Лекция 11. Оптимальные каркасы. Алгоритмы Прима и Краскала. Нахождение всех возможных остовов заданного графа.

Оптимальные каркасы и пути

В задаче об оптимальном каркасе (она известна также как задача о кратчайшем остовном дереве) задан граф $G = (V, E)$, каждому ребру которого приписано число, называемое весом ребра. Вес ребра (x, y) будем обозначать через $w(x, y)$. Вес множества $X \subseteq E$ определяется как сумма весов составляющих его ребер. Требуется в графе G найти каркас минимального веса. Для решения этой задачи известно несколько алгоритмов. Рассмотрим два из них.

Алгоритм Прима

Будем предполагать, что граф G связен, так что решением задачи всегда будет дерево. В алгоритме Прима (R. Prim) на каждом шаге рассматривается частичное решение задачи, представляющее собой дерево. Вначале это дерево состоит из единственной вершины, в качестве которой может быть выбрана любая вершина графа. Затем к дереву последовательно добавляются ребра и вершины, пока не получится остовное дерево, т.е. каркас. Для того чтобы из текущего дерева при добавлении нового ребра опять получилось дерево, это новое ребро должно соединять вершину дерева с вершиной, еще не принадлежащей дереву. Такие ребра будем называть *подходящими* относительно рассматриваемого дерева. В алгоритме Прима применяется следующее правило выбора: на каждом шаге из всех подходящих ребер выбирается ребро наименьшего веса. Это ребро вместе с одной новой вершиной добавляется к дереву. Если обозначить через U и F множества вершин и ребер строящегося дерева, то алгоритм Прима можно представить следующим образом. Через $\overline{U}$ обозначаем дополнение множества U до множества V.

Алгоритм Прима.

1 $U = \{a\}$, где a – произвольная вершина графа;
2 $F = \emptyset$;
3 **while** $U \neq V$ **do**
4 найти ребро наименьшего веса (x, y) среди всех ребер, у которых $x \in U, y \in \overline{U}$;
5 добавить к F ребро (x, y), а к U вершину y.

Теорема об алгоритме Прима. *Подграф* (U, F), *построенный с помощью алгоритма Прима, является каркасом наименьшего веса для данного графа.*

Оценим время работы алгоритма Прима. Цикл **while** в строке 3 повторяется $n - 1$ раз. Внутри этого цикла есть еще скрытый цикл в строке 4, где ищется ребро наименьшего веса среди подходящих ребер. Допустим, что этот поиск производится самым бесхитростным образом, т.е. просматриваются все пары вершин (x, y) с $x \in U, y \in \overline{U}$. Если $|U| = k$, то имеется $k(n - k)$ таких пар. Всего получаем

$$\sum_{k=1}^{n-1} k(n-k) = n\sum_{k=1}^{n-1} k - \sum_{k=1}^{n-1} k^2 = \frac{n^2(n-1)}{2} - \frac{(n-1)n(2n-1)}{6} = \frac{n^3 - n}{6}$$

пар, которые нужно рассмотреть. Таким образом, трудоемкость алгоритма будет $O(n^3)$.

Небольшое усовершенствование позволяет на порядок ускорить этот алгоритм. Допустим, что для каждой вершины y из множества $\overline{U}$ известна такая вершина $f(y) \in U$, что ребро $(f(y), y)$ имеет наименьший вес среди всех ребер вида (x, y), где $x \in U$. Тогда при $|U| = k$ для поиска подходящего ребра

наименьшего веса достаточно будет сравнить $n-k$ ребер вида$(f(y),y)$, а общее число анализируемых ребер будет равно

$$\sum_{k=1}^{n-1}(n-k)=\frac{n(n-1)}{2}.$$

В этом случае, однако, необходимы дополнительные действия для обновления таблицы значений функции f при добавлении новой вершины к дереву, т.е. при переносе одной вершины из множества $\overline{U}$ в множество U. Сначала, когда множество U состоит из единственной вершины a, полагаем $f(x)=a$ для всех $x\in\overline{U}$. В дальнейшем эти значения могут меняться. Допустим, на некотором шаге к дереву присоединяется вершина y. Тогда для каждой вершины $z\in\overline{U}$ либо сохраняется старое значение $f(z)$, либо устанавливается новое $f(z)=y$, в зависимости от того, какое из ребер $(f(z),z)$ и (y,z) имеет меньший вес. По окончании работы алгоритма массив f будет массивом отцов построенного дерева относительно корня a. Поэтому отпадает необходимость отдельно запоминать добавляемые к дереву ребра. Алгоритм теперь можно записать следующим образом.

Алгоритм Прима усовершенствованный.

1 $U:=\{a\}$, где a – произвольная вершина графа;
2 **for** $y\in\overline{U}$ **do** $f(y):=a$;
3 **while** $U\neq V$ **do**
4 найти вершину $y\in\overline{U}$ с наименьшим значением $w(f(y),y)$;
5 добавить к U вершину y;
6 **for** $z\in\overline{U}$ **do** **if** $w(f(z),z)>w(y,z)$ **then** $f(z):=y$.

При $|U|=k$ цикл в строке 6 повторяется $n-k$ раз. Таким образом, дополнительное время, необходимое для обслуживания массива f, тоже оценивается сверху квадратичной функцией от n и общая оценка трудоемкости усовершенствованного алгоритма Прима будет .

Алгоритм Краскала

Другой алгоритм для задачи об оптимальном каркасе известен как алгоритм Краскала (J. Kruskal). В нем тоже на каждом шаге рассматривается частичное решение. Отличие от алгоритма Прима состоит в том, что в алгоритме Краскала частичное решение всегда представляет собой остовный лес F графа G, т.е. лес, состоящий из всех вершин графа G и некоторых его ребер. Вначале F не содержит ни одного ребра, т.е. состоит из изолированных вершин. Затем к нему последовательно добавляются ребра, пока не будет построен каркас графа G. Пусть F – лес, построенный к очередному шагу. Новое ребро можно добавить к этому лесу так, чтобы он остался лесом, только если оно соединяет вершины из разных компонент связности леса F. Теперь именно такие ребра будем называть подходящими. Для выбора добавляемого ребра применяется тот же принцип, что и в алгоритме Прима – из всех подходящих ребер выбирается ребро наименьшего веса. Для того чтобы облегчить поиск этого ребра, вначале все ребра графа упорядочиваются по возрастанию весов: $w(e_1)\le\ldots\le w(e_m)$. Теперь последовательность ребер $e_1,\ldots,e_m$ достаточно просмотреть один раз и для очередного рассматриваемого ребра нужно определить, является ли оно подходящим относительно построенного к этому моменту леса F. Подходящие ребра добавляются к F, остальные просто пропускаются.

Теорема об алгоритме Краскала. *Лес F, построенный с помощью алгоритма Краскала, является каркасом наименьшего веса для данного графа.*

Скорость работы алгоритма Краскала зависит от того, как организовано хранение информации о текущем лесе и как выполняется проверка подходящих ребер. Применение специальных структур данных, которые мы не будем здесь рассматривать, позволяет реализовать этот алгоритм с оценкой времени работы $O(mf(n))$, где $f(n)$ – очень медленно растущая функция. В частности, $f(n)<5$ для $n<2^{65536}$.

Кратчайшие пути

Ранее уже была рассмотрена задача о кратчайших путях, для решения которой можно применять метод поиска в ширину. В той задаче под длиной пути понималось число ребер в нем, т.е. все ребра считались имеющими одну и ту же единичную длину. Теперь рассмотрим более общую задачу, в которой длины ребер могут быть различными. Предполагаем, что задан связный граф $G=(V,E)$ и для каждого его ребра (x,y) указана длина этого ребра $l(x,y)$ – неотрицательное число. Длина пути есть сумма длин его ребер.

Рассмотрим задачу отыскания путей наименьшей длины от заданной вершины *a* до всех остальных вершин графа. Возможно, более естественной постановкой задачи кажется поиск кратчайшего пути между двумя заданными вершинами. Однако в настоящее время не известно никакого способа решения этой задачи, существенно лучшего, чем поиск кратчайших путей от начальной вершины до всех остальных. Наиболее известным способом решения этой задачи является описываемый далее алгоритм Дейкстры (E. Dijkstra).

Решение задачи о кратчайших путях из заданной вершины удобно представлять в виде *дерева кратчайших путей.* Это дерево с корнем *a*, являющееся каркасом графа *G*, в котором путь от любой вершины до корня является кратчайшим путем между этими вершинами во всем графе.

Алгоритм Дейкстры очень похож на алгоритм Прима. В нем процесс построения дерева тоже начинается с одной вершины, только в алгоритме Прима это могла быть любая вершина графа, а теперь это должна быть вершина a. В дальнейшем на каждом шаге к дереву присоединяется одно новое ребро (и одна вершина). Это ребро выбирается из подходящих ребер, причем понятие подходящего ребра здесь такое же – это ребро, соединяющее вершину дерева с вершиной, ему не принадлежащей. И правило выбора добавляемого ребра такое же – среди подходящих ребер выбирается ребро наименьшего веса. Разница в том, что в алгоритме Прима веса ребер заданы изначально, а в алгоритме Дейкстры они вычисляются.

Пусть $T=(U,F)$ – частичное дерево кратчайших путей, построенное к некоторому моменту. Обозначим через $L(x)$ длину пути между вершинами *x* и *a* в дереве *T*. В качестве веса подходящего ребра (x,y) принимается величина $L(x)+l(x,y)$.

$$L(x)=\min_{y\in T}\{L(y)+l(y,x)\}$$

Теорема об алгоритме Дейкстры. *Дерево, построенное с помощью алгоритма Дейкстры, является деревом кратчайших путей.*

Алгоритм Дейкстры для ускорения работы модифицируется так же, как алгоритм Прима. Распространим определение функции *L* на множество $\overline{U}$: для $y\in\overline{U}$ положим $L(y)$ равным наименьшему значению величины $L(z)+l(z,y)$ по всем $z\in U$. Через $f(y)$ обозначим то значение *z*, на котором этот минимум достигается. Таким образом, $L(y)$ – это длина кратчайшего пути из *a* в *y*, проходящего только через вершины множества *U* (кроме вершины *y*), а $f(y)$ – предпоследняя вершина этого пути. Теперь алгоритм Дейкстры можно записать следующим образом.

Алгоритм Дейкстры.

7 $U := \{a\}$; $L(a) := 0$;

8 **for** $y \in \overline{U}$ **do** **{** $f(y) := a$ **;** $L(y) := l(a, y)$ **};**

9 **while** $U \neq V$ **do**

10 найти вершину $y \in \overline{U}$ с наименьшим значением $L(y)$;

11 добавить к U вершину y;

12 **for** $z \in \overline{U}$ **do** **if** $L(y) + l(y, z) < L(z)$ **then** **{** $f(z) := y$ **;** $L(z) := L(y) + l(y, z)$ **}**.

Процедура порождения всех деревьев графа

Поскольку, как уже упоминалось ранее, число остовных деревьев графа очень быстро растет с ростом числа его дуг, то, очевидно, нужен эффективный метод порождения исчерпывающего, но без повторений, списка остовных деревьев. Один из таких способов состоит в использовании элементарных преобразований деревьев для последовательного порождения остовов, начиная с некоторого начального остова T_0 Методы, основанные на этом принципе, даны в работах Маеды [1], Пауля [2], Чена [3] и других [4].

Однако процедурам, опирающимся на элементарные преобразования деревьев, присущ следующий недостаток: для порождения нового дерева необходимо привлекать все найденные ранее деревья. Число деревьев становится столь большим, что для хранения их необходимо (с вычислительной точки зрения) использовать вспомогательные устройства памяти, так как быстродействующая оперативная память для этой цели мала. Поскольку при обращении к вспомогательным устройствам памяти значительно возрастает время вычисления, то любой метод, базирующийся на элементарных преобразованиях деревьев, оказывается неэффективным, если не будут применяться такие преобразования, в которых используется только последнее из полученных остовных деревьев. Очевидно, что наилучшим будет такой алгоритм порождения всех остовов графа, когда список остовов строится без повторений и построенные остовы записываются во вспомогательную память, но в процессе работы алгоритма из этой памяти ничего не берется.

Далее мы опишем один такой алгоритм, основанный на поиске, использующем дерево решений. В работах [5,6,7] даются другие алгоритмы, базирующиеся на порождении всех главных квадратных подматриц приведенной матрицы инциденций B_0.

Основной метод. Мы приводим описание алгоритма для случая неориентированного графа, но его распространение на ориентированные графы — для порождения всех остовных древовидностей ориентированного графа — совершенно очевидно.

Первый шаг метода состоит в приписывании номеров ребрам графа G (ребра нумеруются от 1 до m, где m число ребер в графе G. На каждом этапе (т. е. при каждом ветвлении в дереве решений) выбирается ребро, которое вместе с остальными, выбранными уже на предыдущих этапах, будет образовывать часть конструируемого дерева. Таким образом, прежде чем отобрать такое ребро, выясняют, действительно ли добавление его к частично сформированному дереву (которое на этом шаге является набором поддеревьев) не приводит к появлению цикла. Если цикл появляется, то данное ребро отбрасывается и проверке подвергается следующее ребро, с большим номером. Если цикла нет, то ребро добавляется к другим, уже отобранным, и процесс продолжается до тех пор, пока не будет построено остовное дерево. Ребра перебираются в порядке возрастания их номеров; это приводит к исчерпывающему и без повторений решению задачи.

Для организации проверки на возможность образования цикла (при добавлении ребра) каждую вершину x_j помечают парой (r_j, p_j). Первая пометка r_j указывает «корень» поддерева, содержащего вершину x_j. Первоначально $r_j = x_j$ для всех вершин x_j. На некотором шаге два поддерева T_1 и T_2 сращиваются посредством добавления ребра $a_l = (x_\alpha, x_\beta), x_\alpha \in T_1, x_\beta \in T_2$. Если на этом шаге $r_k, (k = 1,2)$ «корневая» пометка вершин в T_k и $r_1 < r_2$ (например), то все вершины в T_2 должны «сменить» свои корневые пометки на r_1 и два поддерева T_1 и T_2 «сольются» в единственное новое дерево T_1.

Вторая пометка p_j приписанная вершине x_j указывает вершину, предшествующую вершине x_j т. е. если (x_k, x_j) − дуга рассматриваемого поддерева, то $p_j = x_k$. Для корневой вершины дерева такая пометка полагается равной нулю.

А. Замена корня дерева

Если корнем дерева T является вершина r и нужно в качестве корня выбрать новую вершину x_s то такую «замену» r на x_s можно осуществить простым обращением ориентации дуг, принадлежащих цепи, идущей от r к x_s не меняя при этом ориентацию других дуг. Соответствующие изменения пометок будут таковы.

Изменение пометок «предшествования»

1. Пусть $x_j = x_s$ и $z = p_j$
2. Положить $x_i = z$, а $z = p_i$
3. Шаг обновления: $p_i = x_i$
4. Если $x_i = r$ то перейти к шагу 5, в противном случае положить $x_j = x_i$ и перейти к шагу 2.
5. Положить $p_s = 0$ стоп.

Изменение корневых пометок

1. У всех вершин, имеющих корневую пометку r заменить ее на пометку x_s.

Б. Сращивание двух поддеревьев

Если осуществлено сращивание двух поддеревьев T_1 и T_2 (путем добавления ребра (x_α, x_β) как было описано ранее), то в пометки необходимо внести следующие изменения:

(i) У вершин с корневой пометкой r_2 заменить эту пометку на r_1.

(ii) Заменить в дереве T_2 корень r_2 на x_β (так, как было описано выше в пункте А), после чего изменить пометку «предшествования» у вершины x_β с $p_\beta = 0$ на $p_\beta = x_\alpha$.

В. Расщепление дерева на две части

Поскольку метод порождения деревьев, рассматриваемый ниже, является поиском, использующим дерево решений, то возникает необходимость удаления некоторых ребер (на шагах возвращения), чтобы испытать затем другие ребра. В такой ситуации удаление ребра приводит к расщеплению некоторого дерева на две части, например на T_1 и T_2 и в пометки одного из этих поддеревьев должны быть внесены изменения. Пусть удаляется ребро $(x_\alpha, x_\beta), x_\alpha \in T_1, x_\beta \in T_2$. Тогда при $p_\beta = x_\alpha$ (т. е. если ребро (x_α, x_β) в первоначальном дереве ориентировано от x_α к x_β) пометки в дереве T_1 можно оставить прежними, а пометки в дереве T_2 должны быть изменены. Если же $p_\alpha = x_\beta$ (т. е. ребро ориентировано от x_β к x_α), то можно не менять пометки в дереве T_2 но нужно изменить пометки в T_1. Предполагая, что пометки меняются в дереве T_2 покажем, как надо «восстанавливать» корень в этом дереве.

1. Положить $S = \{x_\beta\}$ и $p_\beta = 0$ (x_β будет корнем дерева T_2)
2. Найти все вершины x_j с $p_j \in S$ и изменить их корневые пометки на $r_j = x_\beta$. Если таких вершин нет, остановиться.
3. Шаг обновления: $S = S \cup \{x_j | p_j \in S\}$ вернуться к шагу 2.

Следует отметить, что ни у одной вершины, кроме нового корня x_β пометки предшествования менять не нужно. Заметим также, что число описанных выше шагов 2 и 3, которое необходимо для восстановления корня, равно длине самой длинной цепи в T_2 исходящей из вершины x_β.

Описание алгоритма.

Возьмем произвольную вершину x^* графа. Пусть его степень d^*.

Перенумеруем ребра, инцидентные этой вершине: $\{a_1, a_2, \dots, a_{d^*}\}$. Остальные ребра графа $\{a_{d^*+1}, a_{d^*+2}, \dots, a_m\}$. При порождении деревьев ребра будут перебираться в соответствии с введенной нумерацией.

Шаг 1. Приписать к вершинам пометки $(r_i,\ p_i)$, где $r_i = x_i, p_i = 0, \forall x_i$. Положить $k = 1$.

Шаг 2. Выбрать для исследования произвольное ребро. Например, $a_k = (x_i, x_j)$. Если $k \le m$, где m число ребер графа, то перейти к 2(i). При $k = m + 1$ т. е. если «неисследованных» ребер нет, перейти к шагу 5.

(i) Если $r_i \neq r_j$ то ребро a_k можно добавить к ребрам построенных поддеревьев. Перейти к шагу 3. Если $r_i = r_j$, то это означает, что вершины x_i и x_j принадлежат одному и тому же поддереву и добавление ребра a_k приведет к появлению цикла. Отбросить ребро a_k т. е. положить $k = k + 1$ и вернуться к шагу 2.

Шаг 3. Срастить два поддерева, у которых вершины имеют корневые пометки r_i и r_j применив для этого метод, описанный выше в пункте Б.

Шаг 4. Отобрав $n - 1$ ребер, мы получаем некоторое дерево. Запомнить это дерево и перейти к шагу 5. Если отобрано меньше чем $n - 1$ ребер, то положить $k = k + 1$ и вернуться к шагу 2.

Шаг 5. (Возвращение.) Удалить ребро, добавленное последним. Предположим, что таким ребром является a_l Если a_l единственное оставшееся для добавления ребро, $l = d^*$ то остановиться. Все остовные деревья, таким образом, построены. (При любом дальнейшем ветвлении дерева решений вершина x^* останется изолированной.)

В противном случае надо обновить пометки, действуя так, как указано в пункте В, положить $k = l + 1$ и возвратиться к шагу 2.

Рассмотрим результат применения алгоритма к графу, изображенного на рис. 10.1.

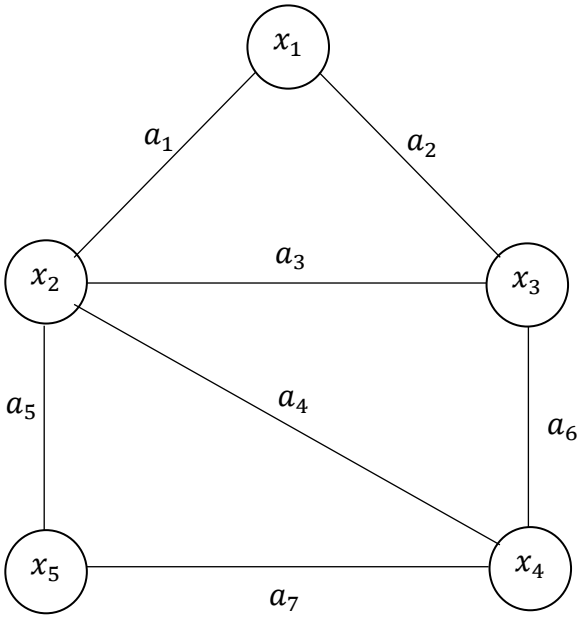


Рис 10.1. Граф для примера

Выберем в качестве x^* вершину x_1; имеем $d^* = 2$.
Перенумеруем ребра, инцидентные этой вершине: В нашем случае $\{a_1, a_2\}$. Остальные ребра графа $\{a_3, a_4, \dots, a_7\}$. На рис. 10.2 изображено соответствующее дерево решений (оно порождено в процессе работы алгоритма).

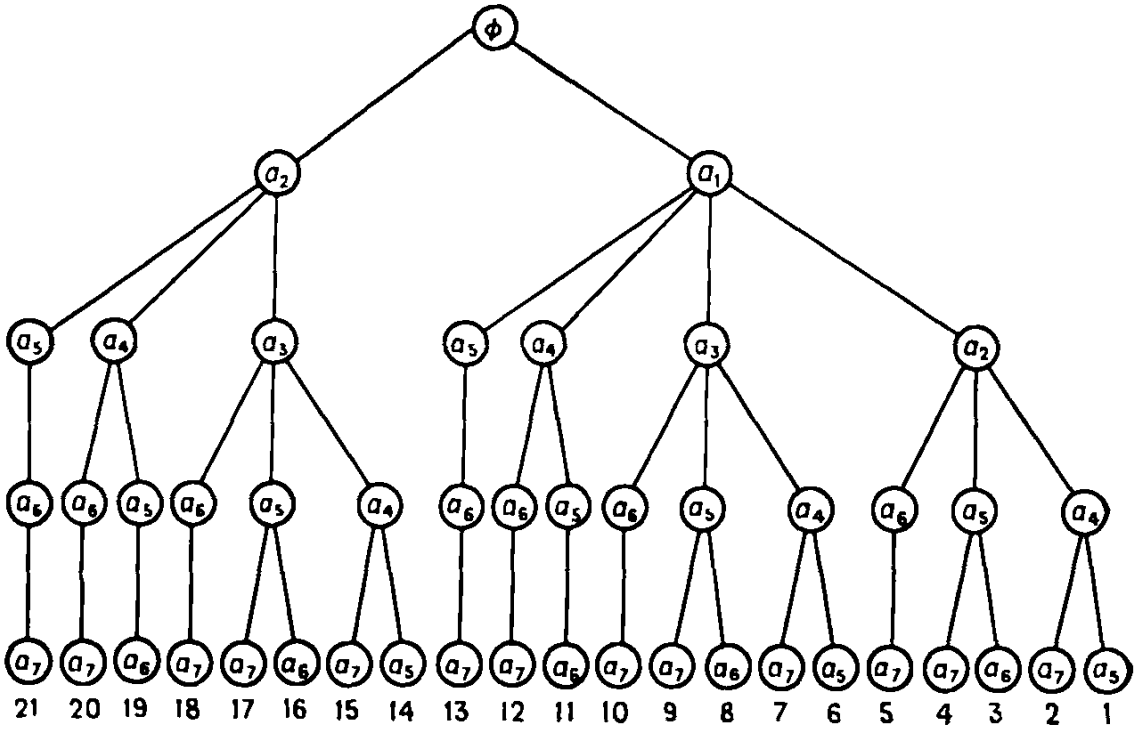


Рис 10.2. Дерево решений

Если взять ребра, указанные в кружочках какой-либо цепи, выходящей из верхнего узла этого дерева и оканчивающейся в самом нижнем узле, то из них можно построить некоторый остов данного графа. Эти остовы перенумерованы числами от 1 до 21 и приведены на рис. 10.3.

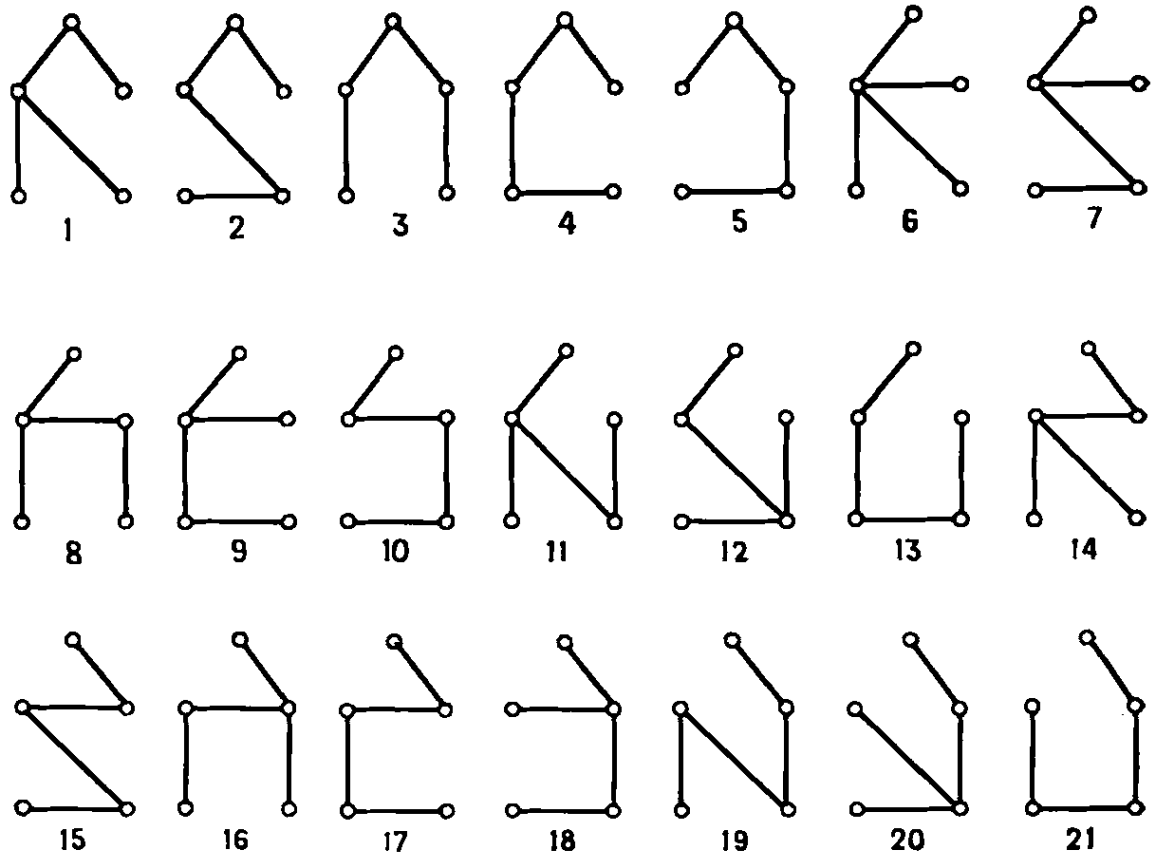
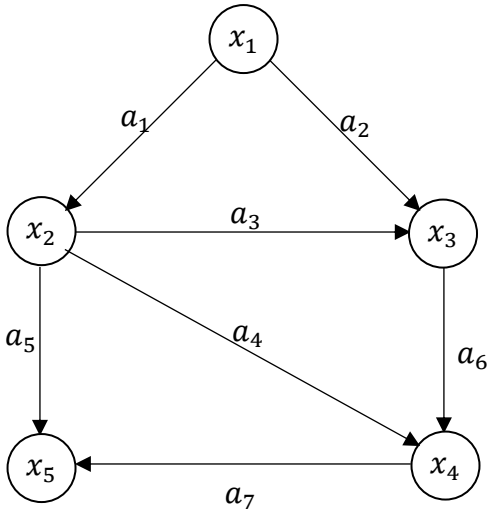


Рис 10.2. Все возможные остовные деревья

Покажем, что у графа, изображенного на рис. 10.1, действительно 21 остов. Это можно установить с помощью теоремы Кирхгофа. Матрица инциденций A данного графа имеет следующий вид (считаем, что каждое ребро ориентировано от его концевой вершины с меньшим индексом к вершине с большим индексом):



Матрица инциденций выглядит следующим образом:

$A =$

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
x_1	-1	-1	0	0	0	0	0
x_2	1	0	-1	-1	-1	0	0
x_3	0	1	1	0	0	-1	0
x_4	0	0	0	1	0	0	-1
x_5	0	0	0	0	1	1	1

Удалим произвольную строку. Например, вторую строку

$A_0 =$

-1	-1	0	0	0	0	0
0	1	1	0	0	-1	0
0	0	0	1	0	0	-1
0	0	0	0	1	1	1

$A_0^T =$

-1	0	0	0
-1	1	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	-1	0	1
0	0	-1	1

$C = A_0 A_0^T =$

2	-1	0	0
-1	3	0	-1
0	0	2	-1
0	-1	-1	3

$$\det(C) = 21$$