БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ * ACADEMIE BULGARE DES SCIENCES
ИЗВЕСТИЯ НА МАТЕМАТИЧЕСКИЯ ИНСТИТУТ
BULLETIN DE L'INSTITUT DE MATHÉMATIQUES
Том (Tome) XIII

ОБ ОДНОЙ ЗАДАЧЕ УПРАВЛЕНИЯ РАЗРАБОТКАМИ

Геро Геров и Николай Стоянов

В процессе управления большими системами широко применяются сетевые модели. Для их обработки известны уже много алгоритмов. Здесь будет рассмотрен вопрос о возможностях автоматического составления сетевого графика в случае частично упорядоченного множества работ.

При подготовке исходной информации для составления сетевого графика удобнее всего потребовать, чтобы каждый исполнитель указал все работы, которые непосредственно предшествуют его работе и без выполнения которых он не в состоянии начать выполнение своей работы. Из этой информации надо составить всю сеть.

Другими словами, для каждой выходящей дуги графа известна совокупность дуг, входящих в вершину, из которой она выходит, т. е. граф задан своими частичными звездами.

Требуется, во-первых, скомпоновать вершины графа, т. е. собрать вместе все дуги, имеющие одинаковые совокупности „входящих" дуг, т. е. перейти от частичных звезд к звездам графа, и, во-вторых, подходящим образом перенумеровать все дуги и вершины так, чтобы когда из i-той вершины существует путь в j-тую, было выполнено условие $i < j$.

Основные предложения. Будем считать, что входная информация для составления сети задается в следующем виде:

$$\begin{array}{llllll} P_i, & v_i, & P_{i1}, & P_{i2} & ., & P_{in_i} \\ P_{i+1}, & v_{i+1}, & P_{i+1\,1}, & P_{i+1\,2}, . & & P_{i+1\,n_{i+1}} \end{array}$$

Здесь P_i — „название" i-той работы; P_{ik} — „название" работ, непосредственно предшествующих i-той работе. Числам v_i в начале алгоритма присвоено нуль.

Под „названием" будем понимать „короткие названия", т. е. будем считать, что существует список всех возможных работ, которые можно встретить в подобной разработке. В этом списке все работы описаны „длинными" словесными названиями и сквозным образом перенумерованы. Номер, соответствующий данной работе, будем считать ее „коротким" названием. Этим мы добиваемся того, что все названия будут иметь одинаковую стандартную длину.

Случай, когда исходная информация составлена так, что все работы участвуют со своими длинными названиями, легко сводится к случаю с короткими названиями.

Еще будем предполагать, что конечная работа данной сети заранее известна и каким-то образом помечена. Хотя это предположение несущественно, оно уменьшает на единицу количество переборов всей исходной информации.

Все работы, у которых отсутствуют коды P_{ik}, являются начальными. Их может быть несколько. Будем считать, что сеть обладает только одной конечной работой, хотя и это предположение не существенно.

Введем еще обозначение

$$\Delta_i = \{P_{i1}, P_{i2}, \ldots, P_{in_i}\}.$$

Прежде чем рассмотреть решение поставленной задачи, дадим один алгоритм быстрого поиска информации в памяти ЭВМ. Он обобщает предложенный в [1] алгоритм для случая, когда надо быстро разыскать n кодов, расположенных в некоторой большой совокупности из m кодов, $m>n$, и среди кодов $X_{k_1}, X_{k_2}, \ldots, X_{k_n}$ есть повторяющиеся. Алгоритм будет состоять из двух частей A и B, которые будем называть коротко A-алгоритмом и B-алгоритмом. Прежде всего, как и в [1], для каждого кода X_i вырабатывается псевдономер $N_i(X_i)$, однозначно соответствующий коду X_i.

A-алгоритм. Вводится $4n$ вспомогательных числа p_i, q_i, u_i, r_i, $i=0, 1, \ldots, n-1$. В начале алгоритма $u_i=0$, $q_i=0$, $p_i=N_i$, а величинам r_i присваивается точный адрес кода X_i в большой совокупности. Числа r_i в ходе алгоритма больше не меняются. Далее осуществляется рекурентный процесс, на $n-k$-том шаге, $n-k=1, 2, \ldots, n$, которого p_k присваивается u_{N_k}, а новое значение u_{N_k} полагается равным k. Введем обозначения $p_k=\delta_1$ и $\delta_i=p_{\delta_{i-1}}$ для $i>1$. На каждом шаге код X_k сравнивается с кодом X_{z_1}, где $z_1=r_{p_k}$. Если они совпадают, то полагается $q_k=p_k$, $p_k=p_{\delta_1}$, а самому p_{δ_1} присваивается нуль. Если они не совпадают, сравниваем X_k с X_{z_2}, где $z_2=r_{\delta_2}$. Если они снова не совпадут, то сравниваем с кодом X_{z_3}, где $z_3=r_{\delta_3}$, и т. д., пока наконец $X_k=X_{z_l}$. Тогда полагается $q_k=\delta_k$, $p_{\delta_l}=0$, а $p_{\delta_{l-1}}=P_{\delta_l}$. Если при этом сравнивании получится $p_{z_i}=0$, то полагается $q_k=0$. Это означает, что этот код встречается в первый раз.

B-алгоритм решает задачу быстрого нахождения кода X_i. По заданному коду Y находится псевдономер $N(Y)$. Находим адрес r_{z_1}, где $z_1=u_{N(Y)}$, и сравниваем код $X_{r_{z_1}}$ с кодом Y. Если они совпадают, то запоминаем адрес этого кода. Берем число и находим адрес r_{z_2}, где $z_2=q_{z_1}$ и т. д., пока не получим $q_{z_1}=0$. Это будут все повторяющиеся коды Y в большой совокупности.

Если код $X_{r_{z_2}}$ не совпадает с Y, то берем $z_2=p_{z_1}$ и сравниваем код $X_{r_{z_2}}$ с Y. Если совпадут, то берем дальше $z_i=q_{z_{i-1}}$ и находим адреса всех кодов, равных Y. Если снова нет совпадения, берем $z_3=p_{z_2}$ и т. д. Очевидно число операций при реализации на ЭВМ этих алгоритмов тоже

линейно зависит от n. Для решения поставленной задачи предлагается следующий алгоритм.

1. Все совокупности Δ_i упорядочиваем так, чтобы

$$P_{i1} < P_{i2} < \cdots < P_{ik_i}$$

и одновременно подсчитываем максимальную длину последовательности кодов Δ_i, $i=1, 2, \ldots, n$. Тогда любую такую совокупность Δ_i будем рассматривать как один более длинный код Δ_i. Для того, чтобы все коды такого вида имели одинаковую длину, дополняем более короткие до максимального добавлением к концу нулей. К совокупности Δ из таких уже стандартизованных длинных кодов применяем A-алгоритм. Тот же алгоритм применяем и к совокупности P кодов P_i.

2. Берем первый элемент Δ_i из Δ. Применяя B-алгоритм, находим все элементы Δ_i из Δ, равные Δ_1. Всем соответствующим v_i присваиваем номер 1. Находим следующий после Δ_1 элемент Δ_j, для которого $v=0$. С ним проводим аналогичную процедуру и всем полученным таким образом присваиваем очередной номер 2, и т. д. до исчерпывания совокупности Δ.

3. Рассматриваем отдельно каждый код P_{ij}, входящий в Δ_i. Применяя B-алгоритм, находим его в совокупности P. Пусть это код P_l. Соответствующее ему v_l ставится на место P_{ij}. Если какого-нибудь кода P_{ij} не будет в совокупности P, это означает, что работа P_{ij} является начальной. Когда все коды, содержащиеся в Δ_i, будут заменены соответствующими значениями v, заменяем P_i на v_i и стираем все коды Δ_k, совпадающие с Δ_i. Берем следующий оставшийся код в Δ и продолжаем аналогично. Когда вся совокупность Δ будет таким образом рассмотрена, остается только применить один из известных алгоритмов перенумерации дуг сети полученного графа.

Легко подсчитать, что число требуемых операций для присваивания нового номера любой вершине графа изложенным алгоритмом не зависит от n, следовательно весь процесс осуществляется за число операций, линейно зависящее от n.

Ввод фиктивных работ. В изложенном выше алгоритме делалось молчаливое предположение, что любая работа P_{ij} участвует только в одном Δ_i, т. е. что для всех i и j либо

$$\Delta_i \bigcap \Delta_j \neq \emptyset,$$

либо

$$\Delta_i \equiv \Delta_j.$$

Из графа исключались все конфигурации типа как на рис. 1. Однако, если все же такие конфигурации существуют в графе (что очень часто бывает, и при сборе исходной информации не всегда удается установить, есть ли такие случаи или нет), применение алгоритма составления сети доведет до того, что одна и та же работа появится в сети несколько раз, исходя из одной вершины и заходя в несколько. Так, например, в ситуации на рис. 1 все три вершины получат разные номера, входящие в них работы, сдублируются (рис. 2). Это дублирование работ не всегда желательно и часто мешает дальнейшей обработке сетевого графика.

Чтобы избавиться от этого неудобства, удобно ввести рассмотрение фиктивных работ. Тогда, например, конфигурация на рис. 2 может при-

нять вид как на рис. 3. Для того, чтобы изложенный выше алгоритм охватывал и эти случаи, дополняем его следующим образом.

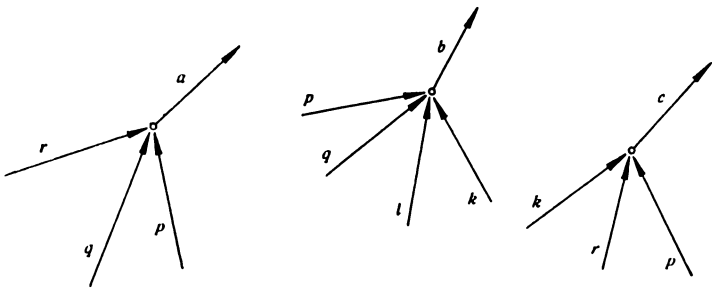

Рис. 1

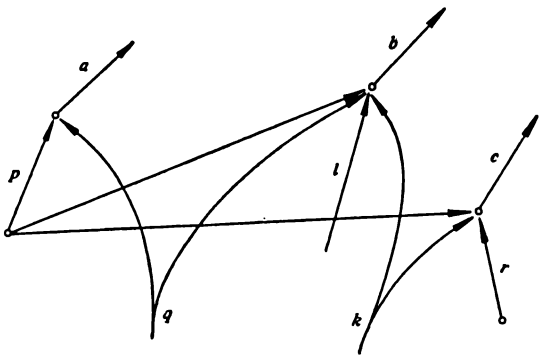

Рис. 2

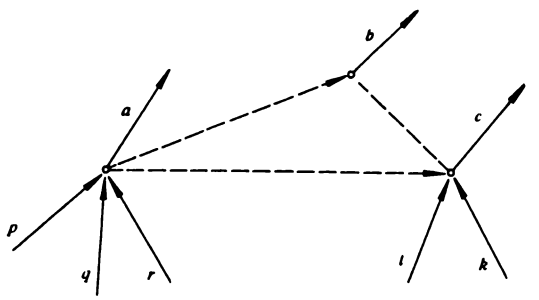

Рис. 3

В пункте 3 для каждого кода P_{ij} разыскивался в совокупности P равный ему код P_k и на месте P_{ij} ставился номер соответствующей вершины графа v_k. Эту операцию дополним сейчас тем, что на месте v_k засылается точный адрес кода P_{ij}. При этом организация памяти должна быть такой, чтобы запись адреса и запись величины v_i отличались друг

от друга. Для этого достаточно, чтобы поле, на котором расположены P_{ij}, начиналось адресом большим $\max\limits_i v_i$.

Если нет необходимости вводить в сеть фиктивные работы, т. е. любая работа участвует только в одной совокупности Δ_i, то все v_i будут меняться только один раз. Если, однако, к v_i придется обратиться еще раз, то это значит, что соответствующая работа участвует хотя бы в двух Δ_i, т. е. придется ввести фиктивную работу. При повторном обращении к v_i там уже есть адрес, где хранится номер вершины. По этому адресу находим величину, которую должны присвоить соответствующему P_{ij}. Запоминаем

(1) $$P_i, \text{ адреса } P_{ij_1}, P_{ij_2}, \ldots$$

и величине v_i присваиваем новый адрес $P_{i_2 j_2}$. Так что величина v_i дает всегда либо номер вершины сети, либо адрес последней работы P_{ij}, которой присвоен этот номер.

При дальнейшем обращении к v_i к последовательности (1) уже добавляются только новые номера $P_{i_3 j_3}$, $P_{i_4 j_4}, \ldots$ В конце этой операции каждому номеру $P_i \in P$ присваивается номер вершины, которой раньше был в v_i. Этот номер находим по адресу, хранящемуся в v_i.

По адресам, выделенным в (1), находим новые номера всех работ, которые порождают фиктивные работы в сети.

В дальнейшем применяется алгоритм переупорядочивания. Тогда все работы получат новые номера $i_1 j_1$, $i_2 j_2, \ldots$

Из всех работ, исходящих из одной вершины

(2) $$i_s j_1, i_s j_2, \ldots, i_s j_k,$$

есть одна, для которой $j_l = \min\limits_i j_i$. Эту работу сохраняем. Все остальные работы (2) заменяются фиктивными работами, исходящими из вершины j_l.

Очевидно, данное дополнение к алгоритму не увеличивает порядок требуемых машинных операций, которые все еще линейно зависят от числа n входящих работ. Этим задача уже решена полностью.

Замечание 1. Если алгоритм ввода фиктивных работ комбинировать с алгоритмом подсчета критического времени, ввод фиктивных работ можно изменить так, чтобы действительной работой оставалась та, у которой $T_i = \min$, а все остальные ввести как фиктивные. Хотя в этом случае алгоритм усложнится, полученная таким образом сеть точно будет соответствовать действительности.

Замечание 2. Ввод фиктивных работ можно осуществить и так. Конец каждой работы, входящей в различные совокупности Δ_i, обособляется в отдельное событие. Из этого конца уже проводятся фиктивные работы. В этом случае, однако, число вершин графа может сильно возрасти.

ЛИТЕРАТУРА

1. Майзлин, И. Е. Об одном способе поиска информации и его применении при реализации на ЭВМ алгоритма нахождения критического пути. Доклады АН СССР, **159** (1964), № 4, 761—793.

Поступило 17. III. 1970 г.

ВЪРХУ ЕДНА ЗАДАЧА ЗА УПРАВЛЕНИЕ НА РАЗРАБОТКИТЕ

Геро Геров и Николай Стоянов

(Резюме)

Разглежда се въпросът за автоматично съставяне мрежовия модел в случая, когато за всяка изходна дъга от графа е известна съвкупността от входящите дъги. Предлага се алгоритъм за композиране на графа, при което върховете се номерират подходящо. Разглежда се и случаят на въвеждане фиктивни работи.

ON ONE PROBLEM OF THE DEVELOPMENT CONTROL

Gero Gerov and Nikolai Stojanov

(Summary)

The automatic composition of a network model is considered for the case when for each outward arc the set of the inward arcs is known. A graph composition algorithm is proposed with an appropriate numbering of the peaks. Also the case when ficticious work is introduced is considered