**Московский государственный технический университет
имени Н.Э. Баумана**

Факультет «Фундаментальные науки»
Кафедра «Математическое моделирование»

А.Н. Канатников

ДИСКРЕТНАЯ МАТЕМАТИКА

Конспект лекций

Для студентов специальности
«Прикладная математика»

Москва
2006

6. АЛГОРИТМЫ НА ГРАФАХ

6.1. Введение

Определение графа и орграфа. Связь с отношениями. Вершины и ребра (дуги). Изображение и представления графов. Подграфы. Остовные подграфы. Порожденные подграфы. Отношение порядка на подграфах и максимальные подграфы. Связность и компоненты связности. Гомоморфизм и изоморфизм графов. Инварианты. Степени вершин. Группа автоморфизмов графов.

Графом (неориентированным графом) называют пару множеств $(V,\ E)$, где произвольное (конечное) множество, элементы которого называются ***вершинами*** графа, а E — множество неупорядоченных пар (т.е. двухэлементных подмножеств) в множестве V. Элементы множества V называются ***ребрами***. ***Ориентированный граф*** — это пара множеств $(V,\ E)$, где V — множество вершин, а E — множество упорядоченных пар, называемых дугами графа.

Орграф всегда можно интерпретировать как отношение, заданное на множестве вершин, и наоборот, любое отношение имеет интерпретацию как орграф. Неориентированный граф можно рассматривать как специального вида орграф, в котором каждое ребро равнозначно паре взаимно обратных дуг. В этом контексте неориентированный граф связывается с симметричным иррефлексивным отношением: определение неупорядоченной пары как двухэлементного множества не допускает наличия замкнутых ребер, т.е. ребер с одинаковыми концами. В то же время дуги с совпадающими началом и концом — петли — допускаются.

Впрочем, подобные ограничения — всего лишь условность. На практике используются неориентированные графы с петлями, а также графы, в которых две вершины могут быть связаны несколькими ребрами или дугами одного направления (мультиграфы).

Для представления графов используют разные подходы. Например, каждый граф можно описать матрицей, в которой каждая строка соответствует одной вершине, а каждый столбец — одному ребру. Элемент c_{ij} этой матрицы равен 1, если i-я вершина является концом j-го ребра (инцидентна ребру), и 0 в остальных случаях. Аналогично для ориентированного графа c_{ij} принимает значение 1, если i-я вершина является начальной для j-й дуги, значение -1, если i-я вершина является конечной для j-й дуги, и 0 в остальных случаях. Если j-я дуга является петлей, то в этом столбце все элементы равны нулю. Указанная матрица называется ***матрицей инциденций***.

В матрице инциденций орграфа сумма элементов столбца равна нулю. Сумма элементов i-й строки равна разности количества выходящих из вершины дуг и количество входящих в нее дуг. Две эти характеристики называются соответственно ***полустепенью исхода*** и ***полустепенью захода***. Для неориентированного графа сумма элементов столбца равна двум, а сумма элементов строки равна количеству ребер, инцидентных вершине, которое называется ***степенью вершины***.

Обычно матрица инциденций имеет большие размеры и на практике используется редко. Чаще используют другой вариант матричного представления графа. Для неориентированного графа с n вершинами составим квадратную матрицу порядка n, элемент c_{ij} которой равен 1, если i-я и j-я вершины связаны ребром, и 0 в остальных случаях. Для орграфа составляется аналогичная матрица, в которой $c_{ij} = 1$, если есть дуга из i-й вершины в j-ю, и $c_{ij} = 0$ в остальных случаях. Указанная матрица называется ***матрицей смежности***.

В информационных системах представление графа напрямую зависит от поддерживаемых типов данных и способов их представления в электронной памяти. Один из распространенных способов опирается на структуры данных, называемые списками. Список представляет собой

массив произвольно расположенных в памяти элементов, связанных друг с другом посредством ссылок. Различают однонаправленные списки, в которых i-й элемент списка содержит ссылку (адрес) на следующий $(i+1)$-й элемент списка, и двунаправленные списки, в которых каждый элемент содержит ссылки на следующий и предыдущий элементы. Чаще всего элементы списка одинаковы по структуре, но это необязательно.

На основе списков можно использовать следующий подход к представлению орграфа из n вершин. Составляется массив из n элементов, описывающих вершины орграфа. Этот массив называют ***массивом лидеров***, хотя в действительности он может быть списком, а не массивом. Каждый элемент массива лидеров содержит ссылку на список, называемый ***списком смежности***. Каждый элемент списка смежности i-го элемента массива лидеров отражает вершину орграфа, в которую из i-й вершины ведет дуга. Этот способ может использоваться и для описания неориентированных графов.

Пусть $G = (V,\ E)$ — граф (орграф). Граф (орграф) $G_1 = (V_1,\ E_1)$ называется ***подграфом*** графа G, если $V_1 \subset V$ и $E_1 \subset E$. Подграф собственный, если одно из включений строгое. Если $V = V_1$ (множество вершин одно и то же), то G_1 называется ***остовным подграфом*** графа G.

Пусть $G = (V,\ E)$ — граф (орграф) и $V_1 \subset V$. В множестве E выберем множество E_1 всех тех ребер (дуг), оба конца которых попадают в V_1. Пара $(V_1,\ E_1)$ составляет подграф графа (орграфа) G, который называют ***подграфом, порожденным множеством*** V_1.

Пусть G_1 — подграф графа (орграфа) G, а G_2 — подграф G_1. Тогда G_2 будет подграфом и G. В этой ситуации мы называем G_2 подчиненным G_1 и пишем $G_2 \subset G_1$. В результате на множестве всех подграфов G возникает отношение порядка. Имея в виду это отношение, мы для любой совокупности подграфов графа (орграфа) G можем говорить о максимальном подграфе, как о подграфе, который не содержится ни в каком подграфе той же совокупности (кроме самого себя). В ряде случаев понятия „наибольший подграф“ и „максимальный подграф“ совпадают.

Последовательность $v_0, v_1, \ldots, v_n$ вершин графа называется ***цепью***, если для любого $i = \overline{1,\ n}$ вершины v_{i-1} и v_i соединены ребром (являются смежными). Цепь можно интерпретировать и как последовательность ребер, в которой любые два ребра смежные (правда, надо оговорить, что стыковочные вершины данного ребра с левым и правым соседями должны быть разные). Цепь простая, если в ней все вершины разные. Цепь $v_0, v_1, \ldots, v_n$, в которой $v_0 = v_n$ (концы совпадают) называется циклом. Хотя обычно фиксируется начальная точка цикла, удобно считать, что в качестве таковой может быть любая вершина в последовательности, т.е. мы не различаем, например, циклы $v_0, v_1, \ldots, v_n, v_0$ и $v_1, \ldots, v_n, v_0, v_1$. Цикл простой, если в нем все вершины разные. Правда, следует исключить из понятия „простой цикл“ последовательность вида v_0, v_1, v_0, в которой одно и то же ребро проходится дважды.

Для орграфов вводятся аналогичные понятия ***пути*** $v_0, v_1, \ldots, v_n$, в котором для любого $i = \overline{1,\ n}$ из вершины v_{i-1} в вершину v_i ведет дуга, и контура — пути, у которого начало и конец совпадают, а также простого пути и простого контура.

Граф G связный, если любые две его вершины можно соединить цепью. На произвольном графе можно построить ***отношение достижимости***, содержащее все пары вершин, которые можно соединить цепью. Легко показать, что на неориентированном графе это отношение является отношением эквивалентности. Классы эквивалентности по этому отношению называются ***компонентами связности***. Каждая компонента связности представляет собой связный подграф. Связный граф содержит одну компоненту связности, несвязный — несколько. Компоненту связности можно также охарактеризовать как максимальный связный подграф.

Для орграфов используют три варианта понятия связности. Орграф сильно связный, если для любой пары $(v,\ w)$ вершин есть путь из v в w и есть путь из w в v. Орграф связный, если для любой пары $(v,\ w)$ его вершин есть путь из v в w или из w в v. Орграф слабо связный, если ***ассоциированный неориентированный граф***, получающийся заменой каждой дуги ребром, является связным. Из сильной связности вытекает связность, а из последней — слабая связность. Для каждого из трех видов связности можно ввести соответствующее отношение: от-

ношение сильной достижимости, отношение достижимости и отношение слабой достижимости. Из трех отношений два являются отношениями эквивалентности. Классы эквивалентности по отношению сильной достижимости называются компонентами сильной связности или ***бикомпонентами***. Понятие компоненты связности также используют, имея в виду максимальный связный подграф орграфа.

Пусть $G_1 = (V_1,\ E_1)$ и $G_2 = (V_2,\ E_2)$ — два графа. Отображение φ: $V_1 \to V_2$, удовлетворяющее условию $\{v,\ w\} \in E_1 \Rightarrow \{\varphi(v),\ \varphi(w)\} \in E_2$, называется ***гомоморфизмом графов***. Если гомоморфизм G_1 на G_2 устанавливает взаимно однозначное соответствие между парами множеств V_1 и V_2, E_1 и E_2, его называют ***изоморфизмом графов***.

Аналогично вводятся понятия гомоморфизма и изоморфизма орграфов. Изоморфные графы (орграфы) рассматриваются как идентичные: их различия связаны со способом реализации, но не с внутренней структурой.

Установить изоморфизм двух графов сложной структуры непросто: эффективного алгоритма решения такой задачи нет. Конечно, существует конечное число отображений из V_1 в V_2, и можно проверить все эти отображения. Однако на практике это неприемлемо: объем вычислений с увеличением размера графа растет как факториал и очень быстро оказывается неподъемным.

Во многих случаях можно установить неизоморфность двух графов, сравнивая их характеристики, определяемые внутренней структурой, а точнее, такие характеристики, которые совпадают у любых двух изоморфных графов. Такие характеристики называют ***инвариантами***. Первыми инвариантами являются количество вершин и количество ребер. Другим важным инвариантом является ***вектор степеней вершин***, который представляет собой упорядоченный список степеней вершин. Можно привести и другие инварианты: количество петель, максимальную длину простой цепи (пути). Некоторые инварианты будут приведены позже в связи с анализом конкретных задач. Здесь же отметим еще один инвариант. ***автоморфизм*** графа — это изоморфизм графа на себя. Множество всех автоморфизмов графа непусто, так как включает в себя тождественное отображение множества вершин, и относительно операции композиции отображений образует группу. Она называется ***группой автоморфизмов***. У изоморфных графов группы автоморфизмов изоморфны.

6.2. Деревья

Неориентированное дерево. Критерии: n вершин и $n-1$ ребро (при наличии связности; единственность цепи, соединяющей произвольные две вершины. Ориентированное дерево. Остовное дерево (остов). Пример орграфа без остова. Максимальный остовный лес и его построение с помощью алгоритма поиска в глубину. Алгоритм построения всех остовных деревьев графа. Понятие кратчайшего остовного дерева. Алгоритм Краскала и алгоритм Прима. Задача Штейнера.

Связный ациклический (т.е. без циклов) неориентированный граф называется ***неориентированным деревом*** (далее просто деревом).

Орграф называется ***ориентированным деревом*** (ордеревом), если выполняются условия:

1) существует выделенная вершина v_0, имеющая полустепень захода 0 (корень ордерева);
2) все некорневые вершины имеют полустепень захода 1;
3) орграф не имеет контуров (т.е. является ***бесконтурным***).

Рассмотрим произвольную вершину ордерева. Если это не корень, то для нее есть предшествующая вершина, из которой в рассматриваемому вершину ведет дуга. У предшествующей есть свой предшественник. Эта цепочка рано или поздно прервется. Но это может произойти, только если очередная вершина есть корень. Таким образом, в ордереве в любую вершину из корня ведет путь, причем этот путь единственный (неединственность означает, что есть вершины, полустепень захода которых больше единицы).

Среди вершин ордерева есть такие, степень исхода которых равна нулю. Такие вершины называют ***листьями***. Путь, ведущий из корня в лист не может быть дальше продолжен и по отношению включения является максимальным, т.е. не является частью другого пути.

Все вершины ордерева распадаются на уровни. Глубиной вершины называется длина пути в эту вершину из корня. Один уровень составляют вершины одной глубины. Все дуги графа ведут с какого-либо уровня на следующий. Отметим, что в ордереве из n вершин имеется $n-1$ дуга, поскольку каждой вершине можно поставить в соответствие дугу, входящую в эту вершину. Это соответствие распространяется на все вершины кроме корня. Таких вершин $n-1$. Стало быть, и дуг тоже $n-1$.

Перейдем теперь к случаю неориентированного дерева, имеющего n вершин. Зафиксируем одну из вершин v_0 этого дерева, назвав ее ***корнем***. Тогда каждая вершина графа связывается с корнем единственной простой цепью, длина которой представляет собой ее глубину. Единственность цепи из корня в вершину позволяет, как и в случае ордерева, разделить вершины графа на уровни в соответствии с длиной цепи, связывающей вершину с корнем: единственная вершина уровня 0 — корень v_0, вершины уровня 1 — вершины, смежные с v_0, вершины второго уровня — те, которые соединяются с корнем двузвенной цепью и т.д.

Все ребра дерева ведут с одного уровня на предшествующий или последующий уровень. Любая вершина v имеет ребро, соединяющее эту вершину с вершиной предыдущего уровня, и притом единственное. В самом деле, соединим v с корнем v_0 простой цепью. Любое ребро, инцидентное v, есть либо последнее звено указанной цепи, либо ведет на следующий уровень, поскольку увеличивает длину цепи. Припишем каждому ребру направление в сторону увеличения уровня вершины. Тогда дерево превратится в ордерево с корнем v_0.

Теорема 6.1. Следующие условия эквивалентны:

- граф является деревом;
- граф связен и имеет $n-1$ ребро, где n — количество вершин;
- любые две вершины графа можно соединить простой цепью, и притом единственным способом.

◀ Если граф — дерево, то в силу его связности любые две вершины можно соединить цепью. Существование двух различных цепей, связывающих две вершины v и w, равносильно существованию цикла, составленного из этих двух цепей (или их частей, если цепи частично совпадают). Наоборот, любой цикл можно разделить на две непересекающиеся цепи, соединяющие две вершины. Поэтому третье условие равносильно условию, что граф — дерево.

В дереве выберем произвольную вершину в качестве корня. Тогда дерево превращается в ордерево, как это описано выше. В ордереве каждой вершине, кроме корня, можно поставить в соответствие входящую в эту вершину дугу. Такое соответствие оказывается биективным. Следовательно, количество ребер в дереве должно быть $n-1$.

Докажем, что любой связный граф с n вершинами и $n-1$ ребром есть дерево. Доказательство проведем индукцией по количеству вершин. Утверждение очевидно, например, при $n=2$. Пусть утверждение доказано для всех графов с $n-1$ вершиной. Выберем произвольный связный n-вершинный граф с $n-1$ ребром. У этого графа есть вершина v степени 1. Действительно, в противном случае, если все степени вершин не меньше 2, общая сумма степеней не менее $2n$, а значит, у графа не менее n ребер. Удалим из графа вершину степени 1 и единственное ребро, инцидентное этой вершине. Получим граф из $n-1$ вершины и $n-2$ ребер. Этот граф связный, поскольку удаленная вершина не может быть внутренней вершиной простой цепи, а следовательно, цепь соединяющая две вершины, не проходит через удаленную вершину. В соответствии с предположением индукции он есть дерево. Тогда и исходный граф есть дерево, поскольку не имеет циклов: удаляемое ребро с тупиковым концом не может быть звеном какого-либо цикла. ▶

Пусть G — связный граф. Всякий остовный подграф графа G, являющийся деревом, называется ***остовным деревом***, или ***остовом***.

Теорема 6.2. Всякий связный неориентированный граф имеет остов.

◄ Рассмотрим множество всех подграфов графа G, являющихся деревьями. Это множество конечно и упорядочено по включению. Следовательно, оно имеет максимальный элемент. Покажем, что максимальное поддерево в G является остовным подграфом. Пусть T — подграф в G, являющийся деревом и содержащий $k < n$ вершин. Среди вершин графа G, не включенных в T, можно выбрать вершину v, смежную с одной из вершин дерева T. В самом деле достаточно выбрать цепь, первый конец которой находится в T, а второй — за пределами T. Тогда первая вершина, не попадающая в T будет смежна с предшествующей, относящейся к T. Итак, существует вершина v, не попадающая в T, связанная некоторым ребром $\{v, w\}$ с некоторой вершиной из T. Добавив к T вершину v и ребро $\{v, w\}$, мы получим подграф G, не имеющий циклов, так как добавляемое ребро не может быть звеном цикла.

Итак, мы показали, что любой подграф-дерево, содержащий $k < n$ вершин, не может быть максимальным. Значит, максимальный подграф-дерево содержит n вершин и является остовным. ►

Неориентированным лесом называется граф, любая компонента связности которого является деревом. Лес — это просто ациклический граф.

Следствие 6.1. Любой граф имеет максимальный остовный лес.

◄ Построим для каждой компоненты связности остов. Объединение всех этих остовов будет остовным подграфом-лесом данного графа. Предположив, что он не максимальный, мы приходим к выводу, что к нему можно добавить ребро графа G, сохраняющее свойство ацикличности. Но это ребро будет относиться к одной из компонент связности, и его добавление будет означать, что к остову компоненты можно добавить ребро, сохраняя ацикличность. Но это противоречит максимальности остова. ►

Замечание. Нетрудно увидеть, что любой максимальный лес распадается на компоненты, являющиеся остовами компонент графа. В самом деле, рассуждая как и выше можно показать, что максимальный лес является остовным. Выбрав подграф леса, порожденный множеством вершин какой-либо компоненты графа G, получим максимальный лес для этой компоненты. Но нетрудно увидеть, что максимальность обеспечивает связность, т.е. это будет не максимальный лес этой компоненты, а дерево.

Доказательство существования остова у связного графа (остовного леса в общем случае) на самом деле не является конструктивным и непригодно для построения остова. Чтобы построить остов, можно использовать один из способов обхода всех вершин графа, называемый ***поиском в глубину***. Суть его можно уяснить на такой задаче: просмотреть все каталоги и подкаталоги данного жесткого диска. Структура каталогов жесткого диска имеет древовидную структуру. Например, мы хотим определить суммарную емкость диска, занятую файлами. Задачу можно решить так: начинаем просматривать корневой каталог. Если очередной элемент каталога является подкаталогом, переходим в этот подкаталог и начинаем просматривать его, выбирая все файлы и суммируя их длину. Если подкаталог полностью просмотрен, мы поднимаемся в родительский подкаталог и продолжаем его просмотр с того элемента, на котором он был прерван. Процесс просмотра в целом будет завершен, когда мы закончим просмотр корневого каталога.

Описанная стратегия обхода всех вершин дерева и называется поиском в глубину. В действительности требование, чтобы анализируемый граф был деревом, несущественно: поиск в глубину можно реализовать на любом графе.

Опишем более детально поиск в глубину, предполагая, что граф задан массивом лидеров и списками смежности. Алгоритм обхода вершин будет следующий. В процессе работы мы будем использовать пометки в массиве лидеров, выделяющие уже просмотренные вершины, механизм стека для возврата назад. При старте в массиве лидеров все вершины непомеченные, стек пуст.

1. Из массива лидеров выбираем первую непомеченную вершину как текущую. Если таких нет, переходим к пункту p6. Иначе переходим к следующему пункту.

2. Помечаем текущую вершину, выбираем ее список смежности, устанавливаем в нуль указатель в списке смежности и переходим к следующему пункту.

3. Увеличиваем значение указателя на единицу и проверяем не закончился ли список смежности текущей вершины. Если нет, переходим к следующему пункту. Если закончился, переходим к пункту p5.

4. В списке смежности в соответствии с указателем выбираем очередную вершину. Если эта вершина помечена, пропускаем ее и переходим к пункту p3. Если это непомеченная вершина, записываем в стек номер текущей вершины и значение указателя для нее. В качестве текущей устанавливаем выбранную вершину и переходим к пункту p2.

5. Если стек пуст, переходим к пункту p1. Иначе выбираем из стека номер записанной вершины, делая ее текущей, и номер указателя для ее списка смежности. Затем переходим к пункту p3.

6. Процесс заканчиваем.

При проведении этого алгоритма выделяется часть ребер, по которым проводится переход к новой текущей вершине, которые в совокупности с пройденными вершинами образуют подграф анализируемого графа. Отметим, что каждое из таких ребер проходится ровно один раз в прямом направлении (пункт p2) и один раз в обратном (пункт p5). Если учитывать только прямой ход, то каждая вершина будет однозначно связана с одним входным ребром. Исключение составляет стартовая вершина, вход в которую выполняется на основании пункта p1. При повторном возврате к этому пункту все вершины будут помечены, поскольку граф связный. Поэтому количество ребер в подграфе на единицу меньше количества вершин. Следовательно, этот граф — дерево, причем остовное.

Итак, рассмотренный алгоритм поиска в глубину позволил построить остов связного графа. Однако роль алгоритма поиска в глубину на самом деле гораздо выше, чем поиск остова графа. Он является основой для широкого спектра алгоритмов, решающих различные задачи анализа графов.

Связный ориентированный граф может не иметь остовного ордерева хотя бы потому, что у него есть пара вершин с полустепенью захода 0. Тогда и у любого подграфа будет не менее двух вершин с полустепенью захода 0, а это уже противоречит структуре дерева. Можно рассматривать подграфы-ордеревья данного орграфа и среди них есть максимальные. Просто максимальный подграф-дерево не будет остовным. Если мы применим алгоритм поиска в глубину, мы получим не ордерево (из-за пункта p1), а ориентированный лес. Так что утверждение теоремы остается в силе, но в ослабленном варианте.

6.3. Остов графа наименьшего веса

Поскольку остов неориентированного графа определен неоднозначно, возникает задача выбора остова, оптимального в том или ином смысле. Например, граф представляет собой сеть дорог. Возникает задача прокладки маршрута наименьшей длины, проходящего через данные населенные пункты. Математически подобного рода задача ставится следующим образом. Пусть дан размеченный (взвешенный) граф, т.е. граф, каждому ребру которого приписан вес. Требуется выбрать остов наименьшего веса.

Прежде чем рассматривать конкретные алгоритмы, представим себе задачу перечисления всех остовов связного графа. Процедуру можно представить как последовательное наращивание ребер строящегося дерева. Если выбраны k ребер исходного графа, из которых нельзя составить цикла, мы получаем подграф, являющийся неориентированным лесом. К этому лесу можно добавить еще одно ребро из оставшихся, убедившись, что новое ребро не порождает циклов с остальными. Процедура продолжается до тех пор, пока не наберется $n-1$ ребро. В результате

мы получим граф, содержащий $n-1$ ребро и не имеющий циклов. Если это был бы лес, то количество ребер в нем было бы равно количеству вершин минус количество компонент. Количество $n-1$ может получиться только в том случае, когда подграф остовный и и имеет одну компоненту связности, т.е. является деревом.

На каждом шаге наращивания поддерева существует несколько возможностей добавления очередного ребра. В результате на каждом шаге наше решение разветвляется, и мы фактически получаем так называемое дерево решений. Листья этого дерева, т.е. вершины степени 1, представляют собой конкретные решения. Совокупность всех листьев есть совокупность всех остовов рассматриваемого графа.

Поиск остова наименьшего веса можно представить как оптимальный спуск из корня дерева остовов в лист. На каждом шаге следует принимать решение, какое ребро из допустимых следует выбрать. Особенность этой задачи состоит в том, что оптимальный выбор ребра на каждом шаге гарантирует оптимальность окончательного результата.

В этой процедуре удобно считать, что наращиваемый подграф изначально включает все вершины, а пошаговая процедура состоит в последовательной склейке компонент подграфа-леса с помощью ребер исходного графа. Тогда стартовый граф представляет собой n изолированных вершин. На каждом шаге процесса наращивания добавляется одно ребро и количество компонент уменьшается на одну. На конечном $(n-1)$-м шаге количество добавленных ребер станет $n-1$, а количество компонент сократится до одной.

Рассмотрим два непересекающихся подграфа $T_1 = (V_1,\ E_1)$ и $T_2 = (V_2,\ E_2)$ графа G. Введем для них характеристику

$$\Delta = \Delta(T_1, T_2) = \min_{v_1, v_2} c(v_1; v_2),$$

где $c(v_1; v_2)$ — вес ребра $\{v_1,\ v_2\}$, а минимум берется по всем тем ребрам $\{x,\ y\}$ графа G, для которых $v_1 \in V_1$, $v_2 \in V_2$.

Теорема 6.3. Пусть подграф-лес T, состоит из компонент T_1, T_2, …, T_k и подчинен некоторому остову наименьшего веса. Из компонент T_2, T_3, …, T_k выберем такую компоненту T_j, на которой $\Delta(T_1, T_j)$ минимально, причем эта величина реализуется на ребре γ^*, соединяющем некоторую вершину компоненты T_1 с некоторой вершиной компоненты T_j. Тогда подграф „T плюс γ^*“ подчинен некоторому остову наименьшего веса.

◀ Рассмотрим остовное дерево T' наименьшего веса, содержащее T. В T' есть цепь α, соединяющая концы ребра γ^*. Если само это ребро не входит в T', то его добавление к графу дает цикл $\gamma + \alpha$. В цепи α есть первое ребро γ, выходящее за пределы компоненты T_1 (начиная с того конца цепи, который находится в T_1). По условию выбора вес ребра γ не меньше веса ребра γ^*. Удаление ребра γ разрывает образовавшийся цикл и сохраняет связность T'. Следовательно, замена γ на γ^* приводит к остовному дереву T'', отличающемуся от T' всего одним ребром. Поскольку $c(\gamma^*) \leqslant c(\gamma)$, вес остова T'' не превышает вес остова T' и является наименьшим, т.е. T'' — остов наименьшего веса. ▶

Доказанная теорема является теоретической основой нескольких алгоритмов выбора остова наименьшего веса. Отметим, что принцип выбора, сформулированный в теореме, обладает некоторой свободой: в качестве компоненты T_1 можно выбирать любую из имеющихся. Произвольность такого выбора означает, что и конкретный алгоритм построения остова можно реализовать по-разному. Рассмотрим два таких алгоритма.

Алгоритм Краскала. В этом алгоритме минимизация выполняется по обоим компонентам T_1 и T_j. На самом деле нет нужды перебирать всевозможные пары компонент и определять на какой паре величина Δ минимальна. Можно просто перебрать все ребра графа и среди них выбрать ребро наименьшего веса, которое, во-первых, не входит в T, а во-вторых, не образует цикла с уже имеющимися в T ребрами. Тогда это будет ребро, соединяющее какие-то две компоненты графа T.

Кратко алгоритм Краскала можно описать так.

1. Начинаем с остовного графа T без ребер.

2. Все ребра графа упорядочиваем по возрастанию веса.

3. Выбираем очередное ребро из упорядоченного списка и проверяем, образует ли оно цикл с уже отобранными. Если да, пропускаем его и повторяем пункт. Если нет, переходим к следующему пункту.

4. Проверяем, если отобрано $n-1$ ребро, то процесс прекращаем. Иначе возвращаемся к предыдущему пункту.

В изложенной стратегии два узких места: сортировка ребер по возрастанию и проверка на появление циклов при добавлении очередного ребра. Первая проблема сводится к выбору эффективного алгоритма сортировки. Отметим, что полная сортировка всех ребер на самом деле не нужна. Отбираться будет $n-1$ ребро, в то время как полный граф содержит гораздо больше ребер: $n(n-1)/2$. Сортировку можно свести к выбору на каждом шаге наилучшего из оставшихся ребер. Эта проблема показывает, что алгоритм Краскала наиболее эффективен для графов с небольшим числом ребер (того же порядка, что и число вершин).

Вторая проблема может решаться с помощью специальной системы пометок вершин графа. Поскольку добавляемое ребро должно связывать вершины разных компонент остовного подграфа T, то и проверять надо, принадлежат ли концы ребра разным компонентам. Достаточно каждой вершине приписать специальный ранг, указывающий на компоненту, к которой относится вершина. При выборе очередного ребра отбрасываем варианты, у которых концы имеют одинаковый ранг. Если выбрано ребро, у которого концы имеют ранги r_1 и r_2, то надо просмотреть все вершины и поменять все ранги r_2 на r_1 (или наоборот). Тогда слияние двух компонент в одну будет отражаться в рангах вершин.

Указанную процедуру проверки на циклы можно сделать более эффективной, если в пометку вершин включать не только ранг, но и дополнительную информацию, позволяющую по вершине восстановить все поддерево.

Алгоритм Прима. В этом алгоритме дерево T_1 фиксировано, остальные компоненты до конца остаются одиночными вершинами. Технически эту стратегию можно реализовать следующим образом. Каждой вершине x на текущем шаге соответствует метка, содержащая вершину y дерева T_1, ближайшую к x, и вес β ребра, соединяющего y с x. Если вершина x не имеет ребер, связывающих ее с T_1, тот ей приписывается специальная метка $(0,\ \infty)$. На k-м шаге просматриваются все вершины, не относящиеся к T_1 и среди них выбирается вершина x_k, для которой вес β_k наименьший. Ребро, соединяющее x_k с T_1, заносится в список отобранных ребер. Затем для всех вершин $x \notin T_1$ вес β сравнивается с весом ребра $\{x_k,\ x\}$ и если последнее меньше, метка вершины x обновляется. Процедура останавливается, когда будет отобрано $n-1$ ребер или n вершин.

6.4. Задача о путях в размеченном графе

Отношение достижимости и матрица достижимости. Граф как отношение (отношение смежности) и транзитивное замыкание. Понятие о размеченном графе. Сведение задачи к вычислению итерации матрицы в полукольце. Алгоритм Дейкстры. Алгоритм Флойда.

Граф $G = (V,\ E)$, у которого каждой вершине приписан вес, называется ***размеченным графом***. К размеченным графам может относиться сеть дорог, каждая из которых имеет оценку качества и протяженности. Для такой сети актуальна задача поиска оптимального маршрута. Размеченный граф может обозначать финансовые потоки между различными составляющими бизнес--процесса, информационные потоки в компьютерной сети и т.п.

Задача о путях в размеченном графе может формулироваться в трех вариантах:

- найти кратчайшую цепь, связывающую две заданные вершины графа;
- для заданной вершины v_0 построить кратчайшие цепи, соединяющие v_0 со всеми остальными вершинами графа;

• построить кратчайшие цепи для всех пар вершин графа.

Задача может ставиться как для неориентированных, так и для ориентированных графов. Под весом цикла (пути) понимается сумма весов всех ребер (дуг), входящих в цепь (путь). Однако важно то, что понятие „сумма“ может меняться. Фактически весами в графе могут быть элементы любого полукольца или кольца. Например, как частный случай поставленной задачи можно рассматривать задачу построения матрицы достижимости для данного графа. Достаточно разметить граф над полукольцом $\mathbb{B}$, причем под суммой понимать произведение в $\mathbb{B}$. Тогда вес каждой цепи будет равен 1, а матрица весов кратчайших путей будет совпадать с матрицей достижимости.

Требование, чтобы веса графа были элементами полукольца, вообще говоря, не является обязательным. Достаточно обеспечить алгоритм, позволяющий по весам звеньев цепи однозначно определить вес всей цепи.

В результате решения общей задачи мы для каждой пары вершин v и w получаем значение наименьшего веса цепи (пути), соединяющей v и w, которое называется ***стоимостью***. Из стоимостей можно составить ***матрицу стоимостей***. Процесс решения задачи состоит в вычислении матрицы стоимостей.

Отметим, что наличие в графе, размеченном над кольцом $\mathbb{R}$, циклов (контуров) отрицательного веса может привести к тому, что задача не будет иметь решения. В этом случае необходимо исключить из рассмотрения такие циклы или по крайней мере ограничить число проходов по ним.

Решение первого и второго вариантов задачи как часть входит в решение третьего варианта. Выделение их как самостоятельных преследует цель выбора специальных более экономичных алгоритмов для их решения.

Остановимся на первом варианте задачи.

Алгоритм Дейкстры. Этот алгоритм рассчитан на случай, когда в графе все веса неотрицательные. Он позволяет для заданной вершины v_0 ориентированного графа построить стоимости $D[v]$ для каждой пары вершин v_0 и v. Алгоритм сводится к построению последовательности $S_1, S_2, \ldots, S_n$ множеств вершин, в которой множество S_{k+1} получается из S_k добавлением одной вершины, и вычислению для каждого множества S_k множества $D_k[v]$ стоимостей, определяемых только по тем путям из v_0 в v, которые не выходят за пределы множества S_k (за исключением конечной вершины v). Начальное множество S_1 состоит из единственной вершины v_0, а массив стоимостей $D_1[v]$ — из весов дуг, связывающих вершину v_0 с вершиной v. При отсутствии дуги, связывающей v_0 с v, полагаем $D_1[v] = \infty$. На каждом шаге алгоритма к множеству S_k добавляется вершина $w = v_{k+1} \notin S_k$, для которой значение $D_k[w]$ является наименьшим. Затем вычисляется массив $D_{k+1}[v]$ с помощью следующей процедуры пересчета. Если $v \in S_{k+1}$, то $D_{k+1}[v] = D_k[v]$. Если же $v \notin S_{k+1}$, то в качестве $D_{k+1}[v]$ выбирается наименьшее из значений $D_k[v]$ и $D_k[v_{k+1}] + \omega(v_{k+1}, v])$, где $\omega(v, w)$ — вес дуги, соединяющей вершины v и w.

Через n шагов алгоритма Дейкстры множество S_k будет содержать все вершины графа, а массив $D_k[v]$ — стоимости пар вершин v_0, v. При реализации на практике массивы $D_k[v]$ можно совместить. При этом на k-м шаге общий массив $D[v]$ будет содержать значения $D_k[v]$. При пересчете значения $D[v]$ для вершин $v \in S_{k+1}$ не изменяются, пересчитываются лишь те, для которых $v \notin S_{k+1}$.

Как видим, алгоритм Дейкстры ориентирован на решение второго варианта задачи о путях. Однако его легко модифицировать для решения и первого варианта задачи: достаточно процесс остановить в тот момент, когда в качестве вершины v_{k+1} будет выбрана заданная конечная вершина пути.

Почему алгоритм Дейкстры приводит к кратчайшему пути. Выбор на каждом шаге точки V_{k+1}, путь к которой (из обследованных в рамках множества S_k) является кратчайшим, позволяет сразу объявить этот путь кратчайшим и во всем графе. Это вытекает из следующих соображений. Пусть на k-м шаге в вершину w ведет кратчайший путь веса c^*, т.е. $D_k[w] = c^*$

и минимальна для всех вершин вне S_k (рис. 6.1). Рассмотрим другой путь, ведущий в w через некоторую вершину v. По условиям выбора $c = D_k[v] \geqslant c^*$. Учитывая, что часть пути из v в w имеет неотрицательный вес, заключаем, что вес альтернативного пути не меньше, чем тот, по которому был вычислен вес $D_k[v]$. Как видим, явно используется условие неотрицательности весов графа. Отметим, что это условие автоматически выполнено, если граф размечен над идемпотентным полукольцом. Можно также условие неотрицательности весов заменить условиекм монотонности: вес любого пути не должен быть меньше веса любой своей части.

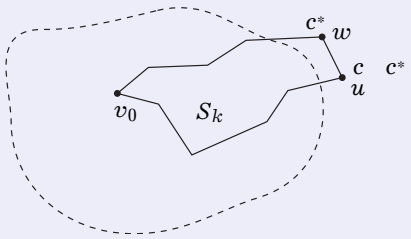

Рис. 6.1

Алгоритм Дейкстры по своей сути — процедура перебора путей в графе для отбора среди них оптимальных. Эту процедуру легко модифицировать так, чтобы можно было получать не только веса оптимальных путей, но и сами эти пути, т.е. пути наименьшего веса. Достаточно в качестве значений массива $D[v]$ рассматривать не стоимости пар вершин, а векторы, первой компонентой которых является вес оптимального пути (стоимость), а остальные компоненты описывают цепочку вершин, представляющую собой оптимальный путь из вершины v_0 в вершину v. На k-м шаге при вычислении $D_{k+1}[v]$ мы либо сохраняем вектор $D_k[v]$, либо к первой компоненте вектора $D_k[v_{k+1}]$ добавляем вес $\omega(v_{k+1}, v])$, а к концу этого вектора присоединяем дугу, соединяющую вершины v_{k+1} и v.

Задача о путях в полукольцах. Во многих случаях в задаче о путях можно считать, что метки дуг (ребер) являются элементами некоторого полукольца. Выбор полукольца позволяет соответствующим образом интерпретировать понятие оптимальности пути.

Считаем, что весом пути является произведение (в полукольце) всех весов дуг, составляющих этот путь, а стоимостью пар вершин — сумма весов всех путей, соединяющих эти вершины. Рассмотрим задачу вычисления матрицы стоимости.

Обозначим через Ω матрицу весов графа. Тогда с учетом правила перемножения матриц заключаем. что Ω^2 содержит стоимости пар вершин, вычисленные по всем путям длины 2, аналогично степень Ω^k содержит стоимости пар вершин, вычисленные по всем путям длины k. Поскольку стоимость пары вершин определяется как сумма весов по всем путям, соединяющим эти вершины, заключаем, что матрица стоимости C определяется формулой

$$C = \Omega^* = E + \Omega + \Omega^2 + \dots$$

Здесь Ω^* — итерация матрицы в рассматриваемом полукольце. Отметим, что в силу конечности графа любой путь длины больше n содержит цикл (контур) и содержит в себе как часть другой путь. Вес пути — произведение всех дуг. Следовательно, часть пути имеет больший вес и при суммировании поглощается. Это значит, что $\Omega^{n+1} \leqslant \Omega^n$. Поэтому

$$C = \Omega^* = E + \Omega + \Omega^2 + \dots \Omega^n.$$

Если граф размечен обычными числовыми весами, достаточно в качестве умножения полукольца рассмотреть сложение действительных чисел, а в качестве сложения полукольца — минимум двух действительных чисел. Чтобы эта операция имела нейтральный элемент, достаточно ввести в рассмотрение вес ∞, обозначая им пары вершин, не связанные дугой графа. Вычисление минимума — коммутативная идемпотентная операция, относительно которой сложение дистрибутивно.

Анализ процедуры непосредственного вычисления Ω^* показывает, что алгоритм сводится к методу динамического программирования.

Из теории полуколец вытекает, что итерация A^* матрицы A может быть получена как решение матричного уравнения $X = AX + E$. Вычисление отдельного j-го столбца x_j матрицы X сводится к матричному уравнению $x_j = Ax_j + e_j$, где e_j — j-й столбец единичной матрицы. А это матричное уравнение может интерпретироваться как матричная форма записи системы линейных уравнений.

Таким образом, вычисление матрицы стоимости сводится к решению нескольких систем линейных уравнений в некотором полукольце. В данном случае можно ограничиться вычислением одного столбца матрицы стоимостей, что равносильно вычислению массива $D[v]$ стоимостей для пар вершин v и v_0. Это можно реализовать как решение одной системы линейных уравнений над полукольцом.

Вычисление отдельных элементов матрицы Ω^* можно свести к решению системы линейных уравнений над полукольцом. Решение такой системы осуществляется методом Гаусса (методом исключения неизвестных). Поскольку в конкретном рассматриваемом приложении требуется лишь один элемент матрицы Ω^* (путь наименьшего веса, связывающий две данные вершины), то процедуру вычисления можно провести так, что в методе Гаусса будет использоваться только прямой ход (в системе нужно будет исключать последовательно все неизвестные, кроме того, которое определяется).

Прямое вычисление итерации матрицы весов (например, в случае, когда сложение — это минимум, а умножение — это обычное сложение) можно представить как последовательный пересчет матрицы весов по формуле

$$c'_{ij} = \min\left\{c_{ij},\ \min_{k=\overline{1,n}}(c_{ik} + w_{kj})\right\}, \tag{6.1}$$

где c_{ij} — найденные стоимости, а w_{ij} — веса ребер. С одной стороны, видно, что это позволяет сравнить уже найденный минимальный маршрут длины m с маршрутом, проходящим через вершину k и тем самым получить минимальный маршрут длины $m + 1$. С другой стороны, в терминах полукольца указанный пересчет записывается так:

$$c'_{ij} = c_{ij} + \sum_{k=1}^{n} c_{ik}w_{kj},$$

что в матричной форме равносильно равенству $C' = C + CW$. В итерационной процедуре $C_0 = E$, $C_{k+1} = C_k + C_kW$ мы через n шагов получим итерацию W^* матрицы весов.

Описанный процесс вычисления итерации можно улучшать в разных направлениях. Нетрудно заметить, что итерацию можно получить по итерационной формуле $C_{k+1} = E + C_kW$. Правда, с точки зрения объема вычислений это не дает преимуществ. Преимущество дает такой процесс: $C_1 = E + W$, $C_2 = C_1^2$, $C_4 = C_2^2$, $C_8 = C_4^2$ и т.д. Для вычисления итерации W^* вместо n произведений достаточно вычислить $\log_2 n$ квадратов.

Еще один прием связан с нарушением структуры матричных произведений (все матричные многочлены вычисляются параллельно). Оказывается, что для построения матрицы стоимостей в (6.1) можно не искать минимум по всем вершинам (минимум по k). Достаточно на каждом шаге использовать фиксированную вершину, меняя фиксированную вершину от шага к шагу. Точнее, на k-м шаге мы вычисляем

$$c_{ij}^{(k)} = \min\left\{c_{ij}^{(k-1)},\ c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\right\}, \tag{6.2}$$

где $C^{(0)} = E$. Через n шагов мы получим матрицу $C^{(n)}$, которая и будет являться матрицей стоимостей. Изложенный алгоритм был предложен Р. Флойдом.

Покажем, что алгоритм Флойда действительно приводит к минимальным весам маршрутов между любой парой вершин. Отметим, что на k-м шаге матрица C^k будет содержать для каждой пары вершин минимальные веса, получаемые по маршрутам, проходящим по первым k вершинам графа. Доказательство можно строить методом индукции. На старте в качестве $C^{(0)}$ выбираем матрицу E, имея в виду нуль и единицу соответствующего полукольца, т.е. в данном случае $c_{ij}^{(0()} = 0$ при $i = j$ и $c_{ij}^{(0()} = \infty$ при $i \neq j$. Очевидно, что эта матрица дает веса оптимальных маршрутов длины 0. Предположим, что матрица $C^{(k-1)}$ содержит веса оптимальных маршрутов, пролегающих по вершинам с номерами до $k-1$ включительно. Тогда формула (6.2) для вершин с номерами i и j выбирает наилучший маршрут среди „старых“ веса $c_{ij}^{(k-1)}$ и новых с дополнительной вершиной с номером k. В самом деле, наилучший маршрут, проходящий через k-ю вершину может состоять из двух оптимальных маршрутов, первый из i-й вершины в k-ю, второй из k-й вершины в j-ю. Через n шагов получим для каждой пары вершин вес оптимального маршрута без ограничений на включаемые вершины.

6.5. Циклы, разрезы и задача Эйлера

Эйлеровы цепи и циклы. Циклы в графе. Фундаментальные циклы, числа Бетти. Разрезы. Критерий существования. Приложения.

Эйлеровы графы. Обнаружение замкнутых цепей, и в частности циклов — важная задача в исследовании графов. Во-первых, циклы в графе — существенная характеристика структуры этого графа. Во-вторых, наличие циклов влияет на многие алгоритмы на графах. Например, циклы отрицательного веса могут привести к тому, что задача о кратчайших путях не будет иметь решения.

Одной из задач теории графов, фактически положившей начало этой теории, является задача обнаружения ***эйлеровых циклов*** и ***эйлеровых цепей***. Эйлеров цикл — это закнутая цепь, которая проходит через каждое ребро графа и притом один раз. Эйлеров цикл может не быть циклом в нашем понимании, так как некоторые вершины могут проходится несколько раз. В этом случае мы введем в обиход термин ***квазицикл***, т.е. любая замкнутая цепь, в которой ребра не повторяются. Отметим, что вершины, не попадающие в эйлеров цикл, являются изолированными. Мы этот случай рассматривать не будем и дополним определение эйлерова цикла требованием, что каждая вершина попадает в эйлеров цикл.

Понятие эйлерова цикла имеет прямое отношение к задачам прорисовки фигур без отрыва карандаша от бумаги. Другой исторической задачей является ***задача о Кенигсбергских мостах***, изучавшаяся Л. Эйлером и приведшая к понятию эйлерова цикла. Задача состояла в обходе Кенигсберга с условием, что каждый мост проходится ровно один раз (рис. 6.2). Рассматривая разделенные участки суши как вершины графа, а соединяющие эти участки мосты — как ребра (рис. 6.3), приходим к задаче поиска такой цепи в графе, которая содержит каждое ребро, и притом один раз, т.е. к задаче поиска эйлеровой цепи.

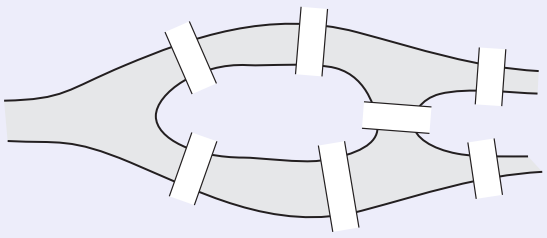

Рис. 6.2

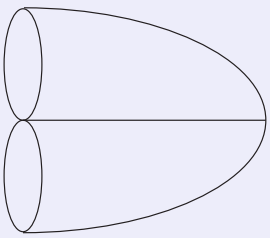

Рис. 6.3

Любой квазицикл можно разделить на несколько циклов, возможно, имеющих общие вершины, но не имеющих общих ребер. Действительно, начав перемещение по квазициклу мы получим последовательность вершин $a_1, a_2, \ldots, a_k, a_i$, где $i < k$. Поскольку в цепи ребра не

повторяются, последовательность вершин a_i, a_{i+1}, ..., a_k есть цикл. Выделим этот цикл, а маршрут движения сократим: a_1, a_2, ..., a_{i-1}, a_i, a_k. Повторяя эту процедуру и учитывая, что количество вершин в цикле конечно, мы придем к набору циклов с общими вершинами, но без общих ребер.

Теорема 6.4. Граф G имеет эйлеров цикл тогда и только тогда, когда он связен и любая его вершина имеет четную степень.

◄ Отметим, что оба отмеченных условия (связность и четность степеней) необходимы для существования эйлерова цикла, поскольку при движении по циклу мы попадаем в каждую вершину и выходим. В результате, во-первых, цикл соединяет все вершины, что обеспечивает связность графа. А во-вторых, для каждой вершины инцидентные ребра разделяются на пары, в каждой из которых одно ребро входящее, а другое выходящее. Поэтому степени всех вершин четные.

Предположим, что граф связен, а все вершины имеют четную степень. Выберем произвольную вершину v_0 и начнем движение по графу по любому инцидентному ребру. Это движение не может закончиться в вершине, отличной от стартовой, поскольку четная степень вершины гарантирует, что если мы пришли в нее, то есть хотя бы одно не использовавшееся ребро. Поэтому в такой вершине цепь можно продолжить, обеспечивая неповторяемость ребер. В то же время в силу конечности множества всех ребер цепь в какой-то момент должна закончиться. Это может произойти только в тот момент, когда мы вернемся в стартовую вершину и не окажется ни одного инцидентного ребра, которое еще не использовалось.

Обозначим полученный квазицикл γ_1. Предположим, что в этот квазицикл входят не все ребра графа. Тогда должно быть хотя бы одно ребро, не вошедшее в квазицикл γ_1, инцидентное одной из вершин v_1 этого квазицикла (в противном случае мы имеет несвязный граф). Отметим, что после удаления квазицикла γ_1 все вершины графа будут иметь четную степень. Начинаем движение из вершины v_1 по ребрам, не вошедшим в γ_1. Рассуждая как и выше, заключаем, что движение закончится в вершине v_1, и мы получаем квазицикл γ_2, не имеющий с γ_1 общих ребер, но имеющий с ним общую вершину v_1. Эти два квазицикла можно объединить в один квазицикл γ_2' (рис. 6.4). Если объединенный квазицикл охватывает не все ребра графа, мы опять можем выбрать вершину v_3 этого квазицикла, у которой есть незадействованные инцидентные ребра, и, начиная движение из вершины v_3, построить следующий квазицикл γ_3. Присоединяем квазицикл γ_3 к γ_2'. Этот процесс продолжаем далее. Он завершится тогда, когда не останется незадействованных ребер. В результате будет получен квазицикл, содержащий все ребра графа, т.е. эйлеров цикл. ►

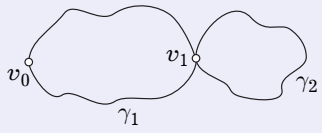

Рис. 6.4

Следствие 6.2. Для того чтобы граф G имел эйлерову цепь, необходимо и достаточно, чтобы он был связен, две его вершины имели нечетную степень, а остальные — четную.

◄ Пусть вершины v_1 и v_2 имеют нечетную степень. Если эти вершины несмежные, добавим к графу ребро γ, соединяющее v_1 и v_2. Получим связный граф, у которого все вершины имеют четную степень. Такой граф имеет эйлеров цикл. Удалив из эйлерова цикла ребро γ, получим эйлерову цепь исходного графа.

Если вершины v_1 и v_2 смежные, мы можем добавить к графу еще одну вершину v', соединив ее с вершинами v_1 и v_2. Опять в расширенном графе существует эйлеров цикл, который после удаления дополнительной вершины и двух ребер становится эйлеровой цепью исходного графа. ►

Фундаментальные циклы. Отметим две характеристики графа, условно противоположные друг другу: несвязность и существование циклов. Для связного графа количество ребер, при которых могут отсутствовать циклы, ограничено ($n-1$ ребро). При увеличении количества ребер появляются циклы, при уменьшении — теряется связность.

Пусть граф G связный. Выберем в нем остовное дерево T. Добавление к T одного ребра графа G приводит к появлению одного цикла. В самом деле, если добавляемое ребро γ входит в два цикла, то объединяя циклы, но уже без γ, получим цикл в дереве T (рис. 6.5). Поскольку дерево не имеет циклов, то и предположение о появлении в связи с γ двух циклов оказывается ложным.

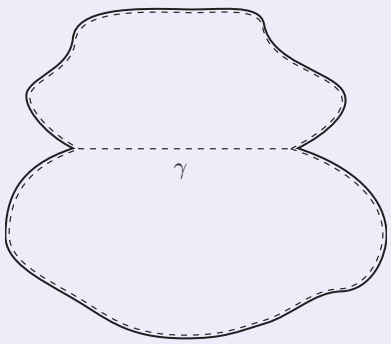

Рис. 6.5

Итак, выбор в графе G остовного дерева разделяет все ребра графа на две группы: древесные ребра, вошедшие в остов, и обратные ребра, не вошедшие в остов. С каждым обратным ребром графа связан вполне определенный цикл графа G. Совокупность этих циклов образует ***фундаментальную систему циклов***. Основным ее свойством является то, что с помощью фундаментальной системы циклов можно построить любой цикл графа, в то время как ни один из циклов фундаментальной системы не выражается через другие. Это похоже на ситуацию в линейном пространстве при рассмотрении понятия базиса. Отметим, что ассоциация с линейной алгеброй оказывается весьма содержательной и не является поверхностной.

Любую цепь в графе без повторений ребер (полупростую цепь) можно интерпретировать как некоторой множество ребер этого графа. Если из данной совокупности ребер графа можно сложить полупростую цепь, то такая цепь единственна (если, конечно, не различать цепи из одних и тех же ребер, но с противоположной ориентацией). Другими словами, множество всех цепей графа можно рассматривать как подмножество булеана множества всех ребер графа G. Булеан множестува ребер с операцией симметрической разности представляет собой абелеву группу, причем в этой группе каждый элемент является противоположным самому себе. Такая абелева группа может рассматриваться как линейное пространство над полем $\mathbb{Z}_2$. В результате мы приходим к понятию ***пространства цепей графа***. В этом линейном пространстве рассмотрим линейную оболочку всех циклов. Получим ***пространство циклов графа***. Наша цель — показать, что фундаментальная система циклов есть базис в пространстве циклов графа.

Отметим, что пространство циклов содержит все квазициклы, так как каждый квазицикл есть сумма нескольких циклов.

Теорема 6.5. Сумма двух квазициклов есть либо пустое множество, либо квазицикл, либо объединение нескольких попарно непересекающихся квазициклов.

◀ Рассмотрим общие вершины двух циклов. В сумме степеней каждой вершины, относящихся к каждому из квазициклов, общие ребра считаются дважды. Поэтому после удаления общих ребер, т.е в результате суммирования, мы из двух квазициклов получим граф, у которого все вершины имеют четные степени. Это означает, что каждая компонента связности такого графа есть эйлеров граф, или квазицикл. Значит, сумма двух квазициклов есть объединение не пересекающихся квазициклов. ▶

Теорема 6.6. Фундаментальная система циклов, порожденная выбранным остовом, является базисом в пространстве циклов связного графа.

◀ Фундаментальная система циклов линейно независима в пространстве циклов. В самом деле, выберем и приравняем нулю произвольную линейную комбинацию $\alpha_1 c_1 + \alpha_2 c_2 + \ldots + \alpha_k c_k$ циклов c_1, c_2, …, c_k с коэффициентами $\alpha_i \in \mathbb{Z}_2$. Цикл c_1 имеет обратное ребро r_1, которое не входит ни в какой другой цикл. Значит, равенство нулю линейной комбинации возможно только при $\alpha_1 = 0$. Повторяя рассуждения для всех остальных циклов линейной комбинации, заключаем, что все коэффициенты линейной комбинации равны нулю.

Пусть c_1, c_2, …, c_k — фундаментальная система циклов, причем каждый цикл c_i связан с обратным ребром γ_i. Выберем произвольный цикл c. Пусть этот цикл содержит обратные ребра r_{i_1}, r_{i_2}, …, r_{i_l}. Тогда множество $c \triangle c_{i_1} \triangle c_{i_2} \triangle \ldots \triangle c_{i_l}$ не содержит обратных ребер, а следовательно, состоит из ребер, принадлежащих остову. Это множество как сумма нескольких циклов, либо пусто, либо является объединением некоторого набора не пересекающихся квазициклов, а значит, набора циклов, не имеющих общих ребер. Однако в остовном дереве вообще нет циклов. Поэтому указанная сумма есть пустое множество, т.е.

$$c \triangle c_{i_1} \triangle c_{i_2} \triangle \ldots \triangle c_{i_l} = \varnothing.$$

Это равенство эквивалентно равенству

$$c = c_{i_1} \triangle c_{i_2} \triangle \ldots \triangle c_{i_l},$$

поскольку в пространстве циклов каждый элемент является противоположным самому себе.

Таким образом, любой цикл графа является линейной комбинацией фундаментальной системы циклов. Следовательно, фундаментальная система циклов — базис в пространстве циклов графа. ▶

Доказанная теорема позволяет определить размерность пространства циклов по числу обратных ребер. Если n — число вершин связного графа, m — число его ребер, то размерность пространства циклов равна $m - n + 1$.

Понятие пространства циклов можно ввести и для несвязного графа. В этом случае любой цикл (квазицикл) есть цикл (квазицикл) одной из компонент. Остов графа будет лесом, но при этом сохраняется свойство, что любое обратное ребро порождает ровно один цикл. Таким образом, пространство циклов имеет размерность $m-n+p$, где p — число компонент связности.

Размерность пространства циклов, равная $c(G) = m - n + p$, называется ***цикломатическим числом графа***. Число $\nu(G) = n - p$, равное числу ребер в остове графа, называется ***коцикломатическим числом***. Обе эти характеристики — инварианты, которые могут использоваться при анализе графов на изоморфность.

Отметим, что не любой базис пространства циклов можно получить исходя из некоторого остова этого графа. Фундаментальная система циклов обладает следующим свойством: каждый цикл фундаментальной системы имеет ребро, при удалении которого разрывается только один цикл. Для графа на рис. 6.6 цикломатическое число равно 4. Нетрудно увидеть четыре независимых цикла этого графа: $1-2-3$, $2-4-5$, $2-3-5$, $3-5-6$. Сумма двух, трех или всех четырех циклов не может быть равна нулю, поскольку у трех циклов из четырех есть „персональные“ ребра. В то же время у внутреннего цикла ребра таковы, что удаление любого из них разрывает сразу два цикла из четырех. Это означает, что рассмотренная система циклов не порождается каким-либо остовом.

Для построения фундаментальной системы циклов следует выполнить следующие шаги.

1. построить остов графа (дерево или лес);
2. разделить все ребра графа на древесные и обратные;
3. для каждого обратного ребра построить соответствующий ему цикл. Для этого в остове необходимо найти единственную цепь, соединяющую концы ребра, и к этой цепи добавить само ребро.

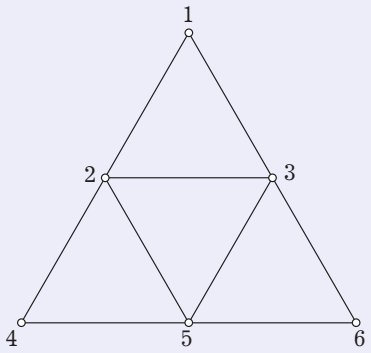

Рис. 6.6

Фундаментальные разрезы. Разделим вершины графа произвольным образом на две группы и выберем в графе множество тех ребер, которые соединяют вершины разных групп. Это множество может быть пустым (это произойдет в том случае, если каждая компонента связности целиком попадает в одну из групп). Если же оно не пустое, то удаление ребер множества из графа увеличивает его связность. Описанное множество называется ***разрезом графа***.

Изучение разрезов позволяет оценить важные свойства графа. Например, анализ разрезов графа городских дорог позволяет найти узкие места в организации транспортных потоков и найти выверенные решения по развитию сети дорог.

Вообще, под разрезом можно было бы понимать любое множество ребер графа, удаление которых повышает связность этого графа. Однако такое определение не эквивалентно предыдущему, оно является расширением понятия разреза. В самом деле, в графе на рис. 6.6 выберем множество из трех ребер $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$. Удаление этих ребер делает граф несвязным (вершина 1 становится изолированной). Однако не существует такого разделения вершин на две группы, при котором концы все трех ребер будут принадлежать разным группам. Множество ребер, удаление которых повышает связность графа, будем называть ***разделяющим множеством***. Разрез — частный случай разделяющего множества.

Множество всех разрезов графа можно рассматривать как полугруппу с операцией объединения (точнее, совокупность всех разрезов в реберном булеане графа с операцией объединения порождает некоторую подполугруппу). Поскольку объединение — идемпотентная операция, ***алгебра разрезов*** есть полурешетка, отношение порядка которой совпадает с отношением включения множеств. Интересна база этой алгебры, т.е. минимальное множество разрезов, порождающее всю алгебру разрезов. Естественно к базе разрезов относить только минимальные разрезы, т.е. такие разрезы, которые не содержат подмножеств, также являющихся разрезами (такие разрезы также называют ***простыми разрезами***).

Условие минимальности позволяет стереть границу между разрезом и разделяющим множеством: минимальное разделяющее множество оказывается разрезом. Отметим, что минимальное разделяющее множество состоит из ребер лишь одной компоненты связности графа. Увеличение связности означает, что хотя бы одна из компонент связности графа разделяется на большее число компонент. Очевидно, что можно ограничиться множеством ребер, входящих только в эту разделяемую компоненту. В свете этого можно ограничиться рассмотрением частного случая связного графа.

Теорема 6.7. Любое минимальное разделяющее множество связного графа разделяет граф точно на две компоненты и тем самым определяется некоторым разделением вершин графа на два множества (т.е. является разрезом).

◀ Предположим, что граф разделился на три или большее число компонент. Поскольку изначально граф был связным, в разрез входит ребро, соединяющее первую компоненту (при любой изначально выбранной нумерации компонент) с какой-то другой. Удаление этого ребра из разделяющего множества приводит к слиянию двух компонент, но оставляет граф несвязным.

Таким образом, мы приходим к другому разделяющему множеству, которое является подмножеством рассматриваемого разделяющего множества. Это противоречит условию минимальности разделяющего множества.

Минимальное разделяющее множество делит граф в точности на две компоненты G_1 и G_2, разделяя тем самым множество вершин графа на две группы V_1 и V_2. Очевидно, что рассматриваемое разделяющее множество содержит все ребра, соединяющие вершины из V_1 с вершинами из V_2, а в силу минимальности не содержит никаких других ребер. Значит, рассматриваемое разделяющее множество порождается группами вершин V_1 и V_2 и является разрезом. ►

Теорема 6.8. Любой разрез графа можно представить как объединение минимальных разрезов.

◄ Чтобы доказать сформулированное утверждение, нужно показать, что любое ребро рассматриваемого разреза входит в некоторый минимальный разрез, являющийся частью рассматриваемого разреза. Из этого факта сразу вытекает, что любой разрез есть объединение минимальных разрезов, подчиненных данному.

Как и выше, можем ограничиться случаем связного графа $G = (V,\ E)$. Рассмотрим некоторое ребро γ разреза R, соединяющее вершины v_1 и v_2. Рассмотрим вспомогательный граф, вершинами которого являются компоненты связности графа $(V,\ E \setminus R)$ (графа G без разреза R), полагая, что две такие компоненты соединены ребром, если в разрезе есть ребро, соединяющее вершины двух компонент связности. Этот вспомогательный граф называют ***конденсацией графа*** G. Так как любые две вершины одной компоненты связности достижимы друг из друга, то связь между двумя компонентами связности можно провести по любому ребру, соединяющему вершины этих компонент. Возникает отношение эквивалентности на ребрах графа G, причем каждому классу эквивалентности будет соответствовать одно ребро конденсированного графа.

В силу построения разрез состоит из некоторого множества упомянутых выше классов эквивалентных ребер. Удаление разреза приводит к тому, что конденсированный граф становится графом без ребер. Отметим, что любую цепь в конденсированном графе, соединяющую компоненты с вершинами v_1, v_2 можно детализировать так, что получится цепь, соединяющая в исходном графе вершины v_1, v_2.

Из этих рассуждений вытекает, что минимальный разрез графа G, подчиненный разрезу R представляет собой объединение нескольких классов эквивалентности, а потому соответствует некоторому разрезу конденсированного графа.

Поскольку разрез порождается двумя множествами вершин V_1 и V_2, конденсированный граф будет состоять из двух групп вершин (компонент связности, попадающих в V_1 или V_2), причем ребра графа будут соединять вершины разных групп (такой граф называется ***двудольным***). Пусть Γ — класс эквивалентности, содержащий ребро γ. Покажем, что Γ является частью некоторого минимального разреза конденсированного графа. Этот разрез порождает минимальный разрез исходного графа, подчиненный разрезу R. Пусть $\tilde{v}_1$ и $\tilde{v}_2$ — вершины конденсированного графа, соответствующие вершинам v_1 и v_2 исходного.

Выберем совокупность тех ребер конденсированного графа, инцидентных $\tilde{v}_1$, которые являются началом цепи из вершины $\tilde{v}_1$ в вершину $\tilde{v}_2$. Эта совокупность является разделяющим множеством конденсированного графа, поскольку его вершины $\tilde{v}_1$ и $\tilde{v}_2$ после удаления множества ребер оказываются недостижимыми друг из друга. В то же время эта совокупность — минимальное разделяющее множество, поскольку для любых двух вершин есть цепь, соединяющая их. Удаляемое множество ребер либо не затрагивает указанную цепь, либо эта цепь проходит через $tilde v_1$. Но тогда либо восстанавливаемое ребро восстанавливает и всю цепь, либо восстановленное ребро позволяет по этому ребру попасть в вершину $\tilde{v}_2$, а затем вернувшись к вершине, соседней с $\tilde{v}_1$, затем построить альтернативную цепь. Значит, сокращение количества ребер в разделяющем множестве восстанавливает связность.

Поскольку минимальное разделяющее множество есть простой разрез, любое ребро разреза R будет входить в некоторый простой разрез, целиком входящий в R. ►

Систему минимальных разрезов, порождающих алгебру разрезов, можно строить, используя остов графа. Поскольку любые вершины связного графа можно соединить цепью в пределах остова графа, любой разрез графа должен быть и разрезом остова. Разрезы дерева устроены просто: минимальным разрезом дерева является любое ребро, поскольку удаление ребра в дереве автоматически делает это дерево несвязным.

Из этого рассуждения вытекает, что любой разрез графа должен содержать хотя бы одно древесное ребро. Можно построить систему разрезов, каждый из которых содержит ровно одно древесное ребро. Достаточно к этому древесному ребру добавить обратные ребра всех содержащих его фундаментальных циклов. Любая простая цепь, соединяющая концы древесного ребра есть часть цикла, содержащего это ребро. Но любой цикл есть сумма фундаментальных циклов, а потому содержит одно из выбранных обратных ребер. Следовательно, любая простая цепь, соединяющая концы древесного ребра, окажется разорванной, а граф станет несвязным. Ясно, что указанное разделяющее множество минимально, так как удаление из него любого ребра восстанавливает одну из цепей, связывающей две вершины. Граф возвращает свою связность.

Построенная система разрезов называется ***фундаментальной системой разрезов***. В фундаментальной системе разрезов каждый элемент — минимальный разрез. Однако отсюда не следует, что любой разрез графа может быть получен как объединение некоторых разрезов фундаментальной системы. Например, треугольник с вершинами 1, 2, 3 и ребрами $\gamma_1 = \{1, 2\}$, $\gamma_2 = \{1, 3\}$, $\gamma_3 = \{2, 3\}$ (например, подграф графа на рис. 6.6, порожденный вершинами 1, 2, 3) имеет остов с двумя ребрами γ_1 и γ_2. Фундаментальную систему разрезов образуют пары ребер $\{\gamma_1, \gamma_3\}$ и $\{\gamma_2, \gamma_3\}$. Разрез $\{\gamma_2, \gamma_3\}$ также минимальный, но не является объединением предыдущих двух.

ОГЛАВЛЕНИЕ