КОНЕЧНЫЕ ГРАФЫ И СЕТИ
Основные определения

Определение. Если на плоскости задать конечное множество V точек и конечный набор линий X, соединяющих некоторые пары из точек V, то полученная совокупность точек и линий будет называться **графом**.

При этом элементы множества V называются **вершинами** графа, а элементы множества X – **ребрами**.

В множестве V могут встречаться одинаковые элементы, ребра, соединяющие одинаковые элементы называются **петлями**. Одинаковые пары в множестве X называются **кратными** (или параллельными) ребрами.

Количество одинаковых пар (v, w) в X называется **кратностью** ребра (v, w).

Множество V и набор X определяют граф с кратными ребрами – **псевдограф**.

$$G = (V, X)$$

Псевдограф без петель называется **мультиграфом**.

Если в наборе X ни одна пара не встречается более одного раза, то мультиграф называется графом.

Если пары в наборе X являются упорядочными, то граф называется ориентированным или орграфом.

Графу соответствует геометрическая конфигурация. Вершины обозначаются точками (кружочками), а ребра – линиями, соединяющими соответствующие вершины.

Определение. Если x = {v, w} – ребро графа, то вершины v, w называются концами ребра x.

Если x = (v, w) – дуга орграфа, то вершина v – начало, а вершина w – конец дуги x.

Определение. Вершины v, w графа G = (V,X) называются **смежными**, если {v,w}?X. Два ребра называются **смежными**, если они имеют общюю вершину.

Определение. Степенью вершины графа называется число ребер, которым эта вершина принадлежит. Вершина называется **изолированной**, если если ее степень равна единице и **висячей**, если ее степень равна нулю.

Определение. Графы $G_1(V_1, X_1)$ и $G_2(V_2, X_2)$ называются **изоморфмными**, если существует взаимно однозначное отображение $\varphi: V_1 \rightarrow V_2$, сохраняющее смежность.

Определение. **Маршрутом (путем)** для графа G(V, X) называется последовательность $v_1\, x_1\, v_2\, x_2\, v_3 \ldots x_k\, v_{k+1}$. Маршрут называется **замкнутым**, если его начальная и конечная точки совпадают. Число ребер (дуг) маршрута (пути) графа называется **длиной** маршрута (пути).

Определение. Незамкнуты маршрут (путь) называется **цепью**. Цепь, в которой все вершины попарно различны, называется **простой цепью**.

Определение. Замкнутый маршрут (путь) называется **циклом (контуром)**. Цикл, в котором все вершины попарно различны, называется простым циклом.

5.1 Матрицы графов

Пусть D = (V, X) – орграф, где $V = \{v_1, \ldots, v_n\}$, $X = \{x_1, \ldots, x_m\}$.

Определение. **Матрицей смежности** орграфа D называется квадратная матрица $A(D) = [a_{ij}]$ порядка n, у которой

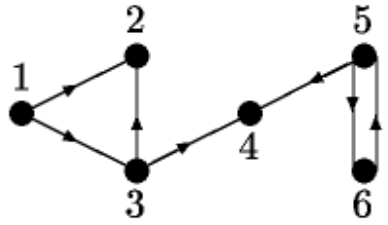
$$a_{ij} = \begin{cases} 1, & \text{если } (v_i, v_j) \in X \\ 0, & \text{если } (v_i, v_j) \notin X \end{cases}$$

Определение. Если вершина v является концом ребра x, то говорят, что v и x – инциндентны.

Определение. **Матрицей инциндентности** оргафа D называется матрица размерности n×m $B(D) = [b_{ij}]$, у которой

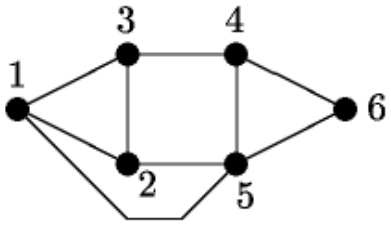
$$b_{ij} = \begin{cases} 1, & \text{если вершина } v_i \text{ является концом дуги } x_j \\ -1, & \text{если вершина } v_i \text{ является началом дуги } x_j \\ 0,, & \text{если вершина } v_i \text{ не инцидентна дуге } x_j \end{cases}$$

Пример матрицы инциденций и смежности для графов, изображенных на рисунках а) и б).



1	-1	-1	0	0	0	0	0
2	1	0	1	0	0	0	0
3	0	1	-1	-1	0	0	0
4	0	0	0	1	1	0	0
5	0	0	0	0	-1	-1	1
6	0	0	0	0	0	1	-1

а)



1	1	1	1	0	0	0	0	0	0
2	1	0	0	1	1	0	0	0	0
3	0	1	0	1	0	1	0	0	0
4	0	0	0	0	0	1	1	1	0
5	0	0	1	0	1	0	1	0	1
6	0	0	0	0	0	0	0	1	1

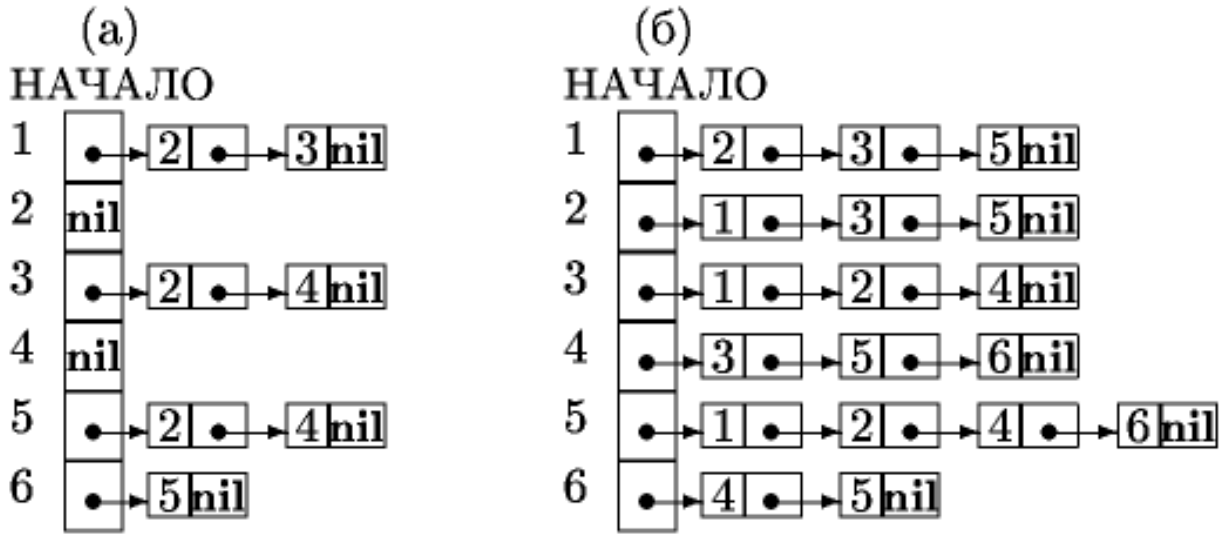
б)

Матрица смежности

(а) .	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

(б) .	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	1	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	1	0	1	0	1
6	0	0	0	1	1	0

Списки инцидентности



Если граф имеет кратные дуги (ребра), то в матрице смежности принимается aij k, где k – кратность дуги (ребра).

С помощью матриц смежности и инциндентности всегда можно полностью определеить граф и все его компоненты. Такой метод задания графов очень удобен для обработки данных на ЭВМ.

Таким образом, операции с графами можно свести к операциям с их матрицами.

5.2 Достижимость и связность

Определение. Вершина w графа D (или орграфа) называется **достижимой** из вершины v, если либо w=v, либо существует путь из v в w (маршрут, соединяющий v и w).

Определение. Граф (орграф) называется **связным**, если для любых двух его вершин существует маршрут (путь), который их связывает. Орграф называется **односторонне связным**, если если для любых двух его вершин по крайней мере одна достижима из другой.

Определение. Псевдографом D(V,X), ссоциированным с ориентированным псевдографом, называется псевдограф $G(V,X_0)$ в котором X_0 получается из X заменой всех упорядоченных пар (v, w) на неупорядоченные пары (v, w).

Определение. Орграф называется **слабо связным**, если связным является ассоциированный с ним псевдограф.

Поиск в глубину (***англ***. *Depth-first search,* ***DFS***)— один из методов обхода графа. Алгоритм поиска описывается следующим образом: для каждой не пройденной вершины необходимо найти все не пройденные смежные вершины и повторить поиск для них. Используется в алгоритмах поиска одно- и двусвязных компонент, топологической сортировки.

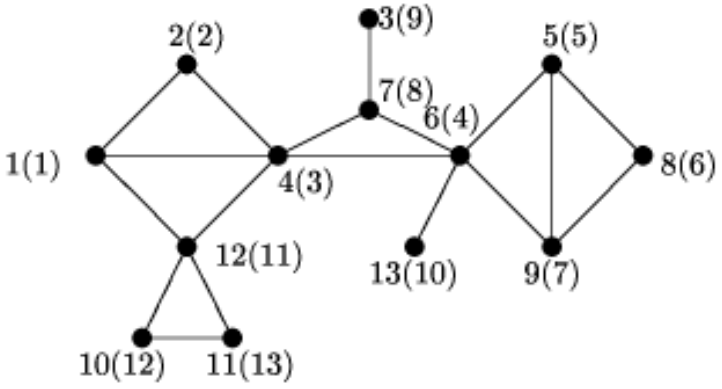
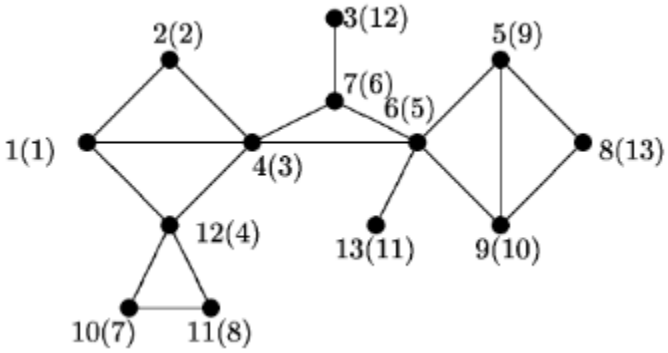


Рис. 2.5: Нумерация вершин графа (в скобках), соответствующая очередности, в которой они просматриваются в процессе поиска в глубину.

Поиск в ширину (**BFS**, Breadth-first search) — метод обхода и разметки вершин графа.

Поиск в ширину выполняется в следующем порядке: началу обхода s приписывается метка 0, смежным с ней вершинам — метка 1. Затем поочередно рассматривается окружение всех вершин с метками 1, и каждой из входящих в эти окружения вершин приписываем метку 2 и т. д.

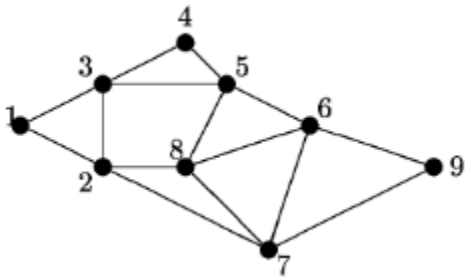


5.3 Эйлеровы и гамильтоновы графы

Определение. Цепь (цикл) в псевдографе G называется **эйлеровым**, если она проходит по одному разу через каждое ребро псевдографа G.

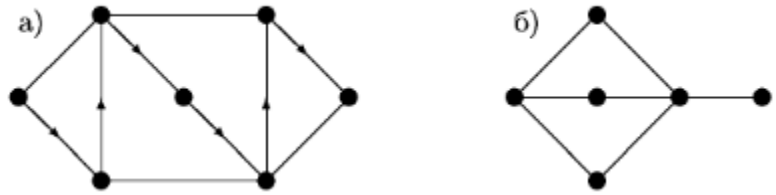
Теорема. *Для того, чтобы связный псевдограф G обладал эйлеровым циклом, необходимо и достаточно, чтобы степени его вершин были четными.*

Теорема. *Для того, чтобы связный псевдограф G обладал эйлеровой цепью, необходимо и достаточно, чтобы он имел ровно две вершины нечетной степени.*



1, 2, 3, 4, 5, 6, 7, 2, 8, 6, 9, 7, 8, 5, 3, 1

Определение. Цикл (цепь) в псевдографе G называется **гамильтоновым**, если он проходит через каждую вершину псевдографа G ровно один раз.



а) Гамильтонов путь в графе
б) Граф, в котором не существует гамильтонова пути

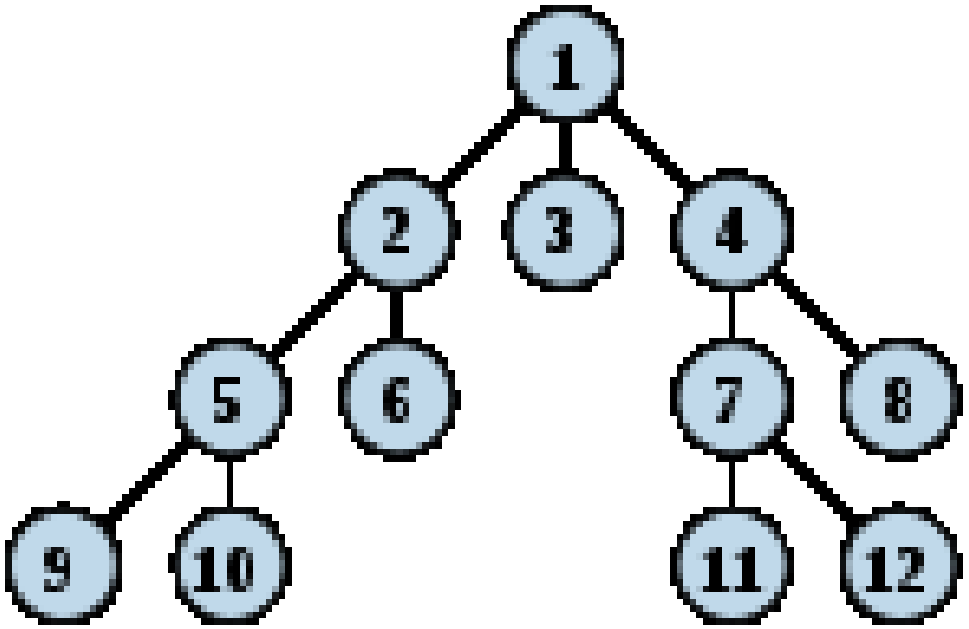
Граф G называется полным, если если каждая его вершина смежна со всеми остальными вершинами. В полном графе всегда существуют гамильтоновы цмклы.

Также необходимым условием существования гамильтонова цикла явояется связность графа.

5.4 Деревья и циклы

Определение. Граф G называется **деревом**, если он является связным и не имеет циклов. Граф G, все компоненты связности которого являются деревьями, называется лесом.

У графа, который является деревом, число ребер на единицу меньше числа вершин. Дерево не содержит циклов, любые две его вершины можно соеденить единственной простой цепью.



Если у дерева G есть, по крайней мере, одно ребро, то у него обязательно найдется висячая вершина, т.к. в противном случае в графе будет цикл.

Для графов, которые сами по себе не являются деревьями, вводится понятие остовного дерева.

Определение. Остовным деревом связного графа G называется любой его подграф, содержащий все вершины графа G и являющийся деревом.

Пусть G – связный граф. Тогда остовное дерево графа G (если оно существует) должно содержать n(G)-1 ребер.

Таким образом, любое остовное дерево графа G есть результат удаления из графа G ровно $m(G) - (n(G) - 1) = m(G) – n(G) + 1$ ребер.

Число $v(G) = m(G) - n(G) + 1$ называется цикломатическим числом связного графа G.

Одной из самых распространенных задач является задача построения остовного дерева минимальной длины графа. Для решения этой задачи применяется следующий алгоритм.

1) Выберем в графе G ребро минимальной длины. Вместе с инцидентными ему вершинами оно образует подграф G_2.
2) Строим граф G_3, добавляя к графу G_2 новое ребро минимальной длины, выбранное среди ребер графа G_2, каждое из которых инцидентно какой-либо вершине графа G_2, и одновременно инцидентно какой – либо вершине графа G, не содержащейся в графе G_2.
3) Строим графы G_4, G_5, …,n , повторяя действия пункта 2 до тех пор, пока не переберем все вершины графа G.

5.5 Алгоритмы поиска пути (Graph Search Algorithm)

Алгоритмы поиска пути, это алгоритмы на графе, которые позволяют найти путь из одной вершины графа в другую. Простейшие алгоритмы перебирают всевозможные варианты на основании некоторых дополнительных данных и выбирают оптимальный, более сложные алгоритмы используют эвристическую функцию для нахождения оптимальных переходов. В этом разделе будут рассмотрены:

- Двунаправленный поиск (Bidirectional Search)
- Поиск по первому наилучшему совпадению (Best-first Search)
- Алгоритм Дейкстры (Dijkstra`s algorithm)
- Алгоритм A* (A* algorithm)

Двунаправленный поиск

Двунаправленный поиск – это алгоритм поиска, который находит расстояние от исходной точки до конечной точки в ориентированном графе. Алгоритм запускает 2 одновременных поиск: от начальной точки к конечной и от конечной точки к начальной. Алгоритм заканчивает свою работу, когда оба поиска приходят в одну точку. Т.к. прямой и обратный поиск могут быть выполнены параллельно друг с другом в разных потоках приложения, чисто теоретически, это позволяет получить двукратное уменьшение времени затрачиваемого на поиск. Однако на практике существует ряд дополнительных проблем, которые связаны с таким ускорением алгоритма.

Алгоритм двунаправленный поиска должен содержать дополнительную логику, которая будет принимать решение по какой из дуг перейти на следующем шаге. От этого выбора сильно зависит успешность алгоритма. Конечная точка должна быть жёстко задана, но на практике часто возникает необходимость задать конечную точку какими либо параметрами, в этом случае алгоритм двунаправленного поиска в исходном виде не применим. Алгоритм двунаправленного поиска так же должен включать логику, которая будет эффективно определять пересечение двух деревьев поиска для нахождения общей точки, что сильно осложняет реализацию, особенно учитывая тот факт, что коэффициент ветвления обратного поиска может отличаться от коэффициента ветвления прямого поиска.

Такая сложность реализации наталкивает на мысль, что алгоритм поиска A* является лучшим решением в случае если мы может определить эффективную эвристическую функцию.

Однако, не смотря на все сложности реализации алгоритм двунаправленного поиска, зачастую работает эффективней, чем алгоритм A*. Тесты Иры Поль, проведённые на 15 элементной мозаике, показали, что алгоритм двунаправленного поиска позволяет найти кратчайшие пути по сравнению с алгоритмом A*, когда эвристическая функция подобрана не достаточно хорошо. К тому же, существует множество ситуаций когда алгоритм A* требует намного больше ресурсов компьютера для успешного завершения.

Поиск по первому наилучшему совпадению

Поиск по первому наилучшему совпадению – это алгоритм поиска, который исследует граф путём расширения наиболее перспективных узлов, выбираемых в соответствии с указанным правилом. На каждом шаге алгоритм должен осуществлять оценку целесообразности перехода в ту или иную вершину графа. В качестве оценки узла n принято считать значение некоторой «эвристической функции оценки $f(n)$, которая, вообще говоря, может зависеть от описания n, описания цели, информации собранной поиском на данный момент, и самое главное, на каких-либо дополнительных знаний о предметной области.

Эффективный выбор текущего лучшего кандидата для расширения поиска, как правило, реализуется с помощью очереди с приоритетом.

Алгоритм поиска A*, рассмотренный далее, является примером оптимального поиска по наилучшему совпадению. Алгоритмы поиска по наилучшему совпадению часто используются для поиска пути в комбинаторном поиске.

Алгоритм Дейкстры

Алгоритм Дейкстры— алгоритм на графах, изобретённый нидерландским ученым Э. Дейкстрой в 1959 году. Находит кратчайшее расстояние от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например его использует протокол OSPF для устранения кольцевых маршрутов.

Вариант 1. Дана сеть автомобильных дорог, соединяющих города Московской области. Некоторые дороги односторонние. Найти кратчайшие пути от города Москва до каждого города области (если двигаться можно только по дорогам).

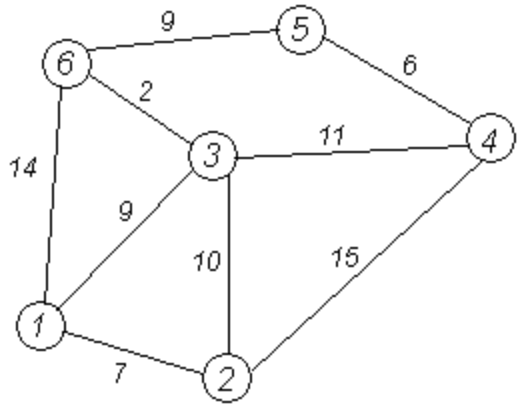
Вариант 2. Имеется некоторое количество авиарейсов между городами мира, для каждого известна стоимость. Стоимость перелёта из A в B может быть не равна стоимости перелёта из B в A. Найти маршрут минимальной стоимости (возможно, с пересадками) от Копенгагена до Барнаула.

Рассмотрим алгоритм. Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до α. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

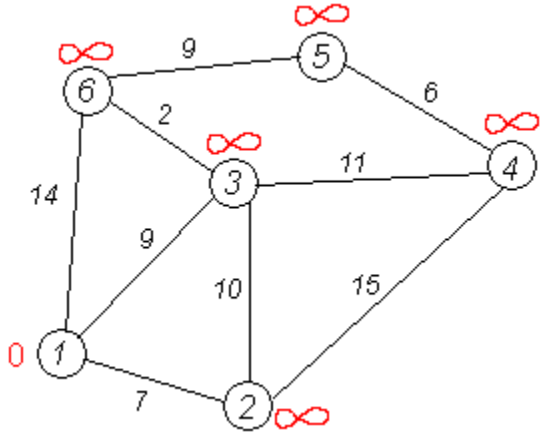
Инициализация. Метка самой вершины α полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от α до других вершин пока неизвестны. Все вершины графа помечаются как не посещённые.

Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовем *соседями* этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещенную и повторим шаг алгоритма.

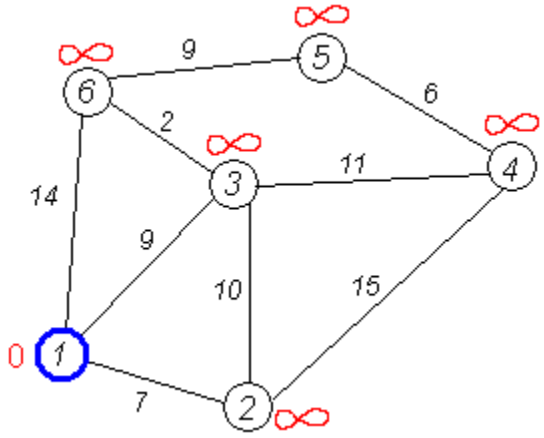
Рассмотрим выполнение алгоритма на примере графа, показанного на рисунке. Пусть требуется найти расстояния от 1-й вершины до всех остальных.



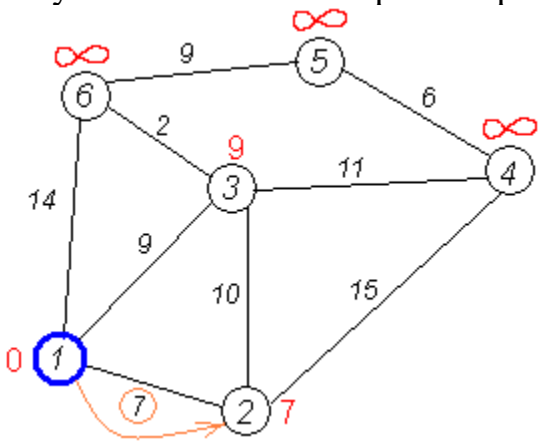
Кружками обозначены вершины, линиями — пути между ними (ребра графа). В кружках обозначены номера вершин, над ребрами обозначена их «цена» — длина пути. Рядом с каждой вершиной красным обозначена метка — длина кратчайшего пути в эту вершину из вершины 1.



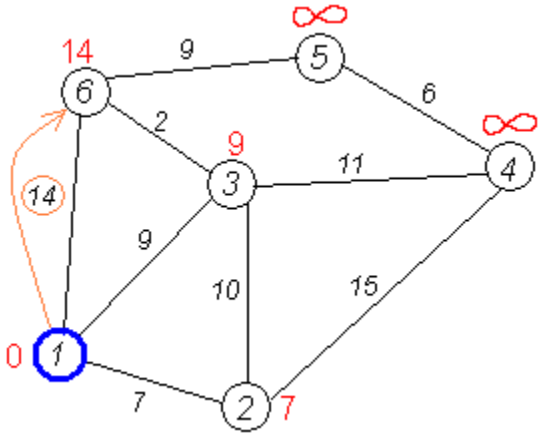
Первый шаг. Рассмотрим шаг алгоритма Дейкстры для нашего примера. Минимальную метку имеет вершина 1. Её соседями являются вершины 2, 3 и 6.



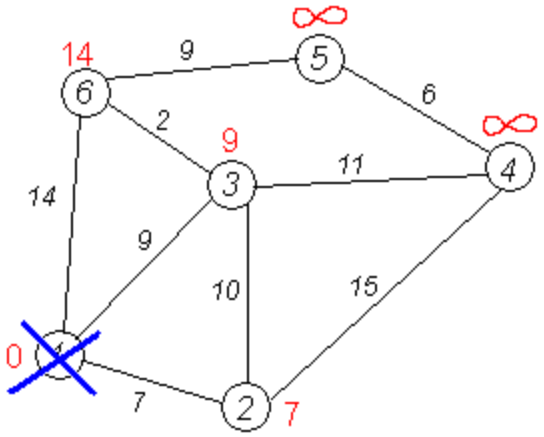
Первый по очереди сосед вершины 1 — вершина 2, потому что длина пути до неё минимальна. Длина пути в неё через вершину 1 равна сумме кратчайшего расстояния до вершины 1, значению её метки, и длины ребра, идущего из 1-ой в 2-ую, то есть 0 + 7 = 7. Это меньше текущей метки вершины 2, бесконечности, поэтому новая метка 2-й вершины равна 7.



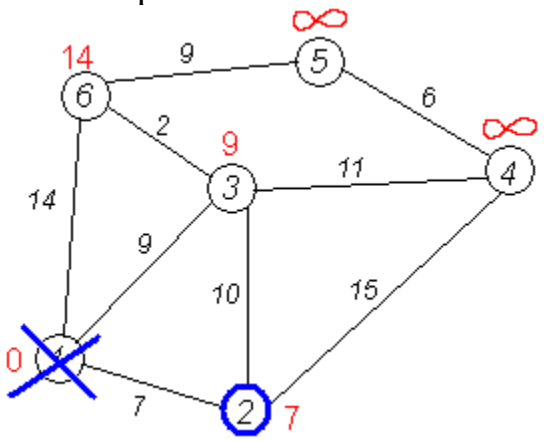
Аналогичную операцию проделываем с двумя другими соседями 1-й вершины — 3-й и 6-й.



Все соседи вершины 1 проверены. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит (то, что это действительно так, впервые доказал Э. Дейкстра). Вычеркнем её из графа, чтобы отметить, что эта вершина посещена.



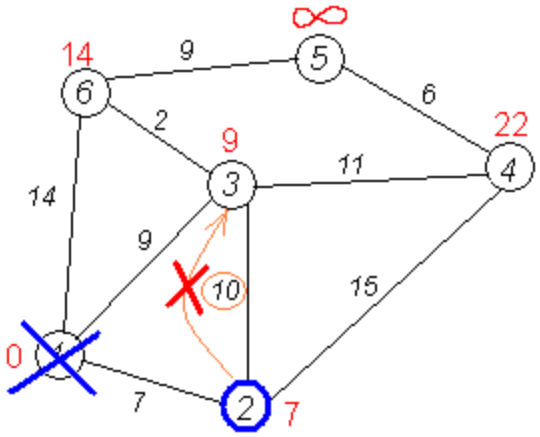
Второй шаг. Шаг алгоритма повторяется. Снова находим «ближайшую» из непосещенных вершин. Это вершина 2 с меткой 7.



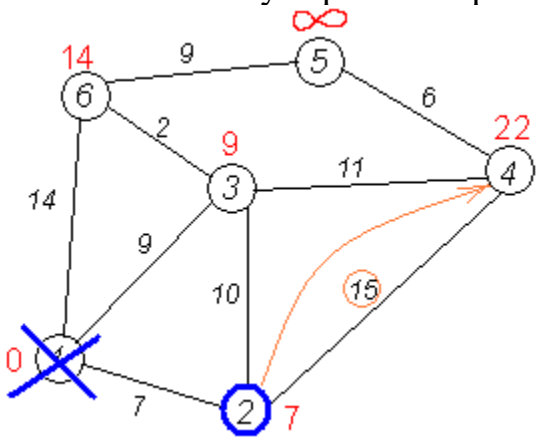
Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю вершину. Соседями вершины 2 являются вершины 1, 3 и 4.

Первый (по порядку) сосед вершины 2 — вершина 1. Но она уже посещена, поэтому с 1-й вершиной ничего не делаем.

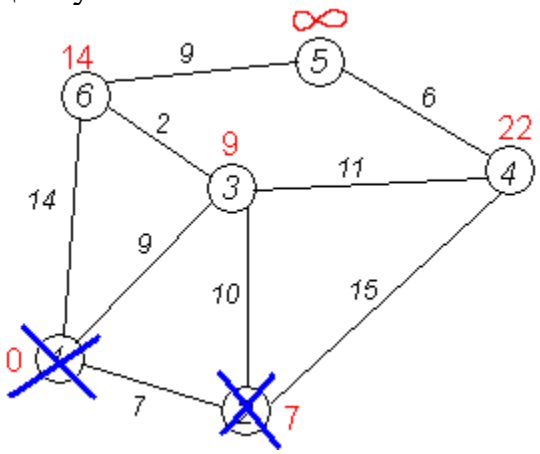
Следующий сосед вершины 2 — вершина 3, так как имеет минимальную метку из вершин, отмеченных как не посещённые. Если идти в неё через 2, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9<17, поэтому метка не меняется.



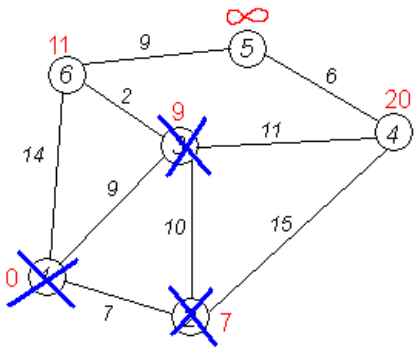
Ещё один сосед вершины 2 — вершина 4. Если идти в неё через 2-ю, то длина такого пути будет равна сумме кратчайшего расстояния до 2-ой вершины и расстояния между вершинами 2 и 4, то есть 22 (7 + 15 = 22). Поскольку 22<∞, устанавливаем метку вершины 4 равной 22.



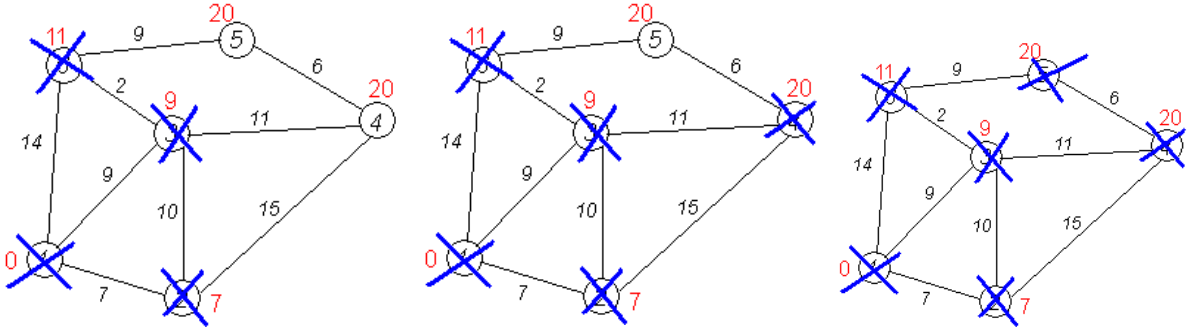
Все соседи вершины 2 просмотрены, замораживаем расстояние до неё и помечаем её как посещенную.



Третий шаг. Повторяем шаг алгоритма, выбрав вершину 3. После её «обработки» получим такие результаты:



Дальнейшие шаги. Повторяем шаг алгоритма для оставшихся вершин. Это будут вершины 6, 4 и 5, соответственно порядку.



Завершение выполнения алгоритма. Алгоритм заканчивает работу, когда вычеркнуты все вершины. Результат его работы виден на последнем рисунке: кратчайший путь от вершины 1 до 2-й составляет 7, до 3-й — 9, до 4-й — 20, до 5-й — 20, до 6-й — 11.

Рассмотрим псевдокод алгоритма Дейкстры:

```
function Dijkstra(Graph, source):
  for each vertex v in Graph:           // Initializations
    dist[v] := infinity ;               // Unknown distance function from source to v
    previous[v] := undefined ;          // Previous node in optimal path from source
  end for ;
  dist[source] := 0 ;                   // Distance from source to source
  Q := the set of all nodes in Graph ;
// All nodes in the graph are unoptimized - thus are in Q
  While Q is not empty:                 // The main loop
    u := vertex in Q with smallest dist[] ;
    if dist[u] = infinity:
      break ;// all remaining vertices are inaccessible from source
    end if ;
    remove u from Q ;
    for each neighbor v of u:           // where v has not yet been removed from Q.
      alt := dist[u] + dist_between(u, v) ;
```

```
        if alt < dist[v]:            // Relax (u,v,a)
            dist[v] := alt ;
            previous[v] := u ;
        end if ;
    end for ;
end while ;

  return dist[] ;
end Dijkstra.
```

АлгоритмA*

Алгоритм A* (произносится «А звезда» или «А стар», от англ.*Astar*) — в информатике и математике, алгоритм поиска по первому наилучшему совпадению на графе, который находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной).Порядок в котором будет происходить обход вершин определяется эвристической функцией которая зависит от расстояния до конечной цели и от стоимости перехода между вершинами графа. Обозначим эвристическую функцию как $f(x)$. Т.к. функция зависит и от расстояния и от стоимости перехода, представим $f(x)$как $f(x) = g(x) + h(x)$, где $g(x)$–функция стоимости достижения вершины xиз текущей вершины (может быть как эврестической так и нет), $h(x)$ – функция оценки расстояния из текущей вершины к вершине x (функция эвристическая).

Эвристическая функция расстояния *h(x)* должна давать приемлимую эвристическую оценку, т.е. не должна переоценивать расстояние. Например для задач нахождения кратчайшего пути эвристическая функция расстояния может представлять собой расстояние по прямой, т.к. расстояние по прямой это физически наименьшее возможное расстояние до цели.

A* пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный. Как и все информированные алгоритмы поиска, он просматривает сначала те маршруты, которые «кажутся» ведущими к цели. При выборе вершины A* учитывает, помимо прочего, *весь* пройденный до неё путь (составляющая *g(x)* — это стоимость пути от начальной вершины, а не от предыдущей). В начале работы просматриваются узлы, смежные с начальной; выбирается тот из них, который имеет минимальное значение *f(x)*, после чего этот узел раскрывается. На каждом этапе алгоритм оперирует с множеством путей из начальной точки до всех ещё не раскрытых (листовых) вершин графа («множеством частных решений»), которое размещается в очереди с приоритетом. Приоритет пути определяется по значению *f(x) = g(x) + h(x)*. Алгоритм продолжает свою работу до тех пор, пока значение *f(x)* целевой вершины не окажется меньшим,

чем любое значение в очереди (либо пока всё дерево не будет просмотрено). Из множественных решений выбирается решение с наименьшей стоимостью.

Чем меньше эвристика *h(x)*, тем больше приоритет (поэтому для реализации очереди можно использовать сортирующие деревья).

Псевдокод алгоритма:

```
function A*(start,goal)
    closedset := the empty set    // The set of nodes already evaluated.
    openset := set containing the initial node    // The set of tentative nodes to be evaluated.
    came_from := the empty map    // The map of navigated nodes.

    g_score[start] := 0    // Cost from start along best known path.
    h_score[start] := heuristic_cost_estimate(start, goal)
    f_score[start] := h_score[start]    // Estimated total cost from start to goal through y.

    while openset is not empty
        x := the node in openset having the lowest f_score[] value
        if x = goal
            return reconstruct_path(came_from, came_from[goal])

        remove x from openset
        add x to closedset
        foreach y in neighbor_nodes(x)
            if y in closedset
                continue
            tentative_g_score := g_score[x] + dist_between(x,y)

            if y not in openset
                add y to openset
                tentative_is_better := true
            else if tentative_g_score < g_score[y]
                tentative_is_better := true
            else
                tentative_is_better := false

            if tentative_is_better = true
                came_from[y] := x
                g_score[y] := tentative_g_score
```

```
            h_score[y] := heuristic_cost_estimate(y, goal)
            f_score[y] := g_score[y] + h_score[y]

    return failure
end function A*


 function reconstruct_path(came_from, current_node)
    if came_from[current_node] is set
       p = reconstruct_path(came_from, came_from[current_node])
       return (p + current_node)
    else
       return current_node
endfucntion reconstruct_path
```

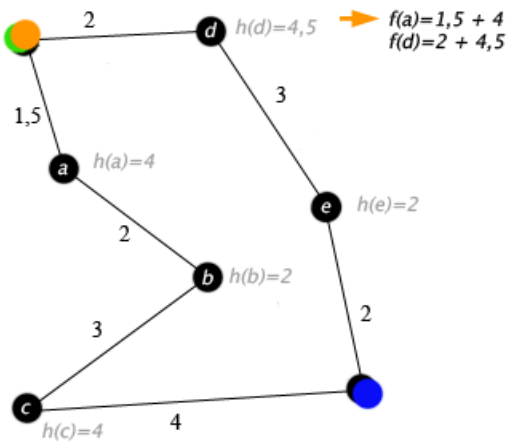
Дополнительную информацию по алгоритму A* можно найти в свободной энциклопедии Wikipedia: http://en.wikipedia.org/wiki/A*_search_algorithm

---

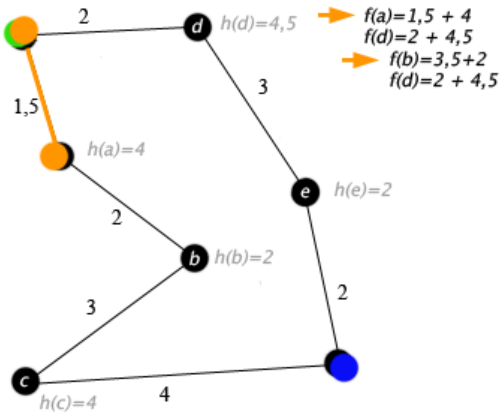
Дополнительнаяинформация:

Дополнительную информацию по алгоритму A* можно найти в свободной энциклопедии Wikipedia: http://en.wikipedia.org/wiki/A*_search_algorithm

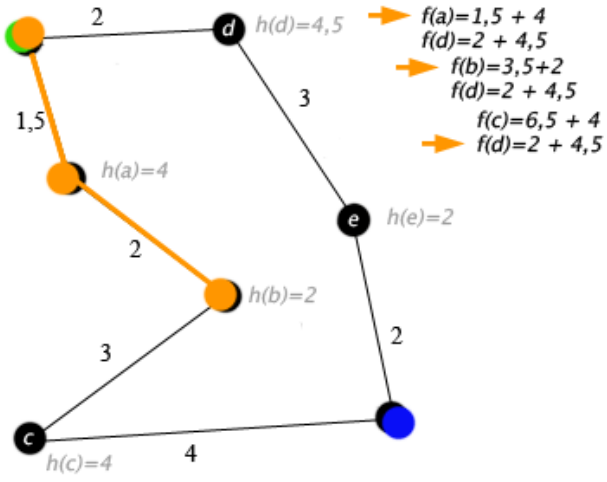
Рассмотрим пример работы алгоритма. В данном примере вершины графа это города, дуги – дороги их соединяющие. Рядом с дугами обозначена стоимость перехода. В случае дорог можно рассматривать стоимость перехода как показатель технического состояния дороги, переходы по дорогам с плохим технически состоянием осложнён. Чем выше стоимость перехода, тем хуже дорога. Оранжевая точка – исходная вершина, синяя точка – конечная вершина.



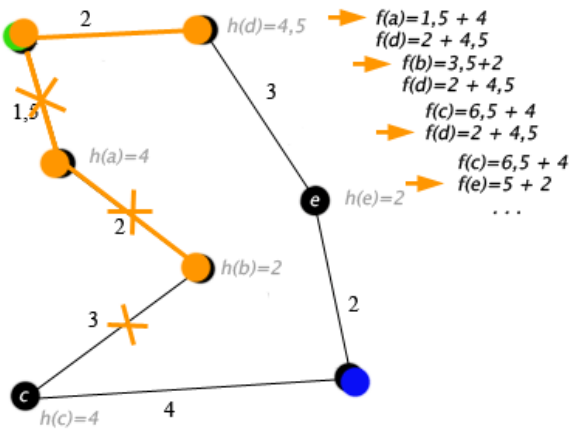
Шаг 1



Шаг 2



Шаг 3



Шаг 4.

Остовное дерево
(Spanning tree)

Остовным деревом связного графа называется граф, полученный из исходного путём удаления всех циклов. В остовное дерево входят все вершины исходного графа. Рёбра остовного дерева соединяют вершины таким образом, что в графе нет циклов, т.е. из любой вершины нельзя попасть в саму себя не пройдя какое-либо из рёбер дважды. Остовное дерево также иногда называют *покрывающим деревом*, *остовом* или *скелетом* графа.

Любое остовное дерево в графе с nвершинами содержит $n - 1$ ребро.

Остовное дерево может быть построено практически любым алгоритмом обхода графа, например поиском в глубину или поиском в ширину рассмотренными ранее. Остовные деревья, построенные при обходе графа с помощью алгоритма Дейкстры, начиная от вершины s, обладают тем свойством, что кратчайший путь в графе из sдо любой другой вершины - это единственный путь до этой вершины в построенном остовном дереве. Существует также несколько параллельных и распределённых алгоритмов нахождения остовного дерева. Как практический пример распределённого алгоритма можно привести протокол STP.

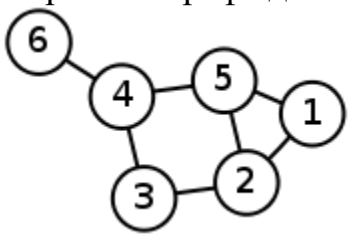
Если каждому ребру графа присвоен вес (длина, стоимость и т. п.), то нахождением оптимального остовного дерева, которое минимизирует сумму весов входящих в него рёбер, занимаются многочисленные алгоритмы нахождения минимального остовного дерева.

Матрица Кирхгофа

Для работы с остовными деревьями в теории графов графы пряно обозначать в виде матрицы Кирхгофа. Благодаря представлению графа в таком

виде, матрица Кирхгофа может использоваться вместе с теоремой Кирхгофа для расчета количества остовных деревьев в графе.

Матрица Кирхгофа это комбинация матрицы степеней вершин и матрицы смежности $L=A+B$. Матрица степеней вершин. А это матрица размерностью $nxn,$ где на главной диагонали расположены степени вершин графа (определение степени вершины графа дано в начале раздела).



Матрица степеней вершин для данного графа будет иметь вид :

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Матрица смежности для данного графа будет иметь вид (заметим, что для смежных вершин используется обозначение -1, а не 1 как в классическом варианте матрицы смежности, в этом особенность матрицы Кирхгофа):

$$B = \begin{bmatrix} 0 & -1 & 0 & 0 & -1 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Матрица Кирхгофа будет иметь вид:

$$L = A + B = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 & 0 & -1 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

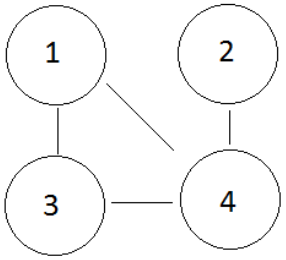
В общем виде матрица Кирхгофа записывается как:

$$l_{i,j} := \begin{cases} \deg(v_i), & \text{если } i = j \\ -1, \text{если } i \neq j \text{ и } v_i \text{ смежна с } v_j \\ 0, \text{иначе} \end{cases}$$

Теорема:

Пусть G — связный помеченный граф с матрицей Кирхгофа M. Все алгебраические дополнения матрицы Кирхгофа M равны между собой и их общее значение есть число остовных деревьев графа G.

Пример. Дан граф G, рассчитаем количество остовных деревьев графа:



Матрица Кирхгофа для данного графа будет иметь вид:

$$L = \begin{bmatrix} 3 & 0 & -1 & -1 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 2 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Найдём алгебраическое дополнение

$$M_{(1,1)} = (-1)^{1+1} * \begin{vmatrix} 1 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 3 \end{vmatrix} = 3$$

На основе данного графа может быть получено 3 остовных дерева: