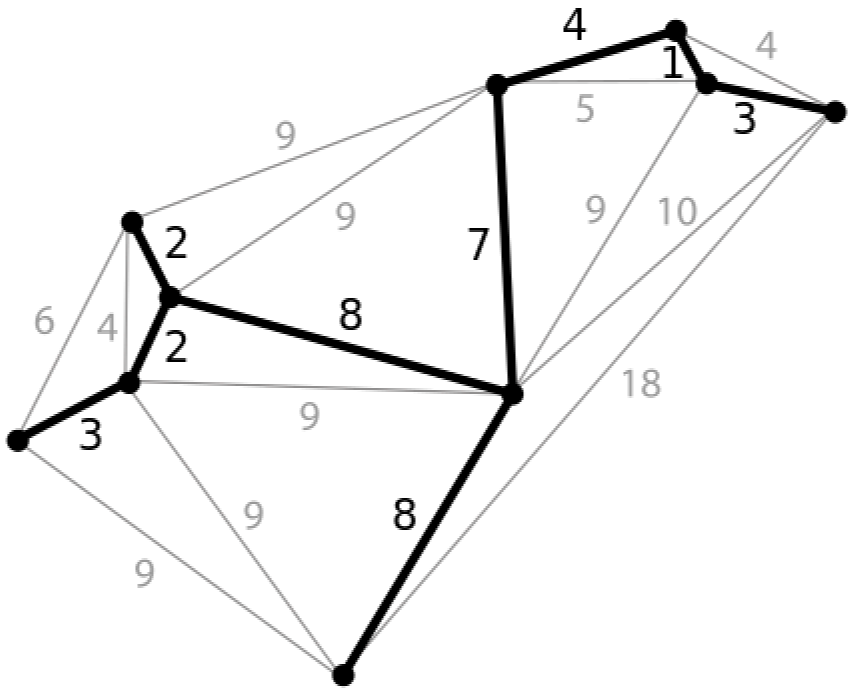
Минимальное остовное дерево

Остовное дерево графа состоит из минимального подмножества рёбер графа, таких, что из любой вершины графа можно попасть в любую другую вершину, двигаясь по этим рёбрам



Остовное дерево также иногда называют *покрывающим деревом*, *остовом* или *скелетом* графа

Свойства

- Любое остовное дерево в графе с **n** вершинами содержит ровно **n-1** ребро.
- Не содержит циклов
- Число остовных деревьев в [полном графе](#) на n вершинах выражается [формулой Кэли](#)

$$n^{n-2}$$

По́лный граф — простой неориентированный граф, в котором каждая пара различных вершин смежна. Полный граф с **n** вершинами имеет **n(n-1)/2** рёбер и обозначается Kn. Является регулярным графом степени **n-1**.

Регуля́рный (одноро́дный) граф — граф, степени всех вершин которого равны, то есть каждая вершина имеет одинаковое количество соседей.

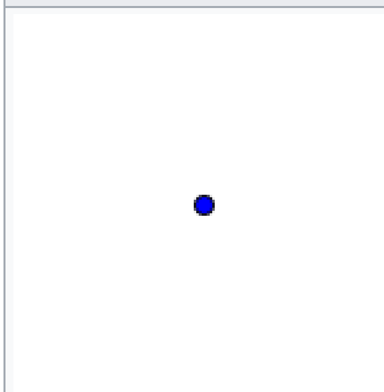
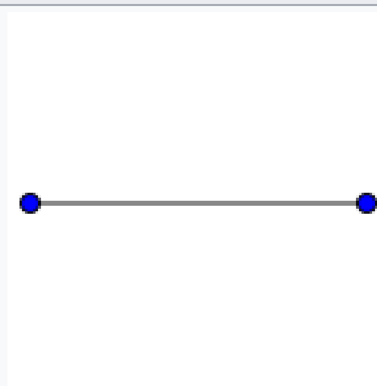
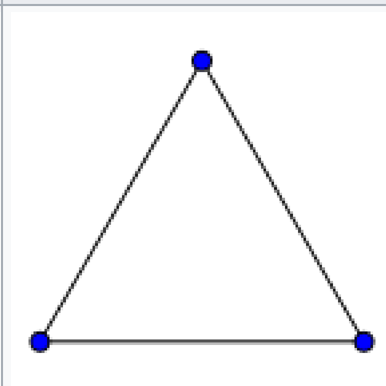
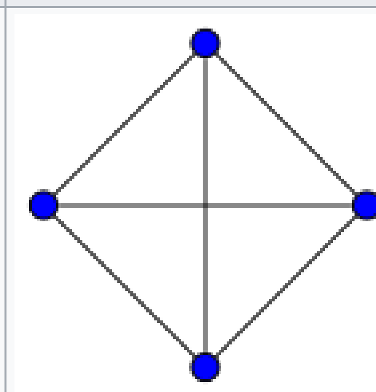
По́лный ориенти́рованный граф — ориентированный граф, в котором каждая пара различных вершин соединена парой дуг (с различными направлениями).

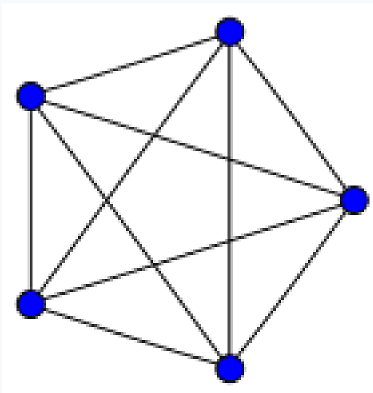
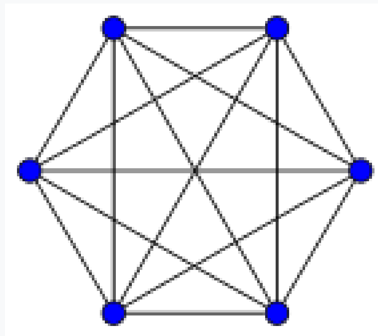
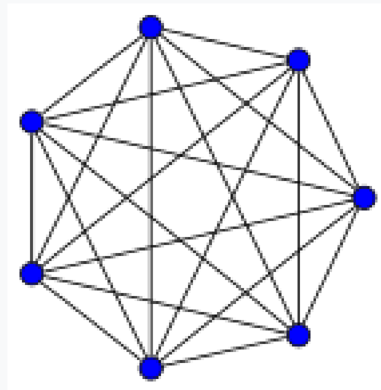
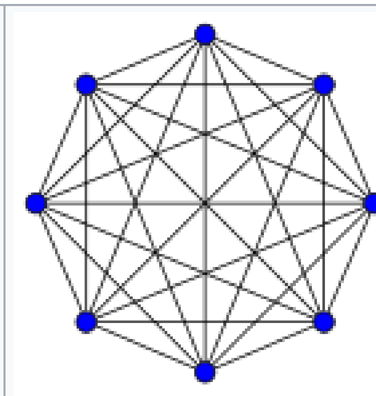
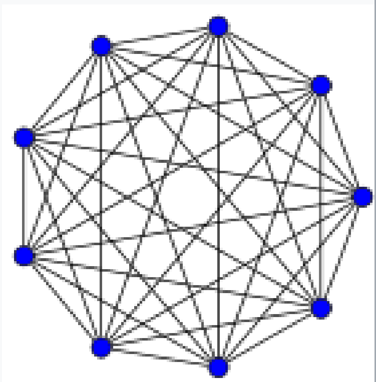
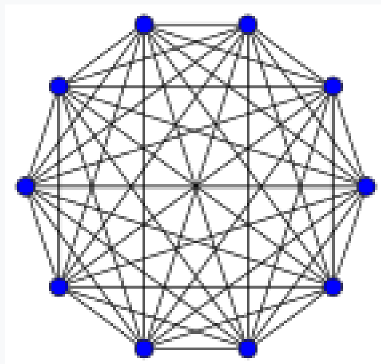
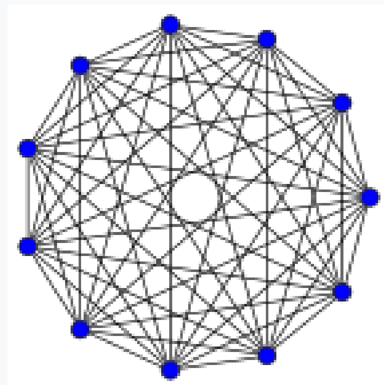
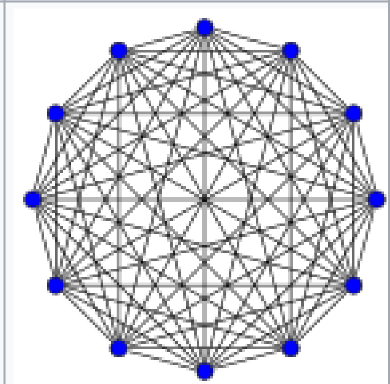
Графы с K1 по K4 являются планарными.

Плана́рный граф — граф, который может быть изображён на плоскости без пересечения рёбер. Иначе говоря, граф планарен, если он изоморфен некоторому **плоскому графу**, то есть графу, изображённому на плоскости так, что его вершины — это точки плоскости, а рёбра — непересекающиеся кривые на ней.

Примеры полных графов

Ниже приведены полные графы с числом вершин от 1 до 12 и количества их рёбер.

K_1: 0	K_2: 1	K_3: 3	K_4: 6
			
K_5: 10	K_6: 15	K_7: 21	K_8: 28

			
K_9: 36	**K_{10}: 45**	**K_{11}: 55**	**K_{12}: 66**
			

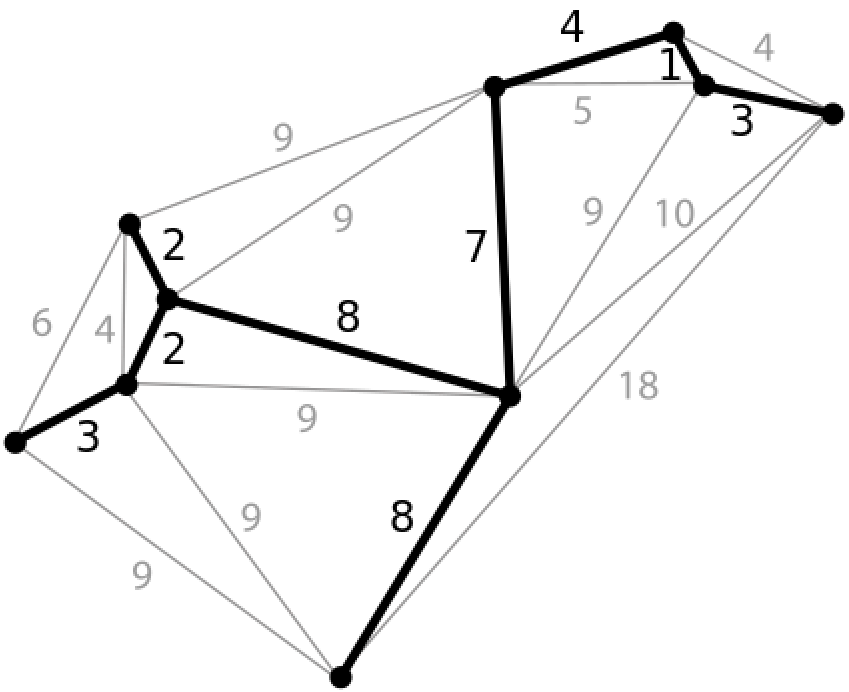
Остовные деревья

Алгоритмы

Остовное дерево может быть построено практически любым алгоритмом обхода графа, например поиском в глубину или поиском в ширину. Оно состоит из всех пар рёбер **(u,v)**, таких, что алгоритм, просматривая вершину u, обнаруживает в её списке смежности новую, не обнаруженную ранее вершину v.

Если каждому ребру графа присвоен вес (длина, стоимость и т. п.), то нахождением оптимального остовного дерева, которое минимизирует сумму весов входящих в него рёбер, занимаются многочисленные алгоритмы нахождения минимального остовного дерева

Минимальное остовное дерево (или **минимальное покрывающее дерево**) в связанном взвешенном неориентированном графе — это остовное дерево этого графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов входящих в него рёбер.



Задача о нахождении минимального остовного дерева часто встречается в подобной постановке: допустим, есть *n* городов, которые необходимо соединить дорогами, так, чтобы можно было добраться из любого города в любой другой (напрямую или через другие города). Разрешается строить дороги между заданными парами городов и известна стоимость строительства каждой такой дороги. Требуется решить, какие именно дороги нужно строить, чтобы минимизировать общую стоимость строительства

Задача Штейнера

Задача Штейнера о минимальном дереве состоит в поиске кратчайшей сети, соединяющей заданный конечный набор точек плоскости. Своё название получила в честь Якоба Штейнера (1796—1863).

История этой задачи восходит ко времени Пьера Ферма (1601—1665), который, после изложения своего метода исследования минимумов и максимумов, написал

Тот же, кто этот метод не оценил, пусть он решит [следующую задачу]: для заданных трех точек найти такую четвертую, что если из неё провести три отрезка в данные точки, то сумма этих трех отрезков даст наименьшую величину.

Р. Куранта и Г. Роббинса:
Очень простая и вместе с тем поучительная проблема была изучена в начале прошлого столетия знаменитым берлинским геометром Якобом Штейнером. Требуется соединить три деревни A, B, C системой дорог таким образом, чтобы их общая протяженность была минимальной.
*Было бы естественно обобщить эту проблему на случай n заданных точек A1,A2,...An следующим образом: требуется найти в плоскости такую точку **P**, чтобы сумма расстояний a1+a2+... +an обращалась в минимум. …*

Эта обобщенная проблема, также изученная Штейнером, не ведет к интересным результатам. В данном случае мы имеем дело с поверхностным обобщением, подобных которому немало встречается в математической литературе. Чтобы получить действительно достойное внимания обобщение проблемы Штейнера, приходится отказаться от поисков одной-единственной точки P. Вместо того поставим задачей построить «уличную сеть» или «сеть дорог между данными деревнями», обладающую минимальной общей длиной

Книга этих авторов завоевала популярность, в результате чего и задачу Ферма, и задачу Ярника—Кесслера сейчас принято называть проблемой Штейнера.

Приближенное решение задачи Штейнера дает алгоритм Краскала.

Минимальные остовные деревья

Алгоритм Прима

Алгоритм Прима — Алгоритм впервые был открыт в 1930 году чешским математиком Войцехом Ярником, позже переоткрыт Робертом Примом в 1957 году, и, независимо от них, Э. Дейкстрой в 1959 году.

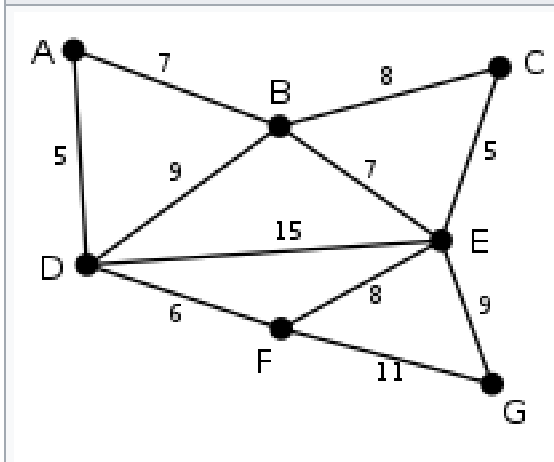
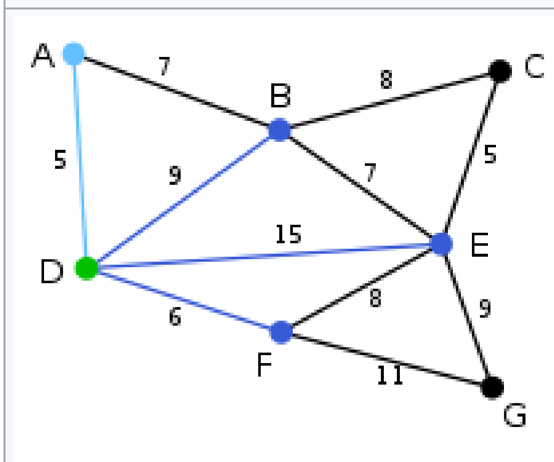
Описание

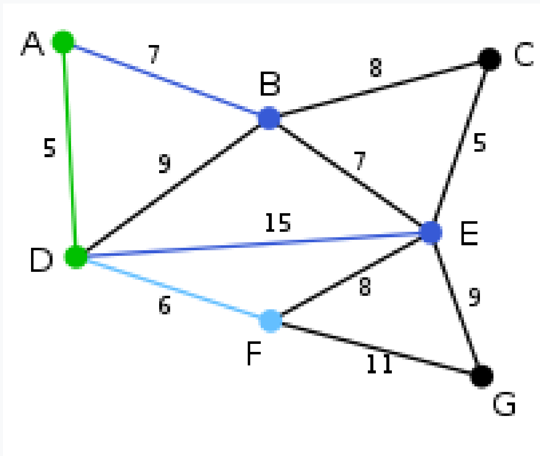
На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая вершине дерева, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

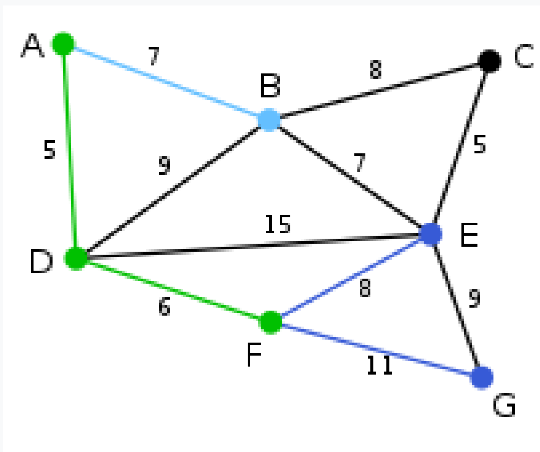
Пример

Изображение	**Множество выбранных вершин U**	**Ребро (u, v)**	**Множество невыбранных вершин V \ U**	
	{}		{A,B,C,D,E,F,G}	
	{D}	(D,A) = 5 **V** (D,B) = 9 (D,E) = 15 (D,F) = 6	{A,B,C,E,F,G}	

<table>
<tr><td>
</td><td>{A,D}</td><td>(D,B) = 9

(D,E) = 15

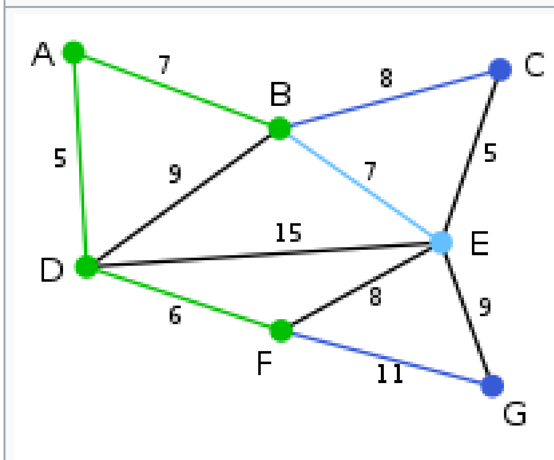
(D,F) = 6 V

(A,B) = 7</td><td>{B,C,E,F,G}</td><td></td></tr>
<tr><td>
</td><td>{A,D,F}</td><td>(D,B) = 9

(D,E) = 15

(A,B) = 7 V

(F,E) = 8

(F,G) = 11</td><td>{B,C,E,G}</td><td></td></tr>
<tr><td>
</td><td>{A,B,D,F}</td><td>(B,C) = 8

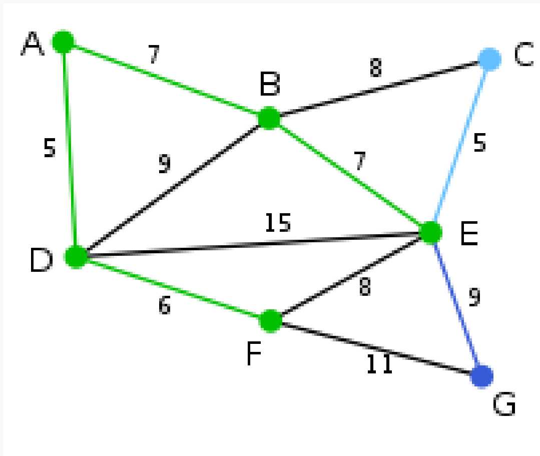
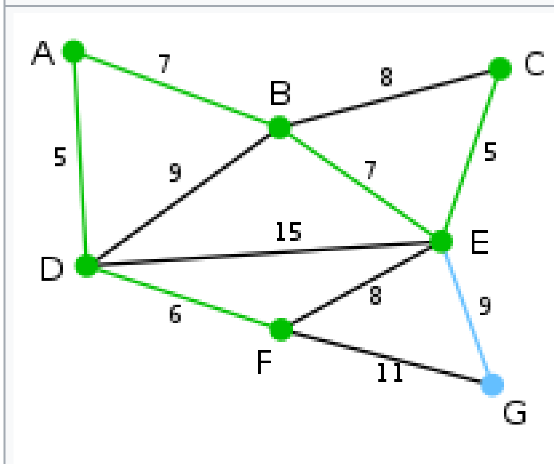
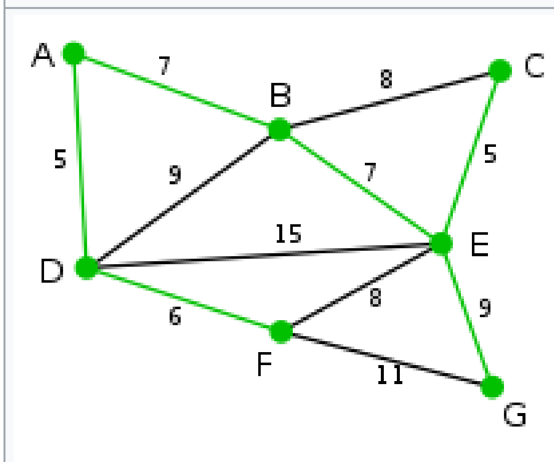
(B,E) = 7 V

(D,B) = 9 цикл

(D,E) = 15

(F,E) = 8

(F,G) = 11</td><td>{C,E,G}</td><td></td></tr>
</table>

<table>
<tr><th></th><th></th><th></th><th></th><th></th></tr>
<tr><td>
</td><td>{A,B,D,E,F
}</td><td>(B,C) = 8
(D,B) = 9 цикл
(D,E) = 15
цикл
(E,C) = 5 V
(E,G) = 9
(F,E) = 8 цикл
(F,G) = 11</td><td>{C,G}</td><td></td></tr>
<tr><td>
</td><td>{A,B,C,D,
E,F}</td><td>(B,C) = 8 цикл
(D,B) = 9 цикл
(D,E) = 15
цикл
(E,G) = 9 V
(F,E) = 8 цикл
(F,G) = 11</td><td>{G}</td><td></td></tr>
<tr><td>
</td><td>{A,B,C,D,
E,F,G}</td><td>(B,C) = 8 цикл
(D,B) = 9 цикл
(D,E) = 15
цикл
(F,E) = 8 цикл
(F,G) = 11
цикл</td><td>{}</td><td></td></tr>
</table>

Реализация

```
ключ[] - массив для минимальных значений весов ребер
остов[v] - остовное дерево с ребрами {остов[v],v}
вершины[] - вершины графа, оставшиеся для рассмотрения

ОстовноеДерево():
   для v из V
       ключ[v] = максимальное-значение
       остов[v] = пусто
       вершины.добавить(МинимальноеРебро(v), v)

   пока вершины не пусто
       v = вершины.ДостатьМинимальный()
       для u из v
           если вершины.содержит(u) и ключ[u] > вес(v,u) то
               остов[u] = v
               ключ[u] = вес(v,u)
               вершины.пересчитатьКлюч(u,ключ[u])

МинимальноеРебро(v):
     результат = максимальное-значение
     для u из v
          результат = min(результат, вес(v,u))
     вернуть результат

ПересчитатьКлюч(u, ключ)
     новыйКлюч = МинимальноеРеброБольшеКлюча(v, ключ)
     вершины.изменить(ключ, новыйКлюч, v)
```

Структуры данных для хранения вершин?

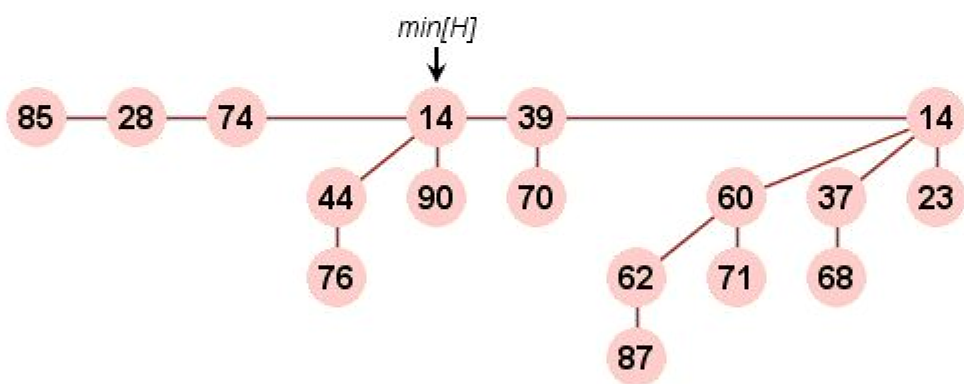
Оценка

Асимптотика алгоритма зависит от способа хранения графа и способа хранения вершин, не входящих в дерево.

Способ представления приоритетной очереди и графа	Асимптотика
Массив d, списки смежности (матрица смежности)	O(V^2)
Бинарная пирамида, списки смежности	O((V+E)\log V)=O(E\log V)
Фибоначчиева пирамида, списки смежности	O(E+V\log V)

Фибоначчиева куча (англ. *Fibonacci heap*) — структура данных, представляющая собой набор деревьев, упорядоченных в соответствии со свойством неубывающей пирамиды. Фибоначчиевы кучи были введены Майклом Фредманом и Робертом Тарьяном в 1984 году.

Структура является реализацией абстрактного типа данных «Очередь с приоритетом», и замечательна тем, что операции, в которых не требуется удаление, имеют амортизированное время работы, равное O(1) (для двоичной кучи и биномиальной кучи амортизационное время работы равно O(log n) Кроме стандартных операций `INSERT`, `MIN`, `DECREASE-KEY`, фибоначчиева куча позволяет за время O(1) выполнять операцию `UNION` слияния двух куч.



Алгоритм Борувки

Алгори́тм Бору́вки — это алгоритм нахождения минимального остовного дерева в графе.

Впервые был опубликован в 1926 году Отакаром Борувкой[en] в качестве метода нахождения оптимальной электрической сети в Моравии. Несколько раз был переоткрыт, например Флореком, Перкалом и Соллином. Последний, кроме того, был единственным западным учёным из этого списка, и поэтому алгоритм часто называют **алгоритмом Соллина**, особенно в литературе по параллельным вычислениям.

Алгоритм

1. Изначально, T — множество вершин графа V(E), (представляющее собой остовный лес, в который каждая вершина входит в качестве отдельного дерева, без ребер).
2. Пока в T число рёбер меньше, чем V-1, где V — число вершин в графе:
    - Для каждой компоненты связности (то есть, дерева в остовном лесе), найдём минимальное по весу ребро, связывающее эту компоненту с некоторой *другой* компонентой связности.
    - Объединим связанные компоненты.
3. Полученное множество рёбер T является минимальным остовным деревом входного графа.

Вспомогательный алгоритм: система непересекающихся множеств (Union-Find)

Во всех алгоритмах решения задачи требуется отслеживать, каким уже построенным фрагментам дерева принадлежат те или иные вершины графа. Для этого используется структура данных «система непересекающихся множеств» (Union-Find). Данная структура поддерживает две операции:

FIND(v)=w

– по вершине v возвращает вершину w – «корень» фрагмента, которому принадлежит вершина v. При этом гарантируется, что вершины u и v принадлежат одному и тому же фрагменту, тогда и только тогда, когда

FIND(u) == FIND(v)

MERGE(u,v)

– объединяет два фрагмента, которым принадлежат вершины u и v.

(Если они уже лежат в одном фрагменте, то ничего не происходит.) При практической реализации удобно, чтобы данная операция возвращала значение истина, если объединение фрагментов имело место, и ложь в противном случае.

Классический последовательный алгоритм Union-Find описан в статье Тарьяна. Каждой вершине v приписывается указатель на вершину-родителя

$parent(v)$

1. Изначально

$parent(v) =v$

для всех вершин.

2.

$FIND(v)$

выполняется следующим образом: полагаем $u=v$, и далее следуем по указателям

$u=parent(u)$

до тех пор, пока не станет

$u==parent(u)$

Это и будет результат операции.

Дополнительно можно «схлопывать» дерево: присвоить всем посещённым вершинами:

$parent(ui)=u$, либо производить схлопывание по пути: $parent(u)=parent(parent(u)))$

3.

$MERGE(u,v)$

выполняется следующим образом: вначале находим корневые вершины

$u=FIND(u)$,

$v=FIND(v)$

Если $u==v$, то исходные вершины принадлежат одному фрагменту и объединения фрагментов не происходит.

Иначе полагаем одно из

$$parent(u)=v$$

или

$$parent(v)=u$$

.Дополнительно можно отслеживать количество вершин в каждом из фрагментов, чтобы меньший фрагмент подсоединять к большему, а не наоборот (оценки сложности доказываются именно при такой реализации, однако на практике алгоритм хорошо работает и без подсчёта количества вершин).

Псевдокод

```
Борувка():
    для v из V
        m = МинимальноеРеброНаружу(v)
        остов.добавить(m)
        Объединить(v, m.v)
```

```
Борувка():
    для v из V
        родитель[v] = v
        следующий[v] = пусто

    для v из V
        если родитель[v] != v0 то
            m = МинимальноеРеброНаружу(v)
            остов.добавить(m)
            Объединить(v, m.исходящее)
```

```
Объединить(v, u):
        родитель[u] = родитель[v]
        следующий[v] = u

МинимальноеРеброНаружу(v):
        u = список[родитель[v]]
        мин = макс-значение
        пока u не пусто
          для m из u.ребра
                если родитель[m.исходящее] != родитель[u] и m.вес < мин.вес то
                        мин = m
          u = следующий[u]

        вернуть мин
```

Алгоритм Краскала

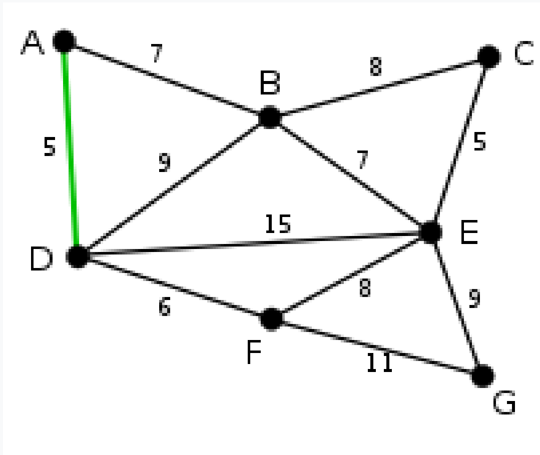
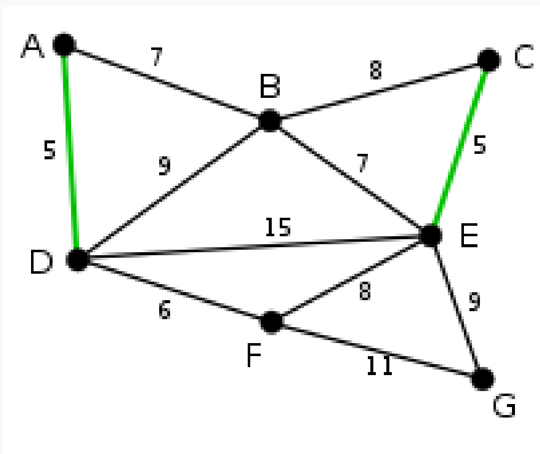
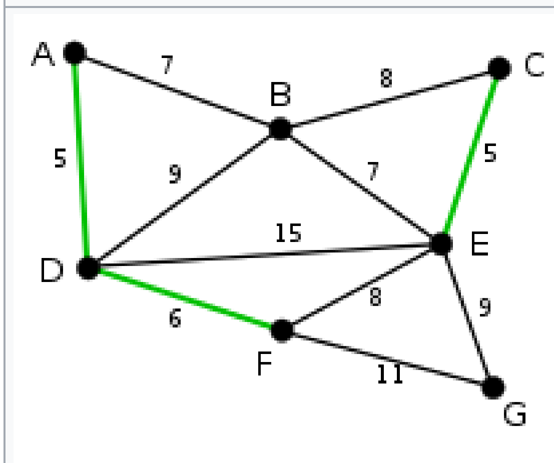
Алгоритм Краскала — алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Также алгоритм используется для нахождения некоторых приближений для задачи Штейнера.

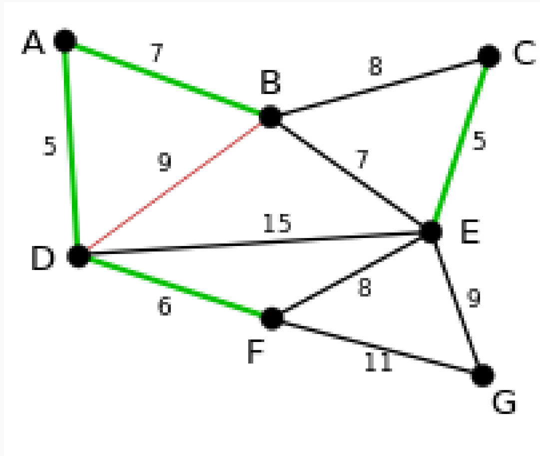
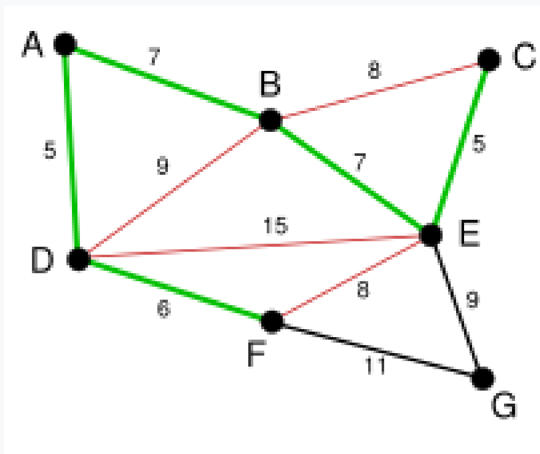
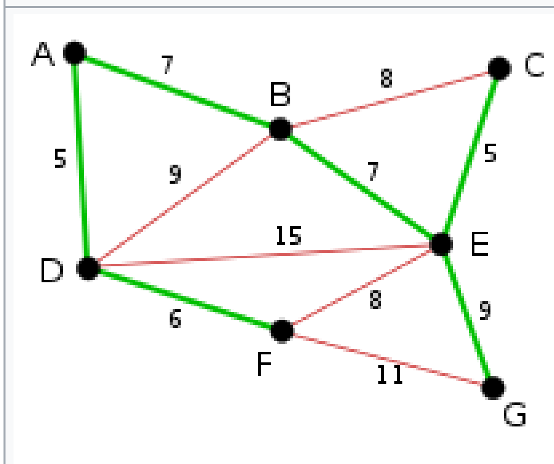
Алгоритм описан Джозефом Краскалом в 1956 году, этот алгоритм почти не отличается от алгоритма Борувки.

Идея алгоритма

Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса.

Пример

Изображение	Описание
	Ребра **AD** и **CE** имеют минимальный вес, равный 5. Произвольно выбирается ребро **AD** (выделено на рисунке).
	Теперь наименьший вес, равный 5, имеет ребро **CE**. Так как добавление **CE** не образует цикла, то выбираем его в качестве второго ребра.
	Аналогично выбираем ребро **DF**, вес которого равен 6.

	Следующие ребра — **AB** и **BE** с весом 7. Произвольно выбирается ребро **AB**, выделенное на рисунке. Ребро **BD** выделено красным, так как уже существует путь (зелёный) между **B** и **D**, поэтому, если бы это ребро было выбрано, то образовался бы цикл **ABD**.
	Аналогичным образом выбирается ребро **BE**, вес которого равен 7. На этом этапе красным выделено гораздо больше ребер: **BC**, потому что оно создаст цикл **BCE**, **DE**, потому что оно создаст цикл **DEBA**, и **FE**, потому что оно сформирует цикл **FEBAD**.
	Алгоритм завершается добавлением ребра **EG** с весом 9. Минимальное остовное дерево построено.

Псевдокод

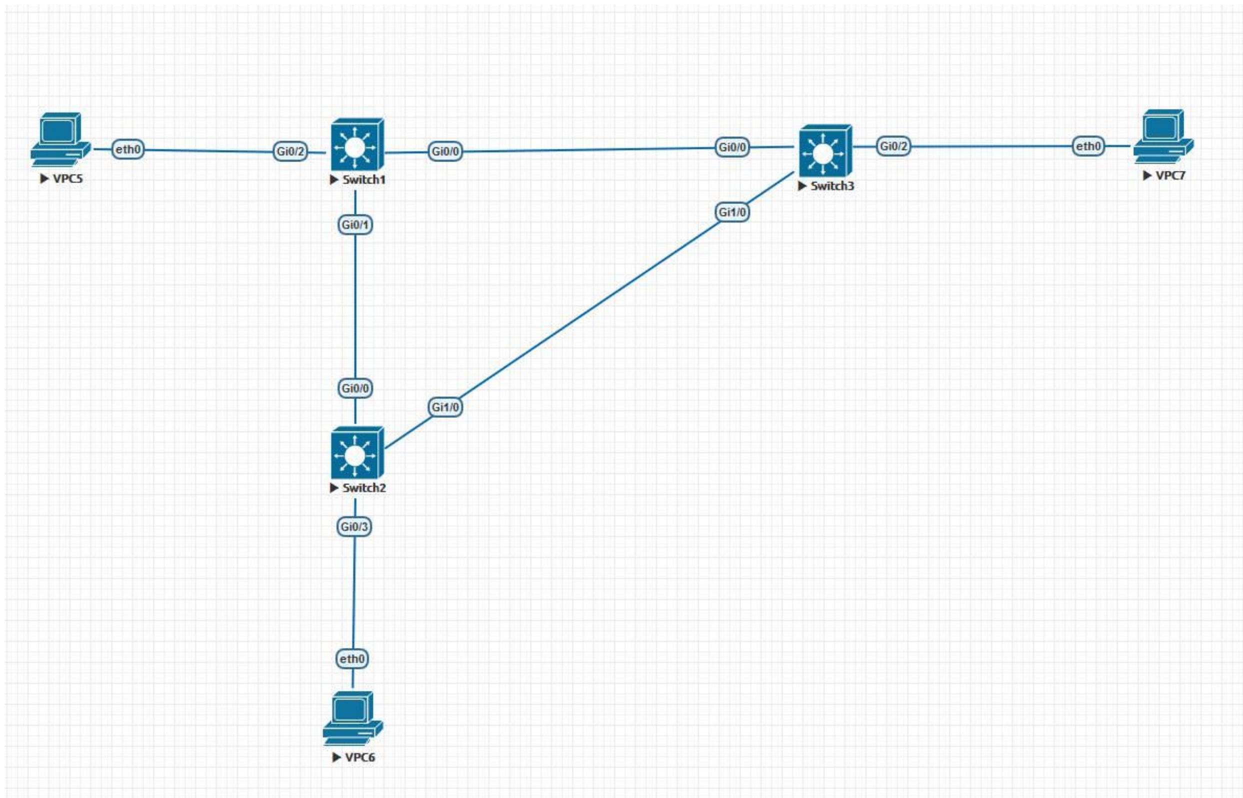
```
Краскал():
ребра.Отсортировать(G(E))

для m из ребра
        если родитель(m.входящее) != родитель(m.исходящее) то
                Объединить(m.входящее, m.исходящее)
                остов.добавить(m)
```

Сети

Сеть — это бесконтурный ориентированный граф
Если у вершины нет ни одной входящей дуги, вершина называется источником сети.
Если у вершины нет ни одной исходящей дуги, вершина называется стоком сети.

Spanning Tree Protocol (**STP**, протокол покрывающего дерева) — канальный протокол. Основной задачей STP является устранение петель в топологии произвольной сети Ethernet, в которой есть один или более сетевых мостов, связанных избыточными соединениями. STP решает эту задачу, автоматически блокируя соединения, которые в данный момент для полной связности коммутаторов являются избыточными.

Необходимость устранения топологических петель в сети Ethernet следует из того, что их наличие в реальной сети Ethernet с коммутатором с высокой вероятностью приводит к бесконечным повторам передачи одних и тех же кадров Ethernet одним и более коммутатором, отчего пропускная способность сети оказывается почти полностью занятой этими бесполезными повторами; в этих условиях, хотя формально сеть может продолжать работать, на практике её производительность становится настолько низкой, что может выглядеть как полный отказ сети.

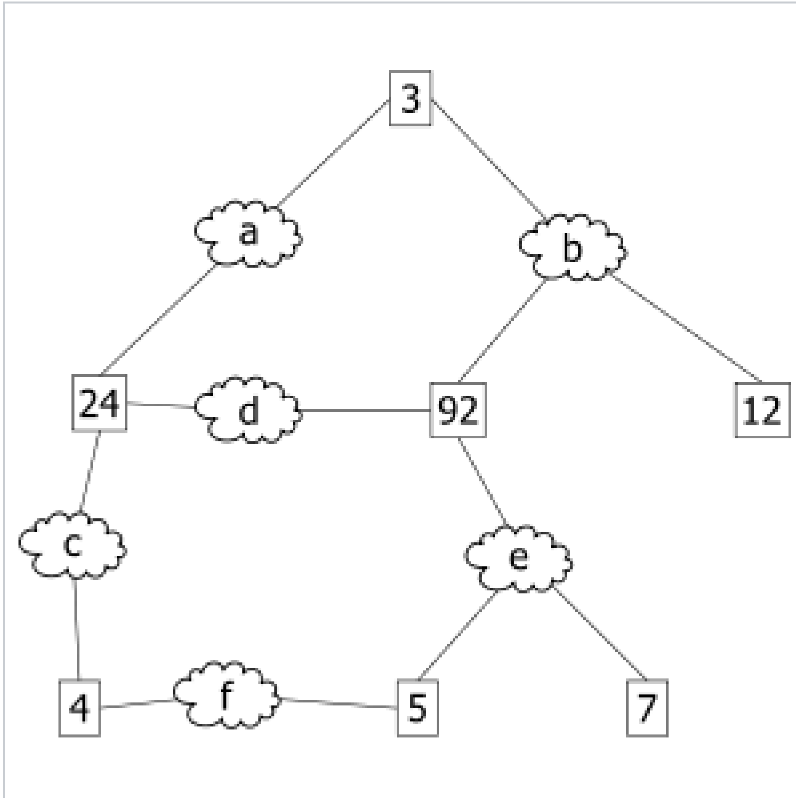


Источник возникновения проблем

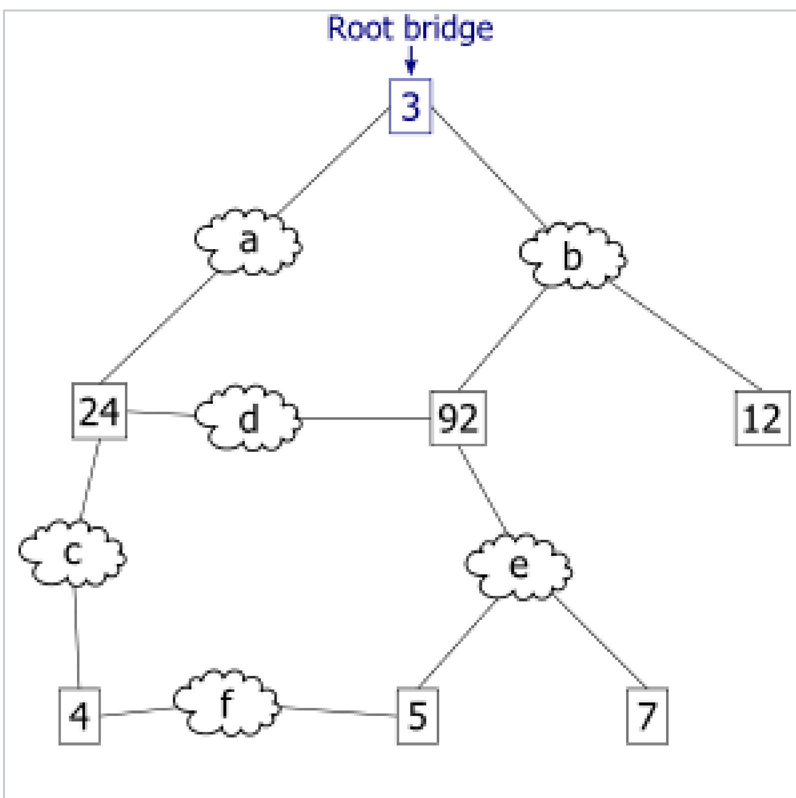
1) Бродаст фрейм
2) Юникаст адрес назначения, но в таблице адресов коммутации отсутствует данный адрес

А зачем делать петли в конфигурации сети?

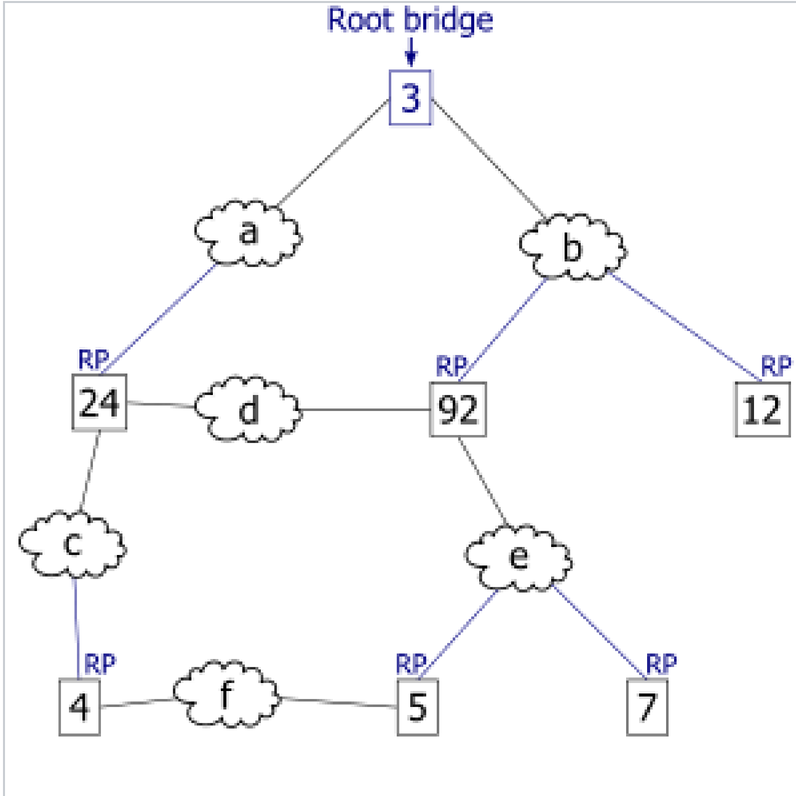
Принцип действия



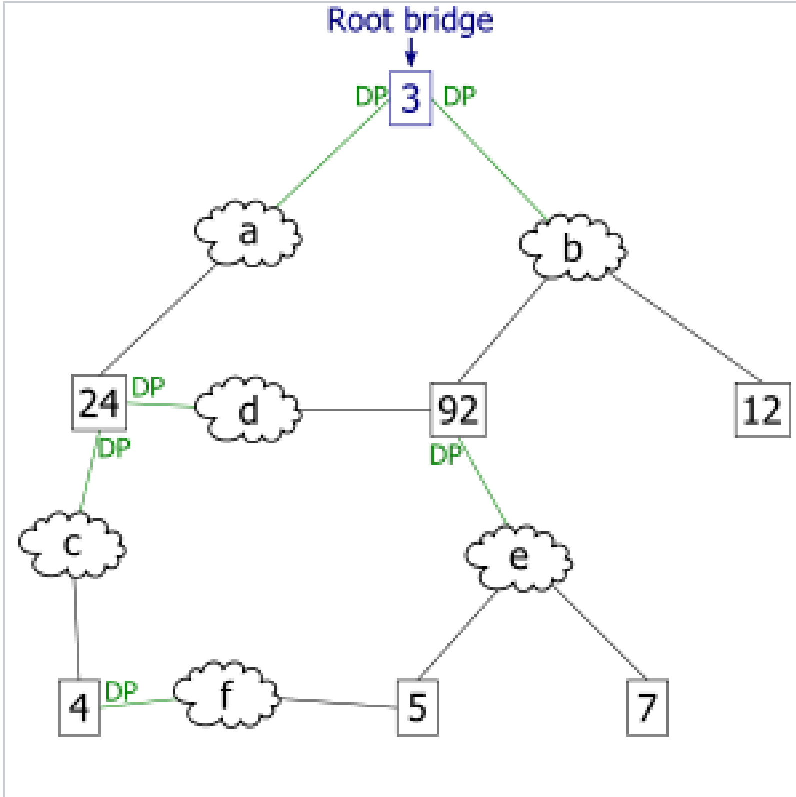
1. Пример сети. Пронумерованные квадраты означают мосты. Облачка означают сетевые сегменты.



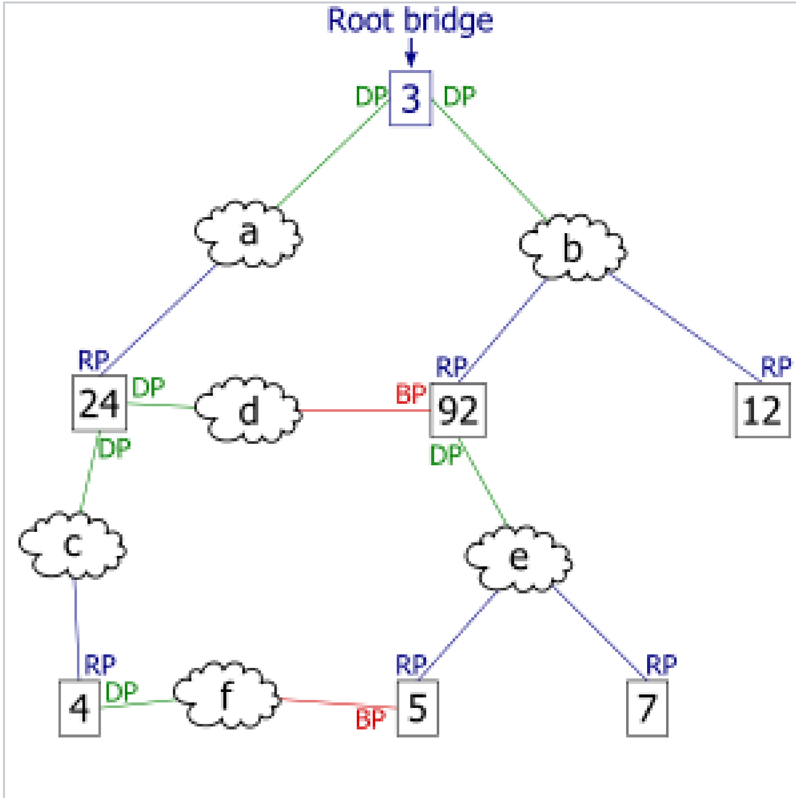
2. Наименьший ID равен 3. Следовательно, мост 3 становится корневым. (RB)



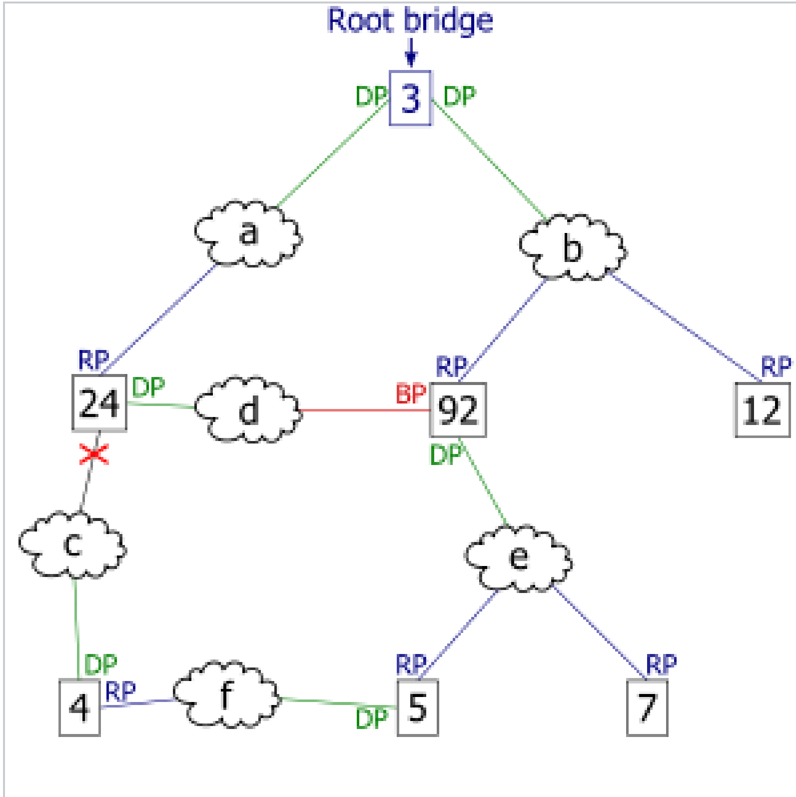
3. Предположим, что вес каждого ребра равен единице. Кратчайший путь от моста 4 к корневому мосту идёт через сегмент сети *c*. Поэтому корневым (RP) портом для моста 4 будет порт, ведущий в сеть *c*.



4. Кратчайший путь к корню из сегмента *e* идёт через мост 92. Поэтому назначенным (DP) портом для сегмента *e* будет порт, соединяющий мост 92 с сегментом *e*.



5. На этой диаграмме порты получили своё состояние с помощью STP. Активные порты, не имеющие состояния переводятся в состояние заблокированных (BP).



6. После ошибки подключения алгоритм spanning tree перестраивает дерево.

Основные понятия

- **Bridge ID** = Bridge priority + MAC;
- **Bridge priority** = vlan xxx + 4096xN, N-множитель, назначается администратором сети (4096x8=32768 default cost);
- **Cost** — «стоимость портов»;
- **Pathcost** — стоимость линка в STP;
- **Hello BPDU** = root ID + bridge ID + cost;
- ***Root port*** (корневой порт) — это порт, который имеет *минимальную стоимость* до любого порта корневого коммутатора;
- ***Designated port*** (назначенный порт) — это порт, который имеет кратчайшее расстояние от *назначенного коммутатора* до корневого коммутатора.

Важные правила

1. Корневым (root) портом назначается порт с самой низкой стоимостью пути (path cost).

2. Возможны случаи, когда стоимость пути по двум и более портам коммутатора будет одинакова, тогда выбор корневого (root) порта будет происходить на основании полученных от соседей приоритета и порядкового номера порта (Lowest Sender Port ID), например **fa0/1**, fa0/2, fa0/3 и корневым (root) станет порт с наименьшим номером.
3. Коммутаторы, по умолчанию, не измеряют состояние загрузки сети в реальном времени и работают в соответствии со стоимостью (cost) интерфейсов в момент построения дерева STP.
4. Каждый порт имеет свою стоимость (cost), обратно пропорциональную пропускной способности (bandwidth) порта и которую можно настраивать вручную.

Алгоритм действия STP (Spanning Tree Protocol)

- После включения коммутаторов в сеть, по умолчанию каждый коммутатор считает себя корневым (root).
- Каждый коммутатор начинает посылать по всем портам конфигурационные Hello BPDU пакеты раз в 2 секунды.
- Если мост получает BPDU с идентификатором моста (Bridge ID) меньшим, чем свой собственный, он прекращает генерировать свои BPDU и начинает ретранслировать BPDU с этим идентификатором. Таким образом в конце концов в этой сети Ethernet остаётся только один мост, который продолжает генерировать и передавать собственные BPDU. Он и становится *корневым мостом* (root bridge).
- Остальные мосты ретранслируют BPDU корневого моста, добавляя в них собственный идентификатор и увеличивая счётчик стоимости пути (path cost).
- Для каждого сегмента сети, к которому присоединено два и более портов мостов, происходит определение designated port — порта, через который BPDU, приходящие от корневого моста, попадают в этот сегмент.
- После этого все порты в сегментах, к которым присоединено 2 и более портов моста, блокируются за исключением root port и designated port.
- Корневой мост продолжает посылать свои Hello BPDU раз в 2 секунды.

Таймеры и сходимость протокола STP

После того, как STP завершил построение топологии без петель, остается вопрос — Как определять изменения в сети и как реагировать на них? Сообщения BPDU при помощи которых работает STP, рассылаются Root Bridge каждые 2 секунды, по умолчанию. Данный таймер называется Hello Timer. Остальные коммутаторы получив через свой root port данное сообщение пересылают его дальше через все назначенные порты. Выше сказано более подробно какие изменения происходят с BPDU при пересылки его коммутаторов. Если в течении времени, определенным таймером Max Age (по умолчанию — 20 секунд), коммутатор не получил ни одного BPDU от корневого коммутатора, то данное событие трактуется как потеря связи с Root Bridge

История создания

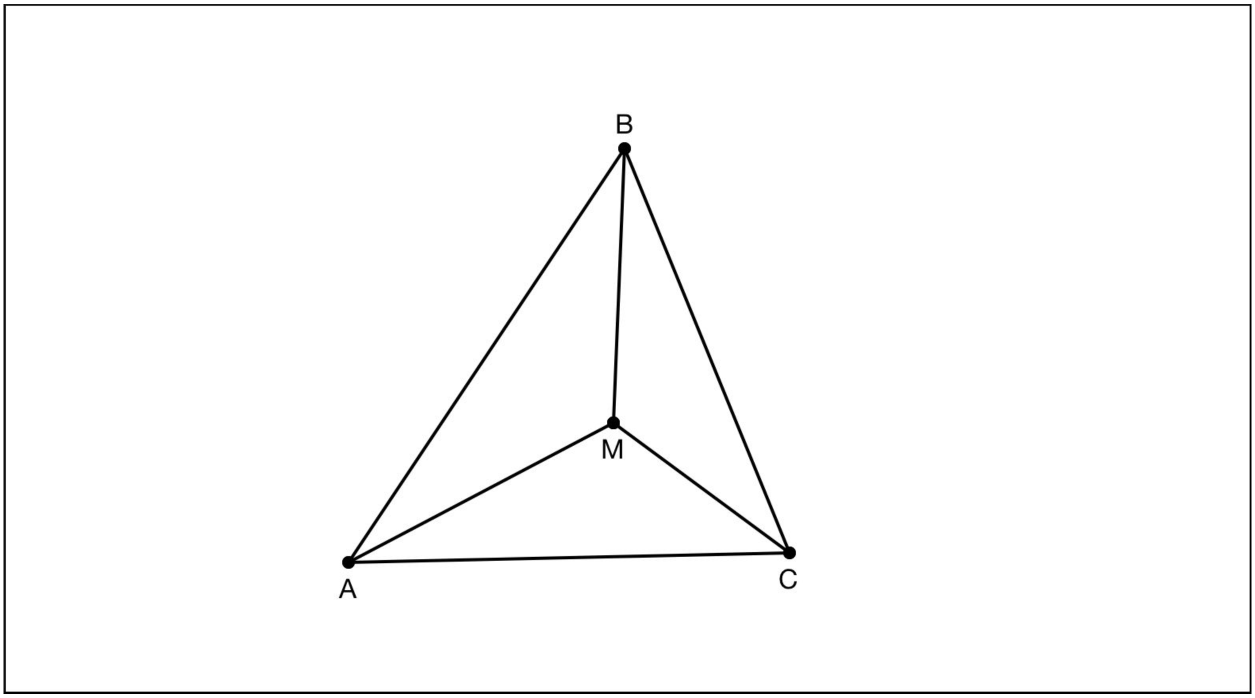
Алгоритм, заложенный в основу STP, был разработан в 1985 году Радией Перлман. Ей дали 1 неделю на разработку алгоритма. Она сделала это за 1 день, а в оставшееся время описала алгоритм в виде стихотворения

> I think that I shall never see
>
> A graph more lovely than a tree.
>
> A tree whose crucial property
>
> Is loop-free connectivity.
>
> A tree that must be sure to span
>
> So packets can reach every LAN.
>
> First, the root must be selected.
>
> By ID, it is elected.
>
> Least-cost paths from root are traced.
>
> In the tree, these paths are placed.
>
> A mesh is made by folks like me,
>
> Then bridges find a spanning tree.
>
> — Radia Joy Perlman

Вопросы?

Задача Штейнера

Нужно найти точку на плоскости, сумма расстояний от которой до вершин этого треугольника наименьшее. TA + TB + TC → min.

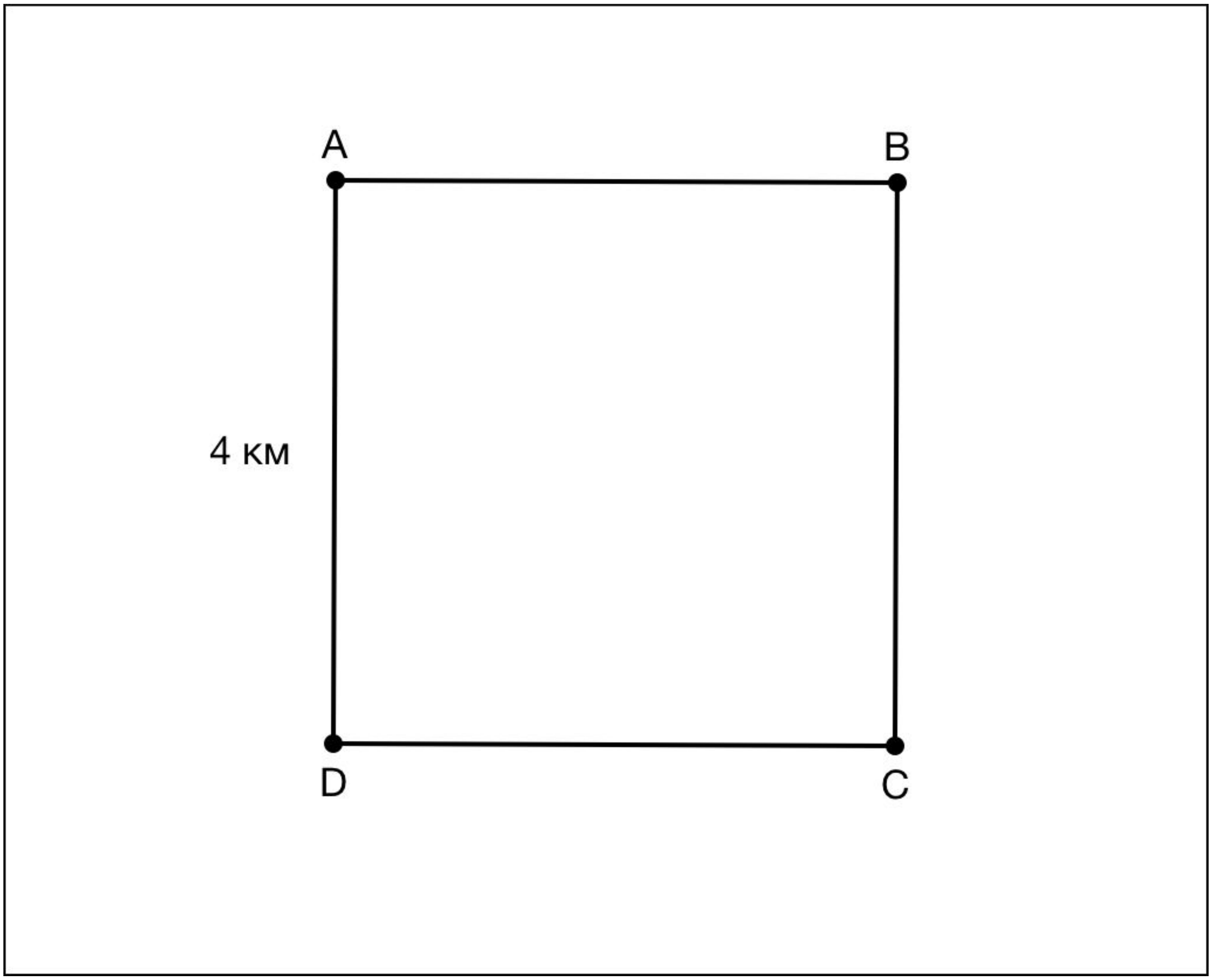


Что же это за точка? Эта точка называется по-разному: точкой Торичелли, точкой Ферма, точкой Штейнера. Представляет она собой точку, из которой все три вершины треугольника видны под одинаковыми углами – 120°.
Впервые решение задачи о точке с наименьшей суммой расстояний до всех вершин треугольника документально впервые появилось в книге итальянского физика и математика Винченцо Вивиани в середине XVII века. Однако известно, что решение этой задачи было получено еще раньше другом Вивиани, Эванджелиста Торичелли – оба они были учениками Галилея. Приведенное в книге решение не было геометрическим, оно было основано на физических принципах. Эту задачу до Торичелли знал, а возможно, и решил Пьер Ферма: они состояли в переписке, и про эту задачу Торичелли узнал именно от Ферма, но было ли у него свое решение – неизвестно.

Первое геометрическое решение этой задачи появилось лишь спустя 200 лет, и автором его стал Якоб Штейнер.

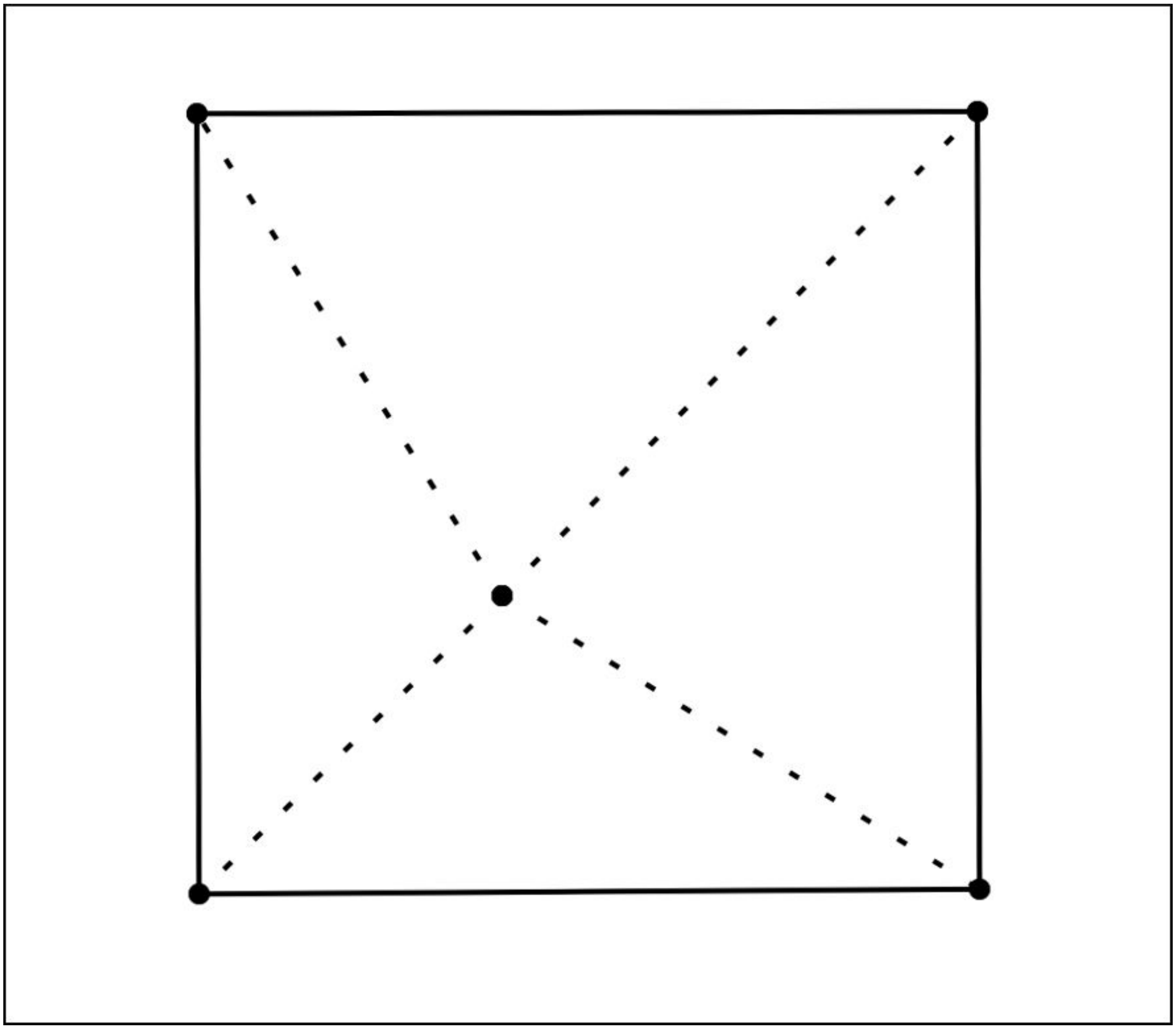
4 точки

Четыре деревни расположены в вершинах квадрата со стороной четыре километра. Существует ли система дорог, которая связывала бы все эти деревни между собой и имела бы суммарную длину не превосходящую 11 километров

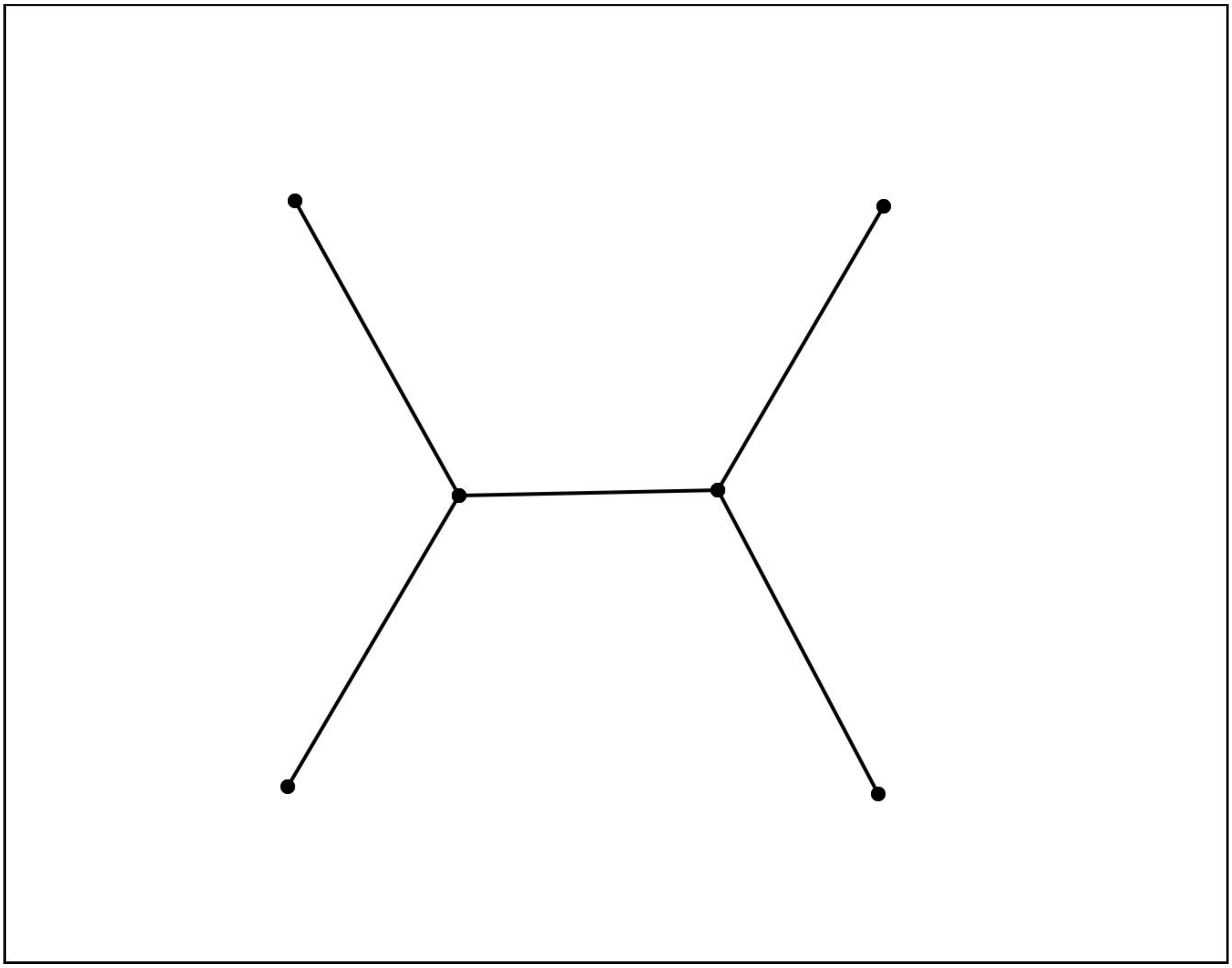


Если соединять деревни последовательно, то ничего короче, чем три стороны квадрата мы не придумаем. При таком соединении у нас будет ровно 12 километров. Можно соединять по диагонали, но это будет только хуже, т.к. диагональ длиннее стороны.

Если поставить дополнительную вершину, тогда из нее можно будет проехать в любые четыре деревни. Если мы возьмем дороги, связывающие диаметрально противоположные вершины, то по неравенству треугольника, сумма их длин будет больше, чем длина диагонали. У и двух других сумма длин тоже больше, чем длина диагонали. Соответственно, сумма всех дорог у нас будет $\geq 2 \cdot 4\sqrt{2} = 8\sqrt{2} \approx 11.31$.



Правильное решение представляет собой нечто похожее на букву Н, все углы в которой равны 120°. Ее длина составляет ≈10.98 километра. Это и есть сеть Штейнера, соединяющая четыре точки.



ЭВРИСТИЧЕСКИЙ АЛГОРИТМ ПОИСКА ПРИБЛИЖЕННОГО РЕШЕНИЯ ЗАДАЧИ ШТЕЙНЕРА, ОСНОВАННЫЙ НА ФИЗИЧЕСКИХ АНАЛОГИЯХ

Для построения алгоритма, находящего минимальное дерево Штейнера для заданного множества терминальных вершин, воспользуемся некоторыми известными свойствами деревьев Штейнера:.

1) Угол между любыми двумя рёбрами в дереве Штейнера всегда >=120°.

2) Ни одна вершина в дереве Штейнера не может быть инцидентна более чем трём рёбрам.

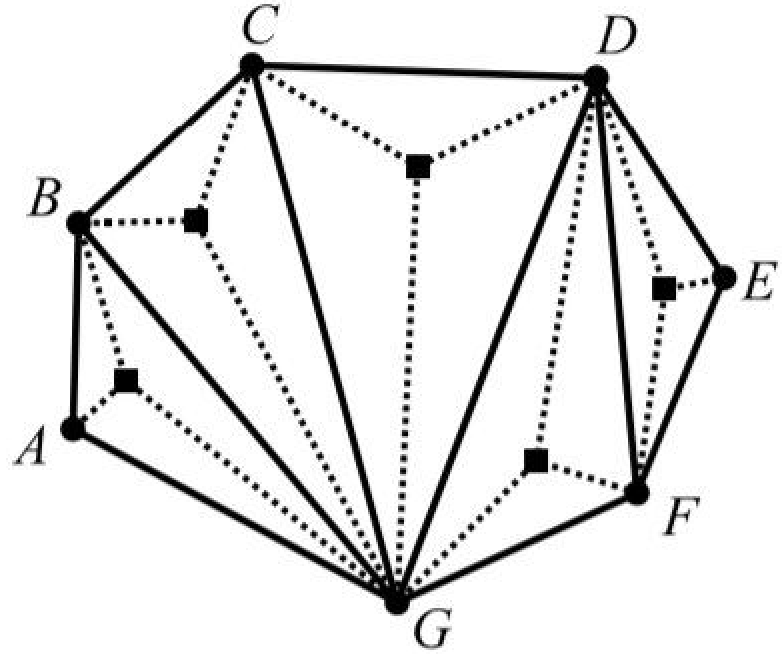
3) Количество точек Штейнера s <= n − 2 , где n — количество терминальных вершин.

4) Все точки Штейнера лежат в выпуклой оболочке, образованной вершинами 1 , , i a … a .

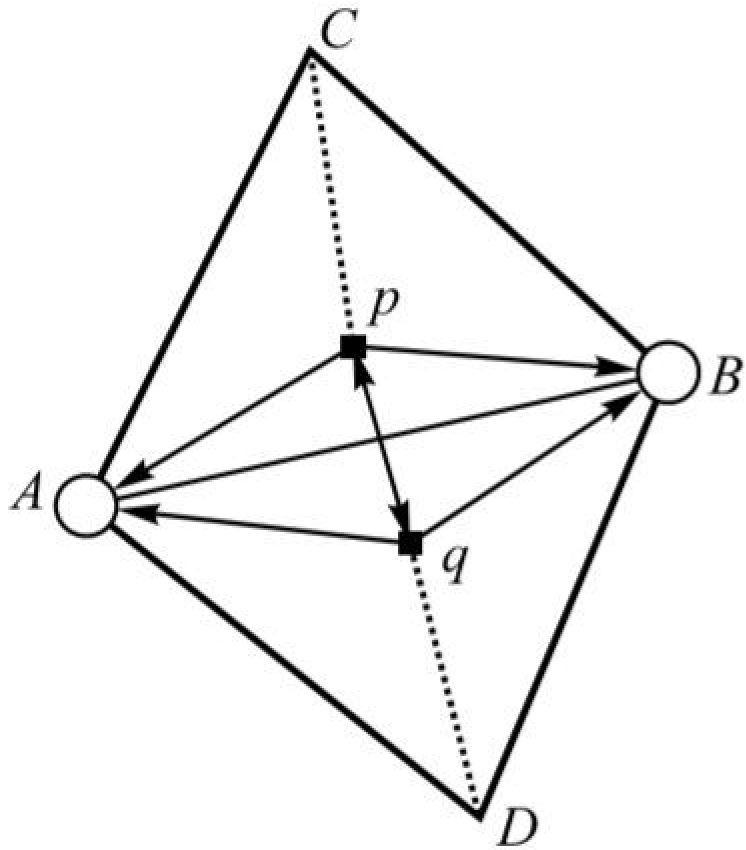
5) Каждая точка Штейнера соединена ребром как минимум с одной терминальной вершиной.

Для нахождения минимального дерева Штейнера воспользуемся механической моделью. Для этого помимо координат x и y припишем каждой точке начальную массу, т.е. фактически заменим множество Q геометрических точек множеством Q* материальных точек
Алгоритм поиска минимального дерева Штейнера состоит из трёх основных стадий:
1. Выбор начального приближения, что эквивалентно выбору начальных координат точек Штейнера.
2. Итеративный пересчёт координат точек Штейнера.
3. Обработка полученного результата.

1) Начальное приближение
   Для определения начального положения точек Штейнера используется триангуляция Делоне. Если все n терминальных вершин образуют выпуклую оболочку, получаем количество треугольников в триангуляции S = n− 2, что точно соответствует максимальному количеству точек Штейнера в минимальном дереве.



2) Итеративный пересчёт координат точек Штейнера.



$$F_i = \frac{m_{SHT} \cdot M}{r^2}$$

3) На графе, полученном с добавлением точек Штейнера запускается алгоритм Краскала.

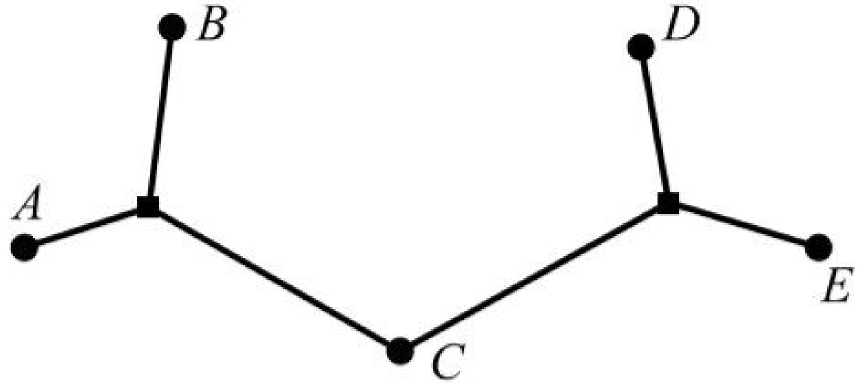


Рис. 3а. Дерево Штейнера, полученное с помощью гравитационной модели

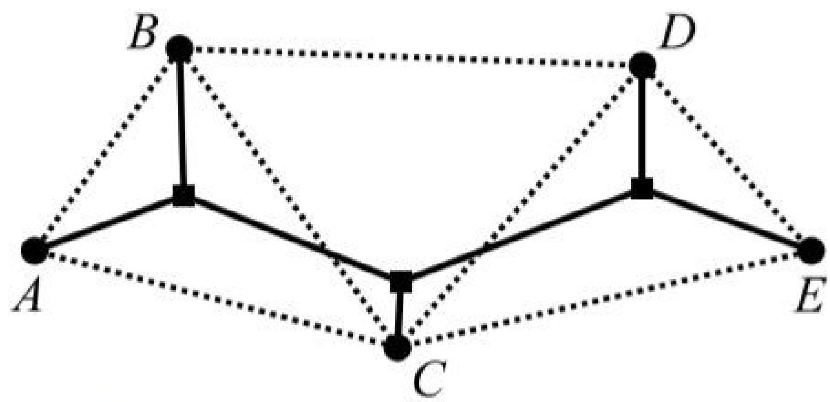


Рис. 3б. Минимальное дерево Штейнера