания на n-мерния булев куб. Първите резултати, свързани с разбиванията на $\{0,1\}^n$, са публикувани в [3], а всички изследвания, представени в четвърта глава, са приети за публикуване в [27].

В раздел 5.2 се дефинира матрицата P_n, представяща предхожданията на векторите от $\{0,1\}^n$. Чрез Теореми 5.13 и 5.15 (и бележките след тях) са определени нейните структурни свойства, използвани за построяването на алгоритмите в пета глава. В раздел 5.3 е представен алгоритъмът Gen за лексикографско генериране на всички МБФ на n променливи ($0 \leq n \leq 7$), който работи за полиномиално общо време. В раздел 5.4 е предложен алгоритъм в две разновидности: Search_First и Search_Last – съответно за определяне на първи МИВ и на последен МНВ на неизвестна

МБФ на n променливи чрез въпроси за членство. Алгоритъмът решава съответната задача, като използва n въпроса за членство и работи за време $\Theta(n)$, без да извършва операции над векторите от $\{0,1\}^n$. В раздел 5.5 е представен алгоритъмът Identify, който идентифицира неизвестна $f \in M^n$ само чрез въпроси за членство. За целта алгоритъмът използва $O(nm)$ въпроса, но има експоненциална времева сложност. Резултатите, получени в пета глава, са публикувани в [2, 28, 29].

Литература

[1] Акимов О. Е., *Дискретная математика. Логика, групы, графы*, Лаборатория базовых знаний, Москва, 2001.

[2] Бакоев В., Генериране на подмножествата на частично наредено множество със запазване на сянката, *Сборник докл. от* IV *конф. "Приложения на математиката в икономиката и проблеми на обучението по математика и информатика"*, ВФСИ "Д.А. Ценов" Свищов и СМБ – Свищов, 1995, 186–192.

[3] Бакоев В., Разбивания на n–мерния булев куб на подкубове, *Годишник на ФМИ при ВТУ "Св. Св. Кирил и Методий"*, Изд. на ВТУ, 1998, 55–67.

[4] Виленкин Н., *Комбинаторика*, Наука, Москва, 1969.

[5] Гаврилов Г., Сапоженко А., *Сборник задач по дискретной математике*, Наука, Москва, 1977.

[6] Гайнанов Д., Об одном критерии оптимальности алгоритма расшифровки монотонных булевых функций, *Журнал вычисл. матем. и матем. физики*, т. 24, Н. 8, 1984, 1250–1257.

[7] Грэхем Р., Кнут Д., Паташник О., *Конкретная математика. Основание информатики*, Мир, Москва, 1998.

[8] Гэри М., Джонсон Д., *Вычислительные машины и труднорешаемые задачи*, Мир, Москва, 1982.

[9] Денев Й., Павлов Р., Деметрович Я., *Дискретна математика*, Наука и изкуство, София, 1984.

[10] Ерусалимский Я. М., *Дискретная математика: теория, задачи, приложения*, Вузовская книга, Москва, 2001.

[11] Катериночкина Н., Поиск максимального верхнего нуля для одного класа монотнных функций k-значной логики, *Доклады АН СССР*, т. 234, Н. 4, 1977, 746–749.

[12] Кнут Д., *Искусство программирования для ЭВМ. Том 1, Основные алгоритмы*, Мир, Москва, 1976.

[13] Ковалев М., Миланов П., Монотнные функции многозначной логики и суперматроиды, *Журнал вычисл. матем. и матем. физики*, т. 24, Н. 5, 1984, 786–789.

[14] Коршунов А., О числе монотонных булевых функций, *Проблемы кибернетики*, Вып. 38, Наука, Москва, 1981, 5–108.

[15] Липский В., *Комбинаторика для программистов*, Мир, Москва, 1988.

[16] Манев К., *Увод в дискретната математика*,(III изд.), Изд. КЛМН, София, 2003.

[17] Пенроуз Р., *Новият разум на царя. За компютрите, разума и законите на физиката*, Унив. изд. "Св. Климент Охридски", София, 1998.

[18] Соколов Н., Об оптимальной расшифровке монотонных функций алгебры логики, *Журнал вычисл. матем. и матем. физики*, т. 22, Н. 2, 1982, 449–461.

[19] Стенли Р., *Перечислительная комбинаторика*, Мир, Москва, 1990.

[20] Яблонский С., *Введение в дискретную математику*, Наука, Москва, 1979.

[21] Aho A., Hopcroft J., Ullman J., *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[22] Andrews G., Congruence properties of the m-ary partition function, *Journal of Number Theory* 3, 1971, 104–110.

[23] Andrews G., *The Theory of Partitions*, Addison-Wesley, 1976.

[24] Angluin D., Hellerstein L., Karpinski M., Learning read-once formulas with queries, *Journal of the ACM*, v. 40, No. 1, 1993, 185–210.

[25] Bakoev V., Manev K., Irreducible Conjunctive Normal Forms of the Zero Function II, *Mathematics and Education in Mathematics*, Sofia, 1997, 209–215.

[26] Bakoev V., Manev K., Some Necessary and Some Sufficient Conditions about the 3-Satisfiability Problem, *Mathematics and Education in Mathematics*, Sofia, 2000, 233–239.

[27] Bakoev V., Algorithmic approach to counting of certain types m-ary partitions, *Disctrete Mathematics*, (accepted in Febr. 2003, to appear).

[28] Bakoev V., Some properties of one matrix structure at monotone Boolean functions, *Proceedings of the EWM International Workshop on Groups and Graphs*, Varna, Bulgaria, 2002, 5–8.

[29] Bakoev V., Generating and Identification of Monotone Boolean Functions, *Mathematics and Education in Mathematics*, Sofia, 2003, 226–232.

[30] Beringer A., Aschenman G., Hoos H., Weis A., GSAT versus Simulated Annealing, *Proc. of the 11-th Eur. Conf. on Art. Intelligence (ECAI94)*, J. Wiley & Sons, 1994, 130–135.

[31] Bioch J., Ibaraki T, Generating and approximating of nondominanted coteries, *IEEE Trans. Parallel and Distributed Systems,* v. 6, No. 9, Sept. 1995, 905–914.

[32] Boros E., Hammer P., Ibaraki T., Kawakami K., Identifying 2-monotonic positive Boolean functions in polynomial time, *Lecture Notes in Computer Science*, 557, 1991, 104–115.

[33] Boros E., Hammer P., Ibaraki T., Kawakami K., Polynomial time recognition of 2-monotonic positive Boolean functions given by an oracle, *SIAM J. Comput.,* v. 26, No. 1, Febr. 1997, 93–109.

[34] Büning H., Xu D., The Complexity of Homomorphisms and Renamings of Minimal Unsatisfiable Formulas, *Proc. of the V-th International Symposium on the Theory and Applications of Satisfiability Testing (SAT2002)*, Cincinnati, Ohio, USA, 2002, 164–181.

[35] Cheeseman P., Kanefsky B., Taylor W., Where the Really Hard Problems Are, *Proc. ot the 12-th Joint Conf. on Art. Intelligence (IJCAI-91)*, Sidney, Australia, 1991, 331–337.

[36] Churchhouse R., Congruence properties of the binary partition function, *Proc. Cambridge Phill. Soc.*, 66, 1969, 371–376.

[37] Cormen T., Leiserson Ch., Rivest R., *Introduction to Algorithms*, The MIT Press, New York, 1990.

[38] Craenen B., Eiben A., Marchiori E., Solving Constraint Satisfaction Problem with Heuristic-based Evolutionary Algorithms, *Proc. of the Congress on Evolutionary Computations, (CEC2000)*, IEEE, 2000, 1571–1577.

[39] Ekin O., Hammer P., Kogan A., On connected Boolean functions, *Discrete Applied Mathematics*, 96–97, 1999, 337–362.

[40] Fleishner H., Kullmann O., Szeider S., Polynomial-Time Recognition of Minimal Unsatisfiable Formulas with Fixed Clause-Variable Difference, *Theoretical Computer Science*, 289, No. 1, 2002, 503–516.

[41] Frank J., Weighting for Godot: Learning Heuristics for GSAT, in *Proc of the 13-th National Conf. on Art. Intelligence (AAAI96)*, Portland, USA, MIT Press, 1996, 358–343.

[42] Fredman M., Khachiyan L., On the complexity of dualization of monotone disjunctive normal forms, *Journal of Algorithms*, 21, 1996, 618–621.

[43] Fröberg C.-E., Accurate estimation of the number of binary partitions, BIT 17, 1977, 386–391.

[44] Gent I., Walsh T., The Satisfiability Constraint Gap, *Artificial Intelligence*, v. 8, No. 1–2, 1996, 59–80.

[45] Gent I., Walsh T., The SAT Phase Transition, *Proc. of the 11-th Eur. Conf. on Art. Intelligence (ECAI94)*, J. Wiley & Sons, 1994, 105–109.

[46] Goldberg E., Testing satisfiability of CNF formulas by computing a stable set of points, *Proc. of the V-th Intern. Symposium on the Theory and Applications of Satisfiability Testing (SAT2002)*, Cincinnati, Ohio, USA, 2002, 54–69.

[47] Gottlieb J., Marchiori E., Rossi Cl., Evolutionary Algorithms for the Satisfiability Problem, *Evolutionary computation*, MIT Press, v. 10, No. 1, 2002, 35–50.

[48] Grimaldi R., *Discrete and Combinatorial Mathematics. An Applied Introduction*, IV-th ed., Addison-Wesley, 1999.

[49] Guptan H., On m-ary partitions, *Proc. Cambridge Phill. Soc.* 71, 1972, 343–345.

[50] Gurvich V., Khachiyan L., On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions, *Discrete Applied Mathematics*, 96–97, 1999, 363–373.

[51] Iwama K., Matsura A., Inclusion-Exclusion for k-CNF Formulas, in *Proc. of the V-th Intern. Symposium on the Theory and Applications of Satisfiability Testing (SAT2002)*, Cincinnati, Ohio, USA, 2002, 182–189.

[52] Johnson D., Yannakakis M., On generating all maximal independent sets, *Information Processing Letters,*, 27, 1988, 119–123.

[53] Kirousis L., Kranakis E., Krizanc D., Stamatiou Y., Approximating the Unsatisfiability Threshold for Random Formulas, *Random Structures and Algorithms*, v. 12, No. 3, 1998, 253–269

[54] Makino K., Ibaraki T., The maximum latency and identification of positive Boolean functions, *SIAM J. Comput.*, v. 26, No. 5, Oct. 1997, 1363–1383.

[55] Makino K., Ibaraki T., A fast and simple algorithm for identifying 2-monotonic positive Boolean functions, *Journal of Algorithms*, v. 26, 1998, 291–305.

[56] Manev K., Bakoev V., Irreducible Conjunctive Normal Forms of the Zero Function, *Proceedings of the Fifth International Conference on Discrete Mathematics and Applications*, v. 5, Blagoevgrad, Bulgaria, 1995, 127–131.

[57] Marchioni E., Rossi C., A Flipping Genetic Algorithm for Hard 3-SAT Problems, *Proc. of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 1999, 393–400.

[58] Reis N., de Sousa J., On Implementing a Configware/Software SAT Solver, in *Proc. of the 10-th Anual IEEE Symposium on Field-Porgramable Custom Computing Machines (FCCM'02)*, Napa, California, 2002.

[59] Rödseth Ö., Some arithmetical properties of m-ary partitions, *Proc. Cambridge Phill. Soc.*, 68, 1970, 447–453.

[60] Shmulevich I., Korshunov A., Astola J., Almost all monotone Boolean functions are polynomially learnable using membership queries, *Inform. Process. Letters*, 79, 2001, 211–213.

[61] Shuler R., Schöning U., Watanabe O., A probabilistic 3-SAT algorithm further improved. In *Proc. of the 19-th Ann. Symp. on Theoret. Aspects of Comp. Science (STACS2002)*, v. 2285 of *Lect. Notes in Comp. Science*, Springer, 2002, 192–202.

[62] Szeider S., Generatizations of matched CNF formulas, *Proc. of the V-th International Syposium on the Theory and Applications of Satisfiability Testing* (SAT 2002), Cincinnati, Ohio, USA, 2002, 292–307.

[63] Szeider S., Homomorphisms of Conjunctive Normal Forms, *Discrete Applied Mathematics*, (accepted in May 2000, to appear, available electronically at http://www.cs.toronto.edu/~szeider/).

[64] Uribe T., Stickel M., Ordered Binary Decision Diagrams and the Davis-Putnam Procedure, *Lect. Notes in Comp. Science*, v. 845, Sept. 1994.

[65] Wegener I., *The Complexity of Boolean Functions*, J. Wiley & Sons, and B. G. Teubner, Stuttgart, 1987.

[66] Werchner R., Harish T., Drechsler R., Becker R., Satisfiability problem for Ordered Functional Decision Diagram, *Representation of Discrete Functions*, Kluiwer Acad. Publ., 1996, 233–248.

[67] Yovovic V., Kilibrada G., On the number of Boolean functions in the Post classes F_8^μ, Discrete Mathematics and Applications, v. 9, No. 6, 1999, 563–586.

Адреси в Интернет

[68] http://home.san.rr.com/ronz/index.html – *Ron Zeno's site (Ramblings in mathematics and computer science)*, (2002).

[69] http://mathpages.com/home/kmath030.htm – *Dedekind's Problem.*

[70] http://mathpages.com/home/kmath094.htm – *Generating the Monotone Boolean Functions.*

[71] http://mathpages.com/home/kmath515.htm – *Progress on Dedekind's Problem.*

[72] Sloane N. (2003), *The On-Line Encyclopedia of Integer Sequences*, published electronically at http://www.research.att.com/~njas/sequences/