АКАДЕМИЯ НАУК СССР

ЖУРНАЛ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Том 14

(ОТДЕЛЬНЫЙ ОТТИСК)

5

МОСКВА · 1974

ЖУРНАЛ
ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И МАТЕМАТИЧЕСКОЙ ФИЗИКИ

Том 14 Сентябрь 1974 Октябрь № 5

УДК 519.95

ЭКОНОМНЫЙ АЛГОРИТМ ВЫДЕЛЕНИЯ БЛОКОВ В ГРАФЕ

Е. А. ДИНИЦ, М. А. ЗАЙЦЕВ, А. В. КАРЗАНОВ

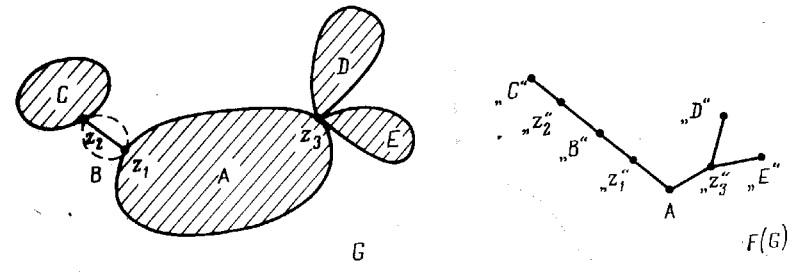
(Москва)

Предлагается алгоритм, организующий поиск блоков в связном графе. Оценка числа действий алгоритма по порядку минимально возможная: $O(p)$, где p – число ребер графа.

Рассматривается связный неориентированный граф G без кратных ребер и петель с множеством вершин V, $|V|=n$, и множеством ребер U, $|U|=m$.

1. Блоки в графе[*]. Вершина z графа G называется *разделяющей*, если при удалении ее (вместе со всеми инцидентными ей ребрами) нарушается связность графа. Граф, не содержащий разделяющих вершин, называется *несепарабельным*. Компонентой несепарабельности или *блоком* графа G называется максимальный по включению несепарабельный подграф[**] его. Известно, что количество блоков в G не превосходит $n-1$.

Граф блоков $F(G)$ графа G определяется следующим образом. Его вершины соответствуют блокам и разделяющим вершинам графа G (см. фиг. 1). Вершины «z» и «B» графа блоков соединены ребром при $z \in B$ (здесь z – разделяющая вершина, а B – блок).



Фиг. 1

Известно, что разделению графа на блоки соответствует разбиение множества его ребер на непересекающиеся классы по отношению эквивалентности $R: R(u_1, u_2) \Leftrightarrow$ существует простой (несамопересекающийся по вершинам) цикл, содержащий ребра u_1 и u_2.

[*] Обоснование свойств графов, описанных ниже, см. в [1, 2].

[**] Подграфом $G(V')$, $V' \subset V$, называется часть графа G с множеством вершин V', содержащая все ребра, которые соединяют эти вершины в графе G.

Ветвью блоков, висящей на (разделяющей) вершине z, называется подграф $G'=G(V'\cup\{z\})$, $V'\not\ni z$, такой, что вершины из V' не соединены ребрами с вершинами из $V\setminus(V'\cup\{z\})$ и подграф $G(V')$ связен (т. е. вершина z не разделяющая в G'). Очевидно, вершина z принадлежит только одному блоку в графе G', будем называть его *корневым* для ветви G'.

Пусть фиксирована вершина a. Тогда каждый блок B (кроме $B\ni a$, если a — не разделяющая в G) выступает как корневой для единственной ветви блоков из класса $\{G'\,|\,a\notin V'\}$.

2. Нахождение разделяющих вершин может потребоваться при решении задач на электрических, надежностных и других сетях. Например, если решается задача распараллеливания вычислений, то возникает необходимость узнать, какие части программы обязательно будут работать (независимо от конкретной реализации процесса в точках разветвления схемы вычислений). На схеме таким частям соответствуют вершины, отделяющие выход от входа. Разделение графа на блоки может существенно понизить размерность задач, заданных на больших сетях, и тем самым резко сократить трудоемкость их решения.

Экономные алгоритмы решения близкой задачи о «более грубом» разделении графа на бикомпоненты предлагались в [3–5].

В настоящей статье предлагается алгоритм, выделяющий блоки и разделяющие вершины в заданном графе G, а также строящий граф блоков $F(G)$. Трудоемкость алгоритма $O(m)$ действий, а объем требуемой памяти $O(m)$ ячеек. Для справедливости этих оценок необходимо, чтобы граф был задан в виде списков для каждой вершины v графа вершин, соседних с ней (т. е. соединенных с ней ребром).

Предлагаемый алгоритм основан на определенном способе обхода графа, описанном ниже. В результате изучения свойств обхода алгоритм будет получен принципиально. Затем будет дано формальное описание алгоритма, удобное для программирования.

3. Под обходом Q графа G будем понимать движение от вершины к вершине по ребрам графа, причем каждое ребро проходится не более одного раза в каждом направлении. Траекторию движения при обходе обозначим Q, начальную вершину обозначим a (она может быть любой). Будем считать, что последовательным вершинам $a=v_0, v_1, \ldots, v_\alpha, \ldots, v_\omega$ маршрута Q соответствуют последовательные моменты $0, 1, \ldots, \alpha, \ldots, \omega$ обхода, а ребрам его — ходы обхода. Первый ход по ребру будем называть прямым, второй (в противоположном направлении) — обратным. Условимся, что прямой ход задает ориентацию ребер (связанную с данным обходом). Далее запись $u=(y, z)$ будет означать, что ребро u ориентировано от y к z. Ребро, по которому мы впервые попадаем в вершину v (очевидно, прямым ходом), будем обозначать u_v.

Правила обхода. а. Если мы впервые оказываемся в вершине v, то следующий ход должен быть прямым по любому ребру, инцидентному v и не фигурировавшему ранее в обходе, если такое существует; в противном случае следующим должен быть обратный ход по ребру u_v при $v\neq a$ и окончание обхода при $v=a$.

б. Если прямым ходом мы попадаем в вершину, уже фигурировавшую в обходе, то следующий ход должен быть обратным по тому же ребру (такое ребро графа и такой момент обхода будем называть бетрефальными, от немецкого betreffen — касаться).

в. Если обратным ходом мы попадаем в вершину v, то далее поступаем, как в случае а.

Основное (здесь) свойство обхода по правилам а — в состоит в том, что, войдя в ветвь блоков G' по ребру u, мы находимся внутри G' до тех пор, пока не обойдем ее полностью, заканчивая движение по G' обратным ходом по ребру u (см. ниже).

4. Изучим свойства обхода, построенного по описанным правилам.

Рассмотрим основную (небетрефальную) часть $\overline{Q}$ обхода Q, получаемую при выбрасывании из Q всех бетрефальных участков. Так как выбрасываются только ходы туда-обратно по бетрефальным ребрам, то $\overline{Q}$ может рассматриваться, как обход части $\overline{G}$ графа G, образованной небетрефальными ребрами, т. е. ребрами типа u_v, где v пробегает $V\setminus\{a\}$. Легко проверить, что $\overline{G}$ — дерево, ориентированное от корня a. Для каждой вершины $x\in\overline{G}$ существует и единствен ориентированный путь $L_x\subset\overline{G}$ из a в x.

Будем называть ребро, не фигурировавшее в обходе, *непройденным*, пройденное один раз — *полупройденным*, пройденное два раза — *пройденным*.

Утверждение 1. *Множество полупройденных ребер в любой момент α обхода $\overline{Q}$ (т. е. в любой небетрефальный момент обхода Q) совпадает с множеством ребер «текущего» пути $L_\alpha=L_{v_\alpha}$.*

(Легко доказывается по индукции.)

Назовем вершину *свободной*, если все инцидентные ей ребра непройденные, *замкнутой*, если все они пройденные, и *открытой* — в остальных случаях. Рассмотрим вершину $v\neq a$. Очевидно, что она перестает быть свободной в тот момент, когда включается в текущий путь вместе с входящим в нее ребром u_v. Докажем, что вершина v становится замкнутой в тот момент, когда текущий путь перестает включать v и u_v в результате обратного хода по ребру u_v. Действительно, по правилу а вершине v не инцидентны непройденные ребра, а по утверждению 1 — полупройденные, что и требовалось доказать. Аналогично доказывается, что в момент ω окончания обхода вершина a замкнута. Тем самым, доказано

Утверждение 2. *Множество открытых вершин совпадает с множеством вершин текущего пути, не считая начального и конечного моментов, когда оно пусто.*

Утверждение 3. *В момент ω все ребра пройденные и все вершины замкнутые.*

(Доказывается так же, как для обхода, описанного в [6].)

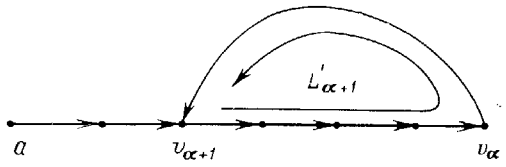
Следствие. Дерево $\overline{G}$ содержит все вершины графа G, т. е. представляет собой его каркас (связывающее дерево). (Тем самым, описанный метод обхода графа может быть использован для построения каркаса заданного графа.)

Каркас $\overline{G}$ обладает важным специфическим свойством.

Утверждение 4. *Пусть $u=(y, z)\in U$ — хорда каркаса $\overline{G}$ (т. е. $u\notin\overline{G}$). Тогда $z\in L_y$.*

Доказательство. По определению $\overline{G}$, u — бетрефальное ребро. Пусть прямой ход по нему был совершен в момент α, т. е. $v_\alpha = y$, $v_{\alpha+1} = z$. Вершина z не свободна, по определению бетрефальности, и не замкнута, по определению замкнутости. Значит, z открыта и, по утверждению 2, $z \in L_\alpha = L_y$.

Структуру множества полупройденных ребер в бетрефальный момент $\alpha+1$ можно проиллюстрировать фиг. 2, где $L'_{\alpha+1}$ — контур, образующийся в момент $\alpha+1$ из полупройденных ребер.



Фиг. 2

5. По структуре, задаваемой на графе G каркасом $\overline{G}$, однозначно определяется любая ветвь блоков G', $V' \not\ni a$, а именно, множество вершин ветви блоков образует в каркасе $\overline{G}$ отделимую ветвь в следующем смысле.

Рассмотрим часть каркаса $\overline{G}$, отделяемую ребром u от корня дерева $\overline{G}$ — вершины a. Присоединим к ней само ребро $u=(z, y)$. Полученную часть $\overline{G}_u$ каркаса $\overline{G}$ будем называть его *ветвью*, висящей на ребре u и на вершине z. Множество вершин ветви $\overline{G}_u$, кроме z (внутренних), обозначим V_u. Ветвь $\overline{G}_u$ будем называть отделимой, если из вершин множества V_u не исходит ребер (необходимо бетрефальных) в вершины из $V \setminus (V_u \cup \{z\})$, точнее, в вершины L_z, кроме z (уточнение справедливо в силу утверждения 4).

С одной стороны, очевидно, что отделимая ветвь $\overline{G}_u$, $u=(z, y)$, задает ветвь блоков с множеством внутренних вершин $V_u \not\ni a$, висящую на вершине z.

Обратно, рассмотрим ветвь блоков G', $V' \not\ni a$, висящую на вершине z. Так как $\overline{G} \subset G$, вершина z в каркасе $\overline{G}$ отделяет V' от остальных вершин. Значит, V' образует в $\overline{G}$ одну или несколько ветвей, висящих на z. Однако, в силу утверждения 4, внутренние вершины из различных ветвей, висящих на одной вершине, не могут соединяться ребрами. Поэтому связность $G(V')$ влечет за собой единственность такой ветви.

6. Рассмотрим теперь задачу нахождения отделимых ветвей в процессе обхода.

Очевидно, обход ветви $\overline{G}_u$ есть часть обхода Q, начинающаяся прямым ходом по ребру u и кончающаяся обратным ходом по нему. Часть обхода Q, заключенную в этом промежутке, назовем ветвью обхода и обозначим Q_u. Определение отделимости распространим на ветви обхода: Q_u отделима $\Leftrightarrow \overline{G}_u$ отделима. Из описанных выше свойств обхода легко вывести, что ветвь обхода Q_u при $u=(z, y)$ представляет собой обход подграфа $G_u = G(V_u \cup \{z\})$ плюс бетрефальные попадания в вершины пути L_z, кроме z. Тем самым можно считать обоснованным следующий

Критерий отделимости: ветвь Q_u, где $u=(z, y)$, отделима в том и только том случае, когда на протяжении Q_u не было бетрефальных попаданий в вершины текущего пути, расположенные на нем ближе к началу (вершине a), чем вершина z.

Покажем, как можно использовать этот критерий в процессе обхода. Каждое небетрефальное ребро с момента появления его на текущем пути (т. е. после прямого хода по нему) будем считать «подозрительным». Метка подозрительности снимается следующим образом. Пусть в момент α был совершен прямой ход по бетрефальному ребру (y, z): $v_\alpha=y$, $v_{\alpha+1}=z\in L_\alpha$. Тогда, очевидно, все ребра, расположенные на текущем пути L_α дальше от начала, чем ребро, исходящее из z (т. е. все ребра контура $L'_{\alpha+1}$, кроме ребра, исходящего из z), не могут порождать отделимую ветвь обхода. Поэтому все такие ребра, имеющие в рассматриваемый момент метку подозрительности, освобождаются от нее. Очевидно, что отделимость ветви обхода Q_u эквивалентна тому, что к моменту обратного хода по ребру u оно еще считается «подозрительным».

Таким образом, принципиально получен алгоритм нахождения отделимых ветвей обхода. Но для того, чтобы этот алгоритм стал в достаточной степени экономным (в смысле оценки трудоемкости по порядку), необходимо указать способ стирания меток подозрительности, не требующий в каждый бетрефальный момент просмотра всех ребер образующегося контура. Этот способ состоит в следующем.

Нужно иметь список «подозрительных» ребер (с.п.р.) в том порядке, в каком они расположены на текущем пути. При этом не тривиально организовать лишь вычеркивание элементов из с.п.р. В бетрефальный момент из с.п.р. отбрасывается «хвост» (некоторое количество элементов с конца), поэтому задача состоит в том, чтобы найти элемент из с.п.р., который в результате станет последним. Этот элемент можно было бы найти просмотром с.п.р. с конца, если бы вершины обладали признаком, позволяющим определить порядок их расположения на текущем пути. Поэтому предлагается следующее.

Вершины графа нужно нумеровать в процессе обхода числами $1, 2, \ldots, n$ в порядке первого появления их в обходе. Текущий путь изменяется только с «хвоста», причем при наращивании его к нему присоединяется вершина с номером бо́льшим, чем любой ранее присвоенный. Поэтому можно утверждать, что для любого α на пути L_α номера вершин возрастают от a к v_α (что и требовалось).

Покажем теперь, как нахождение отделимых ветвей обхода влечет за собой выделение блоков графа G. Пусть в момент обратного хода по ребру $u=(z, y)$ обнаружилось, что ветвь Q_u отделимая. Тогда множество внутренних вершин соответствующей ветви блоков есть V_u и может быть найдено в этот момент как множество вершин, встретившихся в обходе не ранее, чем вершина y. Для того чтобы в момент нахождения ветви блоков выделить множество вершин ее корневого блока (как было указано в п.1 статьи, этого достаточно для выделения всех блоков графа), достаточно при нахождении каждой ветви блоков множество ее внутренних вершин

исключать из дальнейшего рассмотрения. Тогда к моменту завершения обхода любой ветви блоков G' все ее подветви, висящие на корневом блоке B, будут обойдены, а их внутренние вершины исключены. Останутся лишь вершины из B, что и требовалось. Для экономного выполнения описанных операций предлагается вести текущий список вершин (т.с.в.) в порядке возрастания номеров. Исключение вершин из т.с.в. будет сводиться к отбрасыванию его «хвоста».

7. Итак, переходим к формальному описанию алгоритма, а именно к описанию дополнительных операций, которые нужно исполнять в процессе обхода, чтобы выделить блоки и разделяющие вершины графа G.

а′. Пусть в момент α мы попадаем впервые в вершину v. Присваиваем ей очередной номер $N(v)$. Заносим ее (подразумевается ее идентификатор) в конец массива т.с.в. Ребро u_v заносим в конец массива с.п.р.

б′. Пусть был сделан прямой ход по бетрефальному ребру $(v_\alpha, v_{\alpha+1})$. Просматриваем с.п.р. с конца до первого его элемента — ребра (x, y), для которого $N(x) \leqslant N(v_{\alpha+1})$. Это ребро считаем далее последним в с.п.р.

в′. Пусть обратным ходом по ребру $u_y=(z, y)$ «возвращаемся» в вершину z. Если ребро u_y стоит последним в с.п.р., то делаем следующее. Переносим конец с.п.р. на ячейку назад. Помечаем вершину z как разделяющую, если этого признака у нее еще нет, и заносим z в список нового блока B. Просматриваем т.с.в. с конца, занося просмотренные вершины в список вершин блока B. Процесс продолжаем до тех пор, пока не дойдем до вершины y и не занесем ее. Далее последней в т.с.в. будем считать предыдущую для y вершину в нем.

г′. В конце обхода все вершины из т.с.в. заносим в список очередного блока A $(a \in A)$, если в т.с.в. кроме a содержится еще хотя бы одна вершина.

Если требуется построить также граф блоков $F(G)$, то дополнительные операции следующие.

в″. Добавим к уже построенной части дерева $F(G)$ вершину «B», соответствующую найденному очередному блоку B, и «z», если z не была помечена как разделяющая ранее.

В момент занесения вершины v в список блока B проверяем, не помечена ли эта вершина как разделяющая. Если помечена, то «проводим» в $F(G)$ ребро между «v» и «B».

г″. Аналогично в″.

8. Оценка трудоемкости алгоритма. Рассмотрим вначале собственно обход графа. Покажем сначала, как организовать информацию, чтобы в любой момент поиск непройденного ребра, инцидентного любой вершине, требовал количества действий, не превосходящего некоторой константы.

Перед началом обхода для каждой вершины v списку вершин, соседних с v, придадим ссылочный характер: с каждым элементом списка свяжем две ячейки, содержащие ссылки на предыдущий и последующий элементы списка. С вершиной v свяжем ссылку на первый элемент списка. Для того

чтобы этот список всегда соответствовал непройденным ребрам, достаточно в нужные моменты вычеркивать элементы из него. Вычеркивание элемента здесь, очевидно, требует константы действий.

При такой организации, как легко видеть, однократное выполнение любого из правил а — в имеет константную трудоемкость. Значит, трудоемкость совершения всего обхода, пропорциональная числу ходов его, не превосходит $O(m)$ действий.

Перейдем к оценке трудоемкости алгоритма, т. е. действий по правилам а′ — г′, в″ и г″:

1) каждая вершина графа один раз заносится в т.с.в., один раз просматривается при обратном ходе, один раз заносится в список вершин очередного блока и проверяется, не разделяющая ли она (не считая действий с вершиной, на которой блок «висит»),— это $O(n)$ действий;

2) каждое ребро один раз заносится в с.п.р., один раз просматривается при обратном ходе по нему (не считая просмотра ребра, на котором обратный ход останавливается) — это $O(m)$ действий;

3) собственно построение $F(G)$ — это $O(n)$ действий;

4) все остальные операции при однократном исполнении любого из правил не требуют работы со списками, т. е. имеют константную трудоемкость — это еще $O(m)$ действий.

Таким образом, учитывая неравенство $n \leqslant m+1$, известное для связного графа, получаем, что оценка трудоемкости всего алгоритма $O(m)$ действий.

Оценим, наконец, объем памяти, требующийся для выполнения алгоритма. Очевидно, каждой вершине и каждому ребру графа соответствует лишь ограниченное число ячеек информации. Поэтому объем памяти оценивается как $O(m)$ ячеек.

9. Замечания. 1. Вместо т.с.в. в алгоритме можно было бы вести текущий список ребер по аналогичным правилам. В этом случае на «выходе» алгоритма получалось бы задание блоков списками принадлежащих им ребер. Некоторое упрощение алгоритма (отсутствие аналога специальных операций для занесения в список блока вершины, на которой он висит) получается за счет того, что каждое ребро принадлежит одному и только одному блоку.

2. В неориентированном графе G разделяющим называется ребро, удаление которого нарушает связность графа. Если из графа G удалить все разделяющие ребра, то компоненты связности полученного графа будут подграфами исходного графа. Эти подграфы называются *бикомпонентами* (листами) графа G. Фактор-граф, полученный из G стягиванием каждой бикомпоненты в точку, называется его *графом Герца*.

Бикомпоненты можно определить иначе как подграфы, натянутые на классы вершин по отношению эквивалентности $S: S(v_1, v_2) \Leftrightarrow$ существует цикл, содержащий v_1 и v_2. (Это определение распространяется на ориентированные графы, с заменой «цикл» на «контур».)

Найти бикомпоненты и разделяющие ребра графа G и построить его граф Герца можно с помощью алгоритма, лишь незначительно отличающе-

тося от предлагаемого в настоящей статье алгоритма выделения блоков, с теми же оценками трудоемкости и объема памяти.

Изменения состоят в следующем. Во-первых, в правиле б неравенство $N(x) \leqslant N(v_{\alpha+1})$ заменяется на $N(x) < N(v_{\alpha+1})$. (Это означает, что в бетрефальный момент мы стираем метку подозрительности у всех ребер, лежащих на образующемся контуре $L'_{\alpha+1}$.) Во-вторых, при нахождении отделимой ветви Q_u, $u=(z, y)$, вершину z не включаем в список очередной бикомпоненты. В-третьих, как разделяющие помечаем не вершины, а ребра, на которых висят отделимые ветви.

Построение графа Герца аналогично построению графа блоков.

Полученный алгоритм близок к алгоритму А. В. Карзанова и обобщается на ориентированный случай так же, как последний (см. [5]).

Поступила в редакцию 23.03.1973
Переработанный вариант 4.07.1973

Цитированная литература

1. *H. Whitney.* Non-separable and planar graphs. Trans. Amer. Math. Soc., 1932, 34, 339–362.
2. *О. Оре.* Теория графов. М., «Наука», 1968.
3. *И. А. Фараджев.* Эффективные алгоритмы решения некоторых задач для ориентированных графов. Ж. вычисл. матем. и матем. физ., 1970, 10, № 4, 1049–1054.
4. *И. А. Фараджев.* Алгоритм выделения бикомпонент ориентированного графа. Тр. Третьей школы по матем. программированию и смежным вопр. Вып. 3. М., ЦЭМИ, 1970, 650–654.
5. *А. В. Карзанов.* Экономный алгоритм нахождения бикомпонент графа. Тр. Третьей школы по матем. программированию и смежным вопр. Вып. 2. М., ЦЭМИ, 1970, 343–347.
6. *К. Берж.* Теория графов и ее применения. М., Изд-во ин. лит., 1962.