Уральский федеральный университет

СТРУКТУРЫ ДАННЫХ

Лектор
Спиричева Наталия Рахматулловна

Ст. преподаватель каф. ИТ
Ауд. Р-246

Структуры данных

Составитель курса лекций:
Спиричева Наталия Рахматулловна,
ст. преподаватель каф. Информационных технологий

Структуры данных

Древовидные структуры данных

Структуры данных и алгоритмы

Целью лекции является приобретение студентами следующих компетенций:

- знать методы представления древовидных структур в памяти ЭВМ
- знать и уметь применять алгоритмы поиска путей в графах

Структуры данных и алгоритмы

Основные темы лекции:

- Понятие древовидных структур
- Деревья
- Графы
- Алгоритмы поиска путей в графах

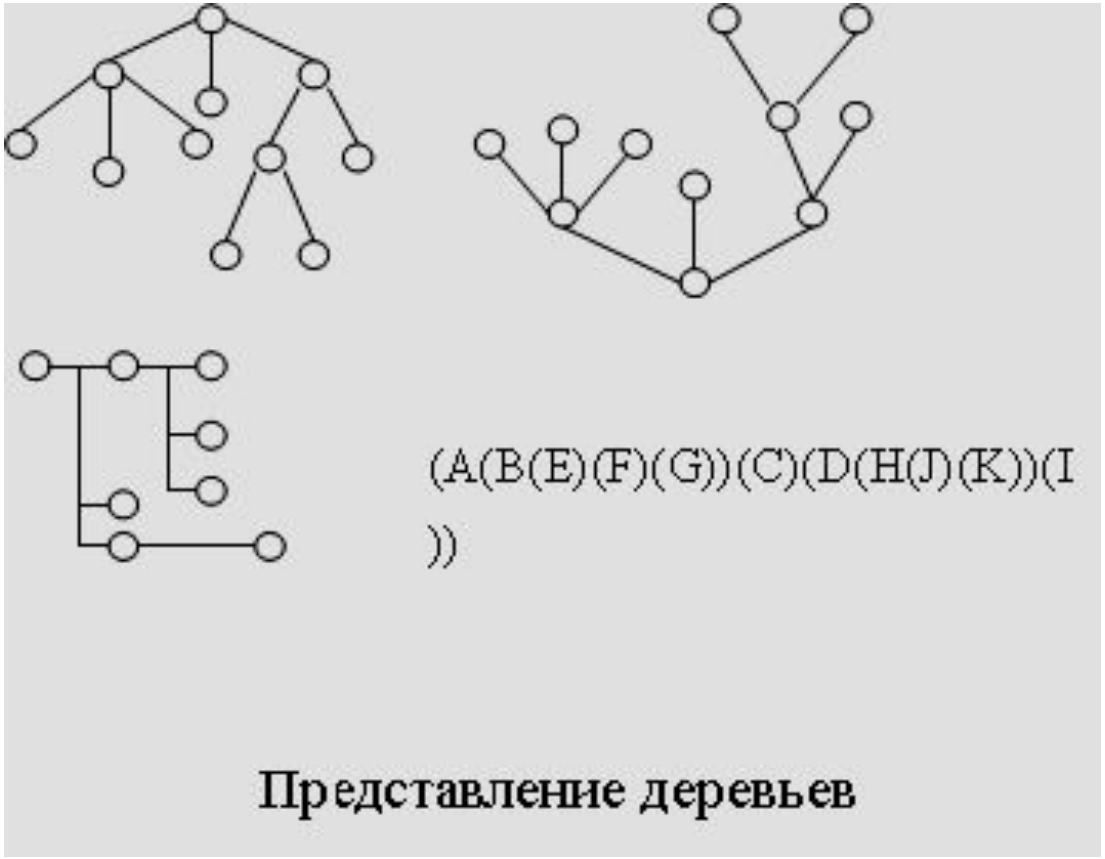
Введение

Дерево - конечное множество, состоящее из одного или более элементов, называемых узлами.

Корень - узел, не имеющий исходного. Все узлы, кроме корня, имеют только один исходный. Есть деревья, состоящие из одного корня. Каждый узел может иметь несколько порождённых.

Введение

Сетевые структуры – весьма общий и гибкий класс связных списков.



Представление деревьев

Введение

Определим *дерево* как конечное множество T, состоящее из одного или более *узлов*, таких, что

а) имеется один специально обозначенный узел, называемый *корнем* данного дерева,

б) остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно непересекающихся множествах T1, …, Tm, каждое из которых в свою очередь является деревом. Деревья T1, …, Tm называются *поддеревьями* данного корня.

Из определения следует:

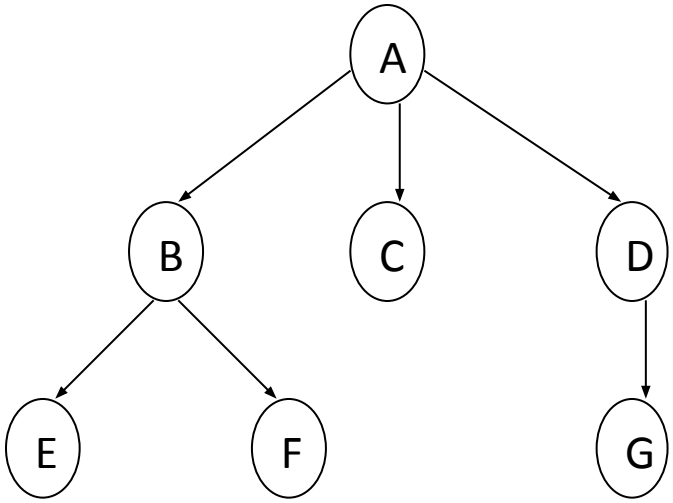
1. Каждый узел дерева является корнем некоторого поддерева, которое содержится в этом дереве.
2. Число поддеревьев данного узла называется степенью этого узла.
3. Узел с нулевой степенью называется *концевым узлом*;
4. Неконцевые узлы часто называют *узлами разветвления*.
5. *Уровень* узла по отношению к дереву T определяется следующим образом: говорят, что корень имеет уровень 1, а другие узлы имеют уровень на 1 выше их уровня относительно содержащего их поддерева Tj этого корня.

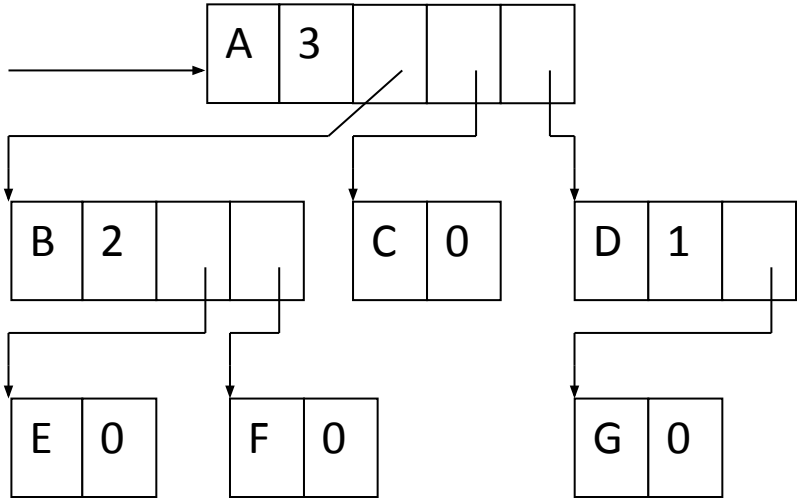
Введение

Обычно дерево представляется в машинной памяти в форме многосвязного списка, в котором каждый указатель соответствует дуге. Это представление называется естественным представлением дерева. Существуют несколько разновидностей такого представления. В одной из наиболее общих разновидностей каждому узлу дерева ставится в соответствие элемент многосвязного списка, причем в каждом элементе отводятся следующие поля: поле данных, поле степени исхода (т.е. числа сыновей) и поля указателей, число которых равно степени исхода.

Введение

Сетевые структуры – весьма общий и гибкий класс связных списков.





Введение

Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может равного нулю) числа непересекающихся деревьев.

Бинарное дерево

Бинарное дерево - конечное множество узлов, которое является пустым или состоит из корня и двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями данного корня.

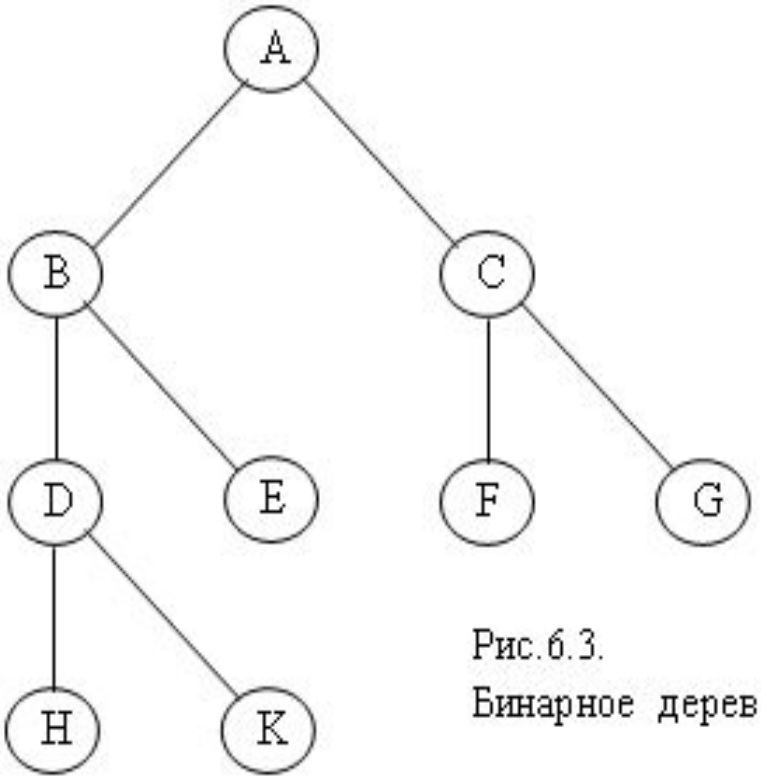


Рис.6.3.
Бинарное дерево

Бинарное дерево

В алгоритмах работы с древовидными структурами наиболее часто встречается понятие обход дерева.

Для обхода бинарных деревьев можно применить один из трех принципиально разных способов:

- в прямом порядке
- в центрированном порядке
- в обратном порядке

Бинарное дерево

Прямой порядок обхода:

Попасть в корень

Пройти левое поддерево

Пройти правое поддерево

Центрированный порядок обхода:

Пройти левое поддерево

Попасть в корень

Пройти правое поддерево

Обратный порядок обхода:

Пройти левое поддерево

Пройти правое поддерево

Попасть в корень

Бинарное дерево

“Прошитые” деревья

В “прошитых” деревьях концевые связи-указатели используются для связи с родителями, такие связи назвали нитями.

Отличие нормальных связей от нитей: в каждом узле хранят две однобитовые переменные *LTag* и *RTag*. Эти переменные равны нулю, если соответствующие связи указывают на поддеревья, и единице, если связи являются нитями.

Левая нить каждого узла указывает на узел, являющийся предшественником данного при центрированном обходе, правая - на узел, являющийся последователем данного узла.

Деревья

Графы

Графы

Граф - некоторое множество точек (называемых вершинами) и некоторое множество линий (называемых ребрами), соединяющих определенные пары вершин. Каждая пара вершин соединяется не больше чем одним ребром.

Графы:

1. взвешенные и невзвешенные
2. ориентированные и неориентированные

Графы

Каждая пара вершин в графе соединяется не больше чем одним ребром. Дуга, соединенная с вершиной, называется *инцидентной этой вершине*. Две вершины называются *смежными*, если существует ребро, соединяющее их. Две дуги называются *смежными*, если они инцидентны одной и той же вершине.

Графы

Пусть *V* и *V`* - вершины и пусть $n \geq 0$; говорят, что «*V0, V1, …, Vn*» - *путь* длины *n* от *V* до *V`*, если *V=V0*, вершина *Vk* смежна с *Vk+1* при $0 \leq k < n$, а *Vn=V`*. Путь *прост*, если вершины *V0, V1, …, Vn-1* все различны между собой, а также различны все вершины *V1, V2, …, Vn*. Граф называется *связным*, если имеется путь между каждыми двумя вершинами этого графа. *Циклом* называется простой путь длины не менее 3 от какой-либо вершины до нее самой. Свободное дерево – это связный граф, не имеющий циклов.

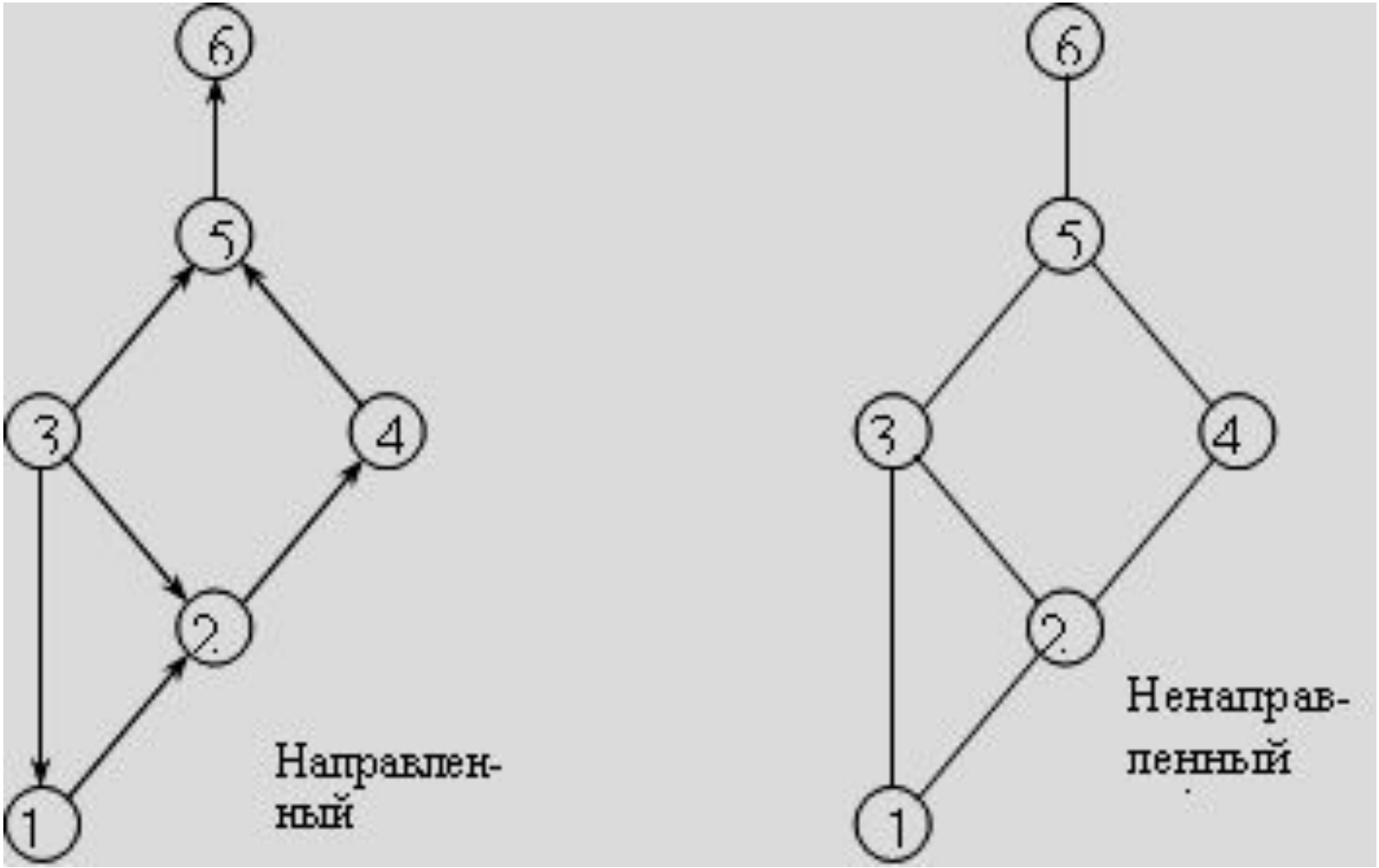
Графы

Граф называется *связным*, если имеется путь между каждыми двумя вершинами этого графа.

Циклом называется простой путь длины не менее 3 от какой-либо вершины до нее самой. Свободное дерево – это связный граф, не имеющий циклов.

Формально *направленный граф* определяется как некое множество вершин и множество *дуг*, причем каждая дуга ведет от некоторой вершины *V* к некоторой вершине *V`*. Если *e* – дуга, идущая от *V* к *V`*, то говорят, что *V* – *начальная* вершина дуги *e*, а *V`* - *конечная* вершина.

Графы



Графы

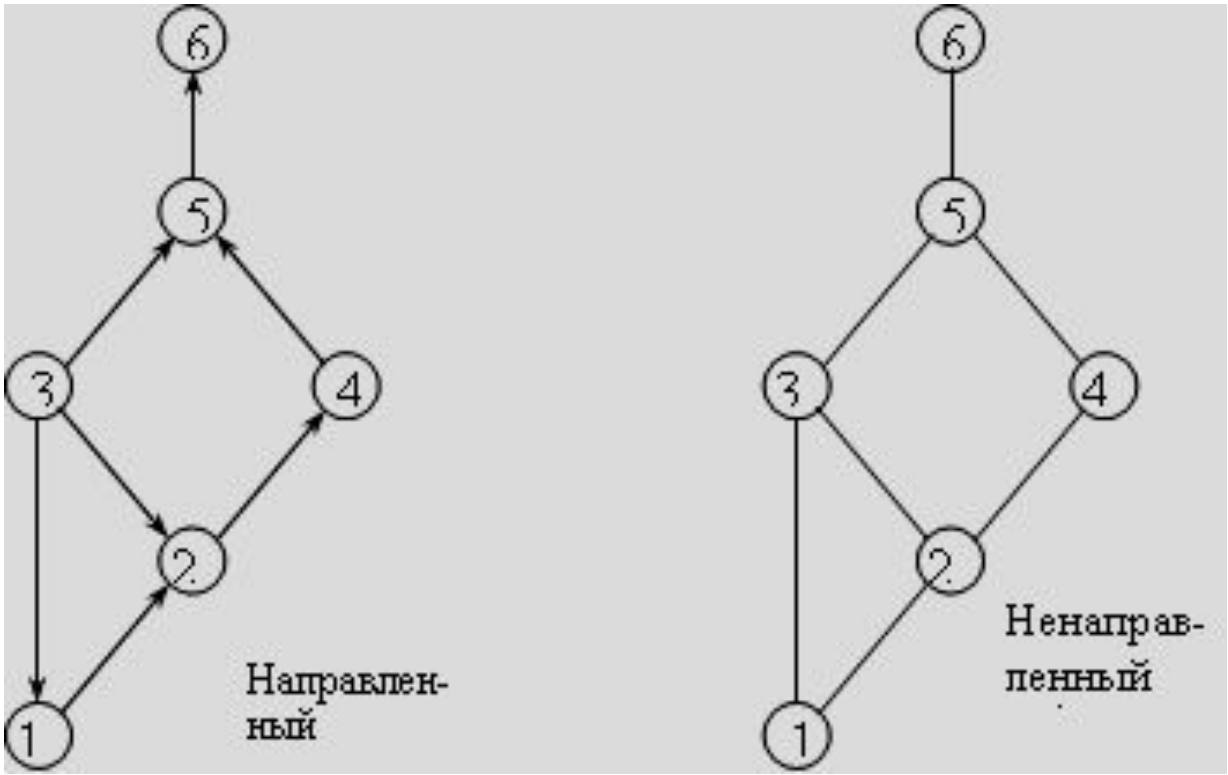
Задание графа:

Класс матриц инциденции. Если граф Г содержит *n* вершин и *m* дуг, то матрица инциденции A(Г)=[aij]mxn определяется так:

$$a_{ij} = \begin{cases} 1, \text{ если вершина } v_j - \text{начало дуги } u_i; \\ -1, \text{ если вершина } v_j - \text{конец дуги } u_i; \\ 0, \text{ если дуга } u_i \text{ не инцидентна вершине } v_j. \end{cases}$$

Графы

$$A(\Gamma) = \begin{Vmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{Vmatrix}$$



Графы

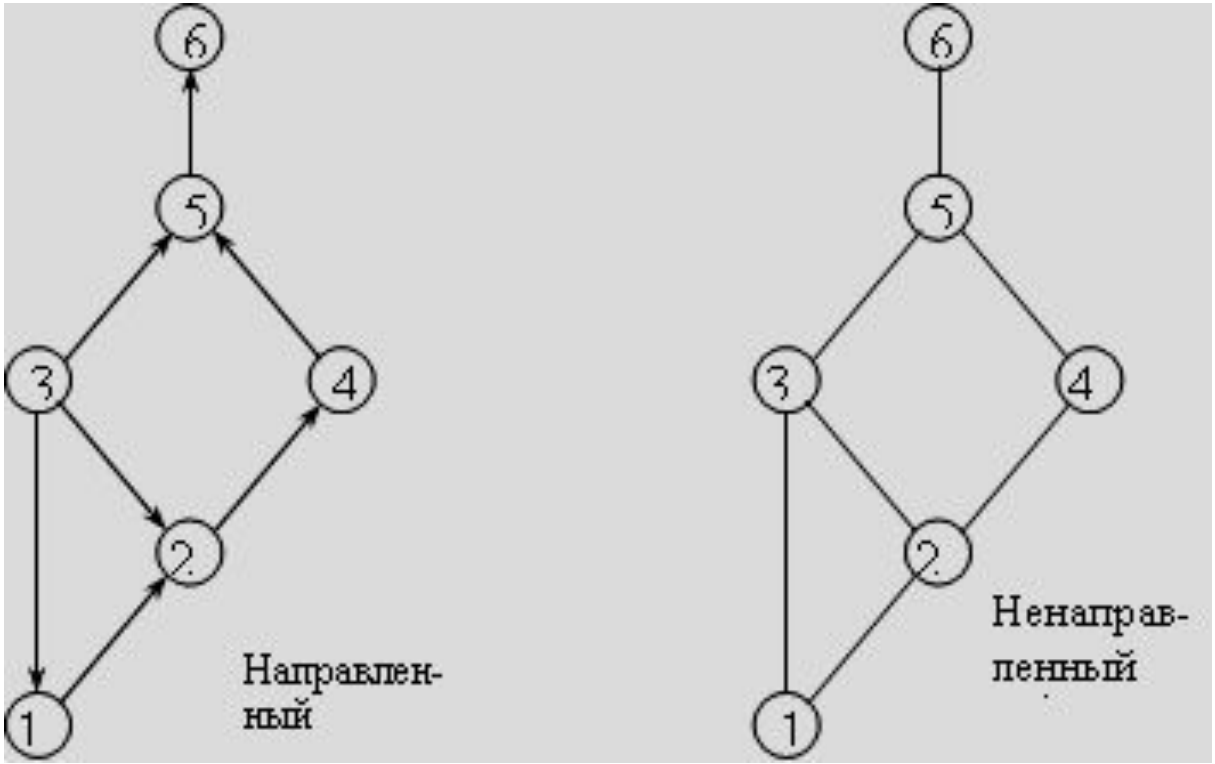
Класс матриц смежности. Матрица смежности S=[sij]nxm невзвешенного графа определяется следующим образом:

$$S_{ij} = \begin{cases} 1, \text{ если } v_i \text{ смежна } v_j; \\ 0, \text{ если вершины несмежны.} \end{cases}$$

Во взвешенном графе каждая единица заменяется на вес соответствующего ребра

Графы

$$S(\Gamma) = \begin{Vmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{Vmatrix}$$



Деревья

Алгоритмы поиска путей в графе

Алгоритмы поиска путей в графе

Путь с минимальным количеством промежуточных вершин

Алгоритм просматривает вершины графа в таком порядке:

- соединённые с исходной вершиной
- соединённые с уже просмотренными, но ещё не просмотренные.

Волновой алгоритм

1. Каждой вершине i приписывается два целых числа Times[i] - временная метка и Previous [i] - метка предыдущей вершины пути (начальное значение Times[i]=0, Previous [i]=0 для всех i).
2. Заводятся два списка "фронта волны" NewFront и OldFront, а также переменная Time (текущее время).
3. OldFront:={ver1}; NewFront:={}; Time:=1.
4. Для каждой из вершин i, входящих в OldFront, просматриваются соседние вершины j, и если Times [j] = 0, то Times [j]=Time, NewFront := NewFront + {j}; в переменную Previous [j] заносится номер i.
5. Если NewFront = {}, то путь не существует, переход к шагу 8.
6. Если одна из вершин совпадает с ver2, то найден кратчайший путь длины Time, переход к шагу 8.
7. OldFront:= NewFront; NewFront:={}; Time:=Time+1; возврат к шагу 4.
8. Восстанавливаем путь, проходя массив P.

Под корректностью алгоритма здесь понимается, что:

1. алгоритм завершает работу за конечное время;
2. если решение существует, то алгоритм находит правильное решение.

Алгоритмы поиска путей в графе

Путь минимальной суммарной длины во взвешенном графе с неотрицательными весами (*алгоритм Дейкстры*)

Функция находит путь минимального веса в графе *G=(V,E),* заданном весовой матрицей *w*, у которой элемент *wi j* равен весу ребра, соединяющего *i*-ю и *j*-ю вершины. При этом предполагается, что все элементы *wi j* неотрицательны. Путь ищется из вершины номер *u1* к вершине номер *u2*. Функция использует алгоритм Дейкстры. Для бесконечности используется число *GM,* его можно задавать в зависимости от конкретной задачи.

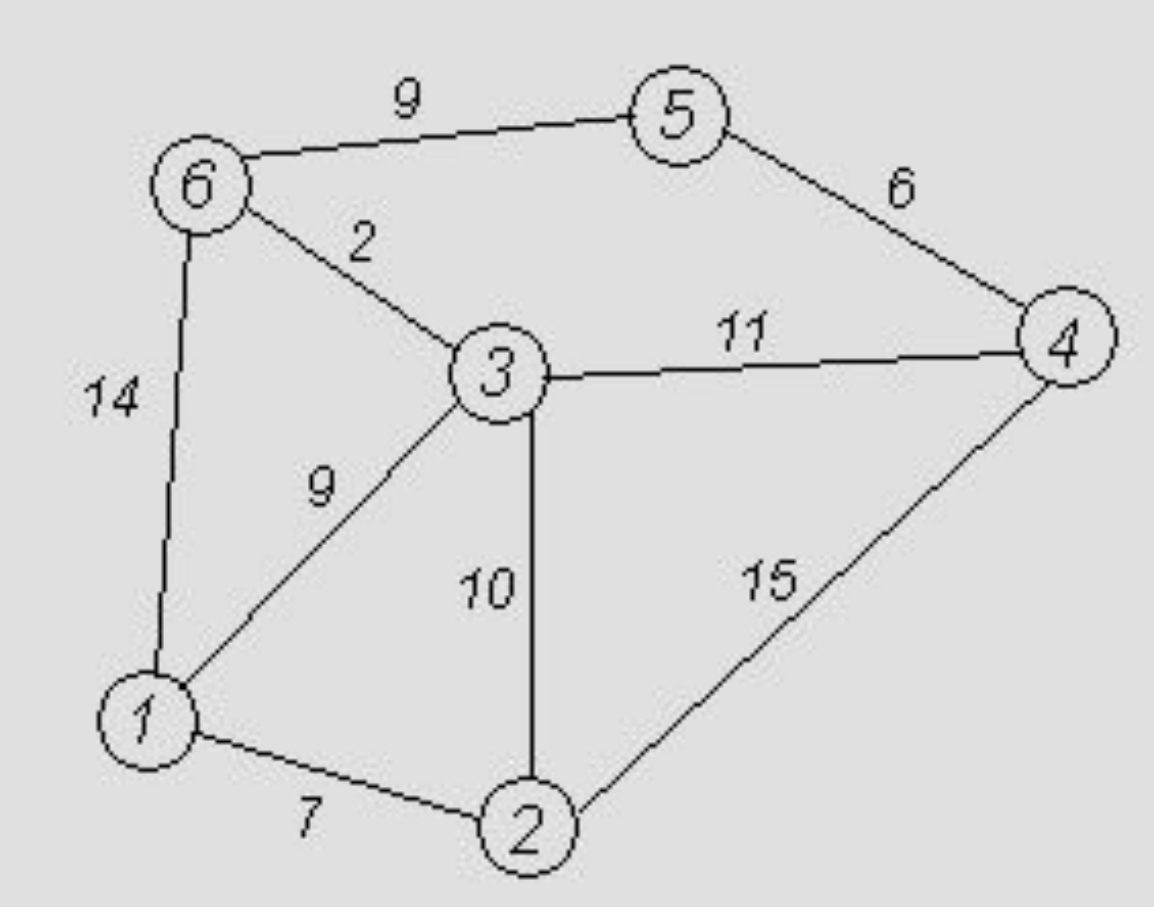
Алгоритмы поиска путей в графе

Алгоритм, по которому происходит поиск, заключается в следующем:

- всем вершинам приписывается вес - вещественное число, *d(i)=GM* для всех вершин кроме вершины с номером *u1*, а *d(u1)=0*;
- всем вершинам приписывается метка *m(i)=0*;
- вершина *u1* объявляется текущей - *t=u1;*
- для всех вершин у которых *m(i)=0*, пересчитываем вес по формуле: *d(i):=min{d(i), d(t)+W[t,i]};*
- среди вершин для которых выполнено *m(i)=0,* ищем ту, для которой *d(i)* минимальна, если минимум не найден, т.е. вес всех “непомеченных” вершин равен бесконечности (*GM*), то путь не существует; ВЫХОД;
- иначе найденную вершину с минимальным весом полагаем текущей и помечаем (*m(t)=1*);
- если t = u2, то найден путь веса d(t),ВЫХОД;
- переходим на шаг 4

Алгоритмы поиска путей в графе

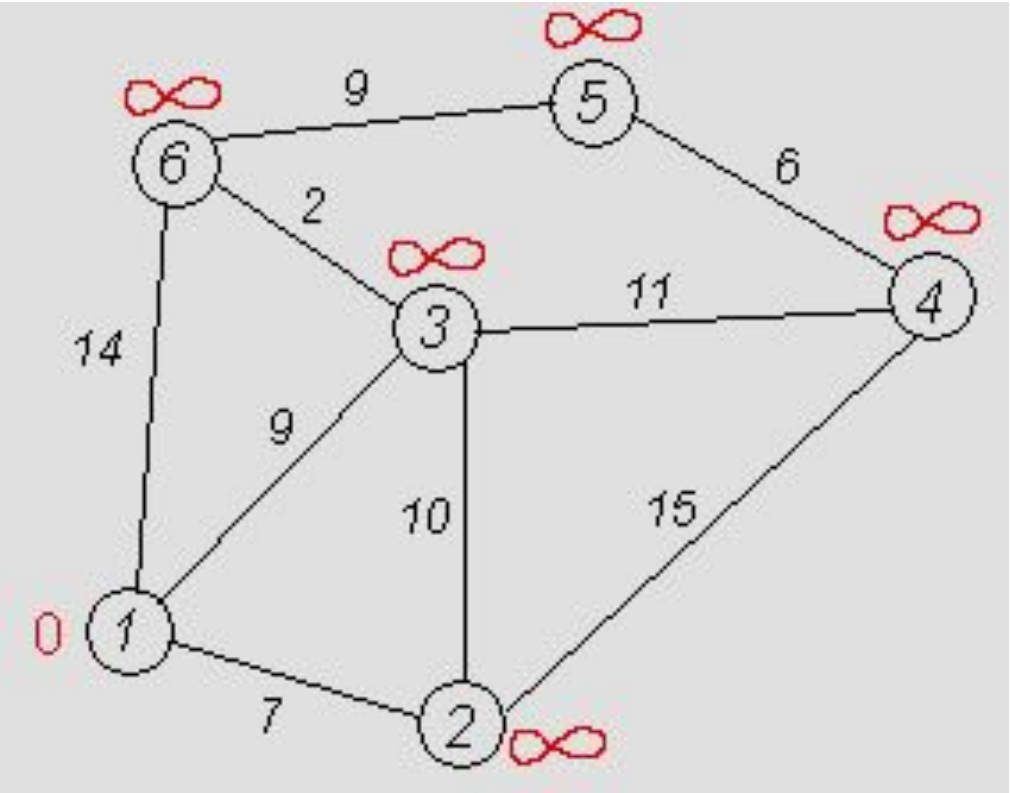
Алгоритм Дейкстры (пример)



Пусть требуется найти расстояния от 1-й вершины до всех остальных.

Алгоритмы поиска путей в графе

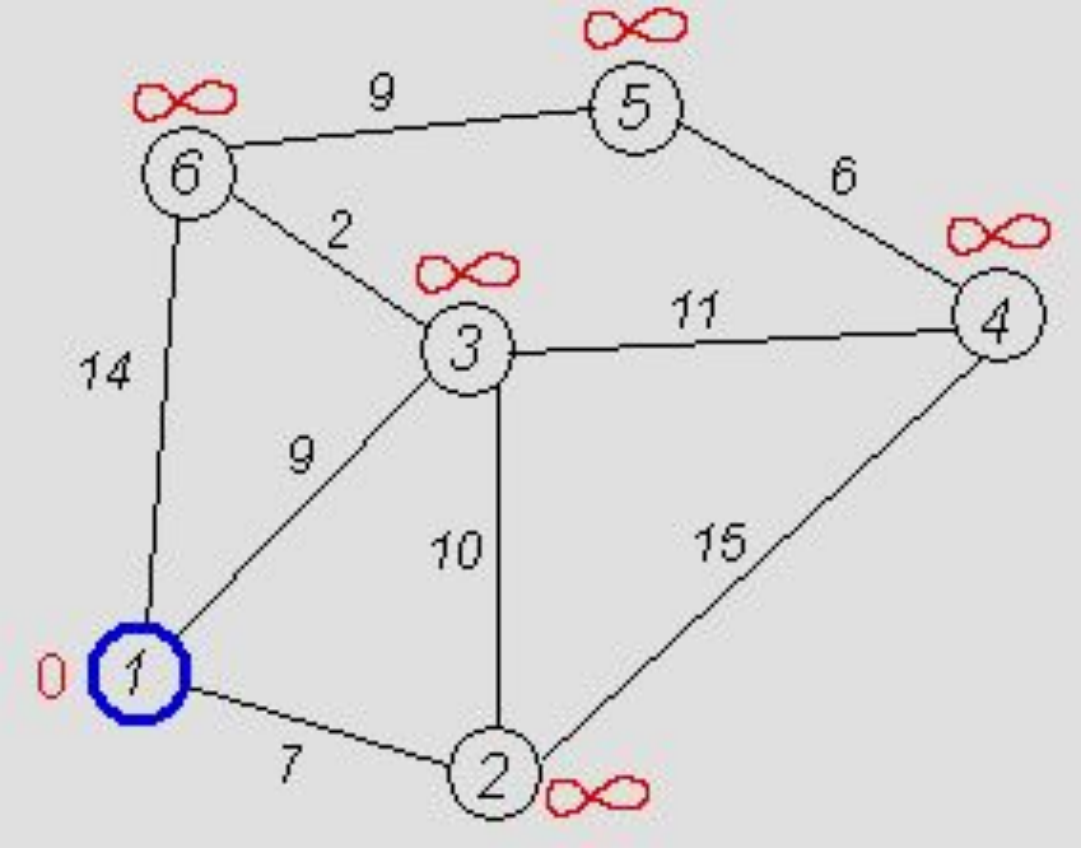
Алгоритм Дейкстры (пример)



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.

Алгоритмы поиска путей в графе

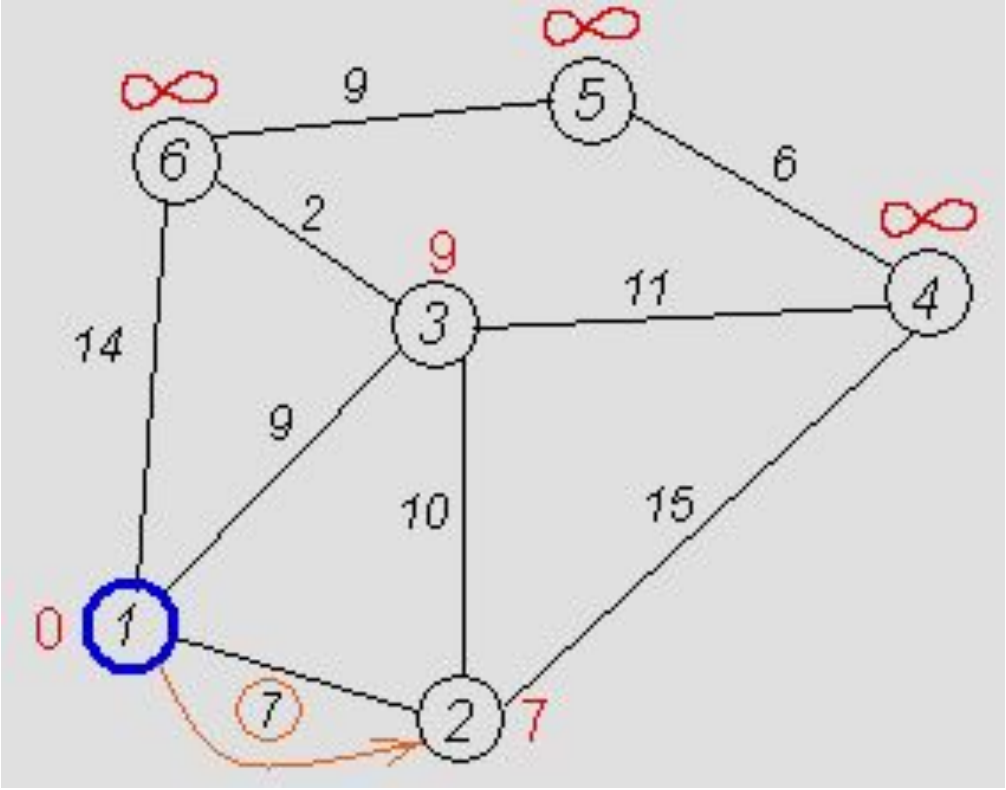
Алгоритм Дейкстры (пример)



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

Алгоритмы поиска путей в графе

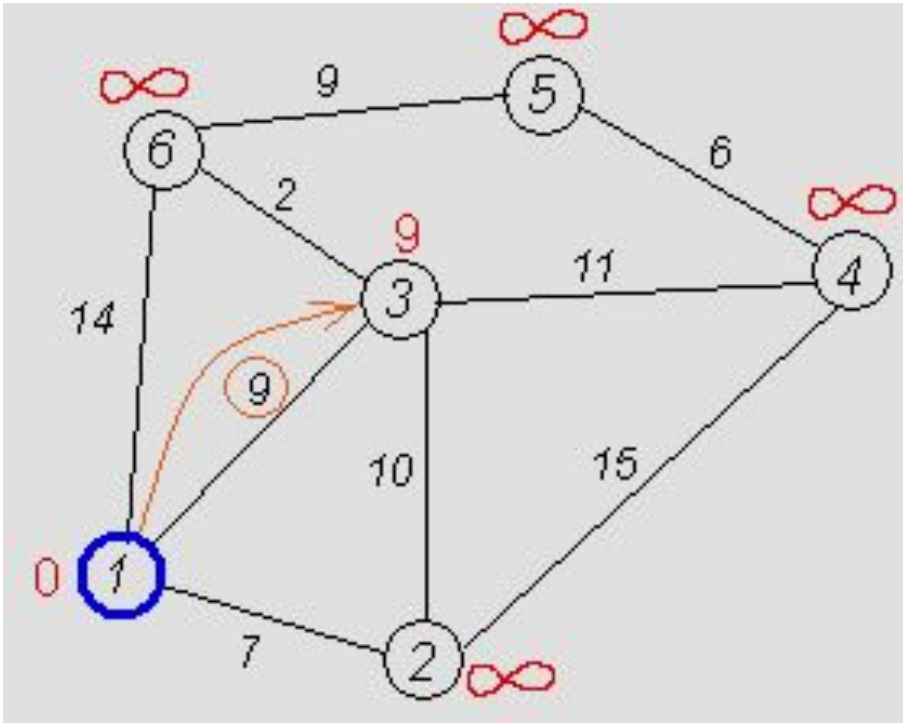
Алгоритм Дейкстры (пример)

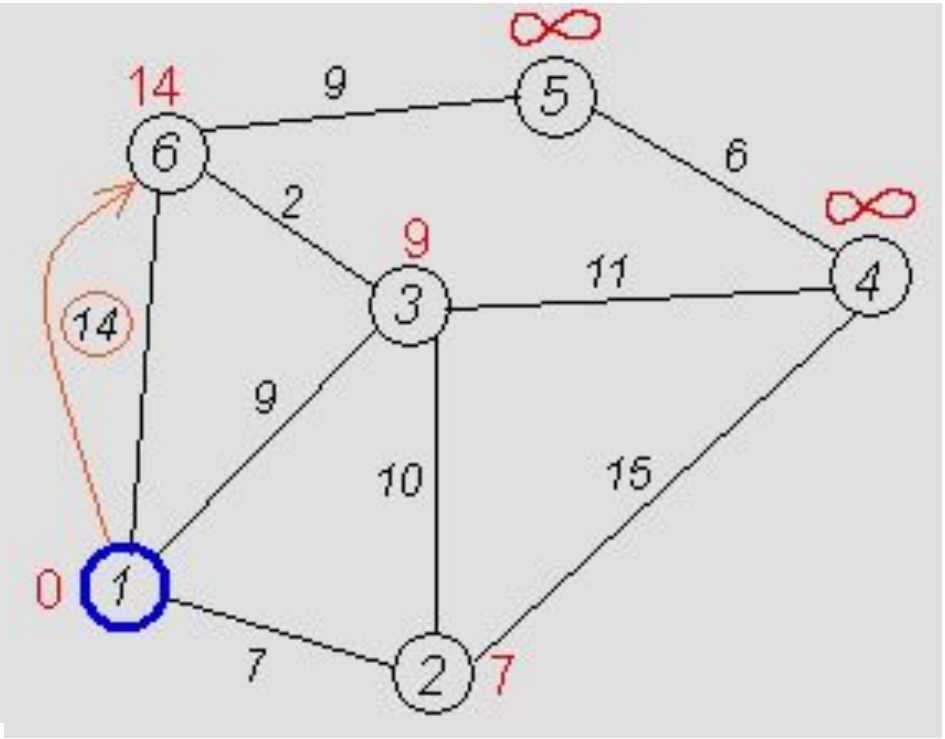


Первый по очереди сосед вершины 1 — вершина 2, потому что длина пути до неё минимальна. Длина пути в неё через вершину 1 равна кратчайшему расстоянию до вершины 1 + длина ребра, идущего из 1 в 2, то есть 0 + 7 = 7. Это меньше текущей метки вершины 2, поэтому новая метка 2-й вершины равна 7.

Алгоритмы поиска путей в графе

Алгоритм Дейкстры (пример)

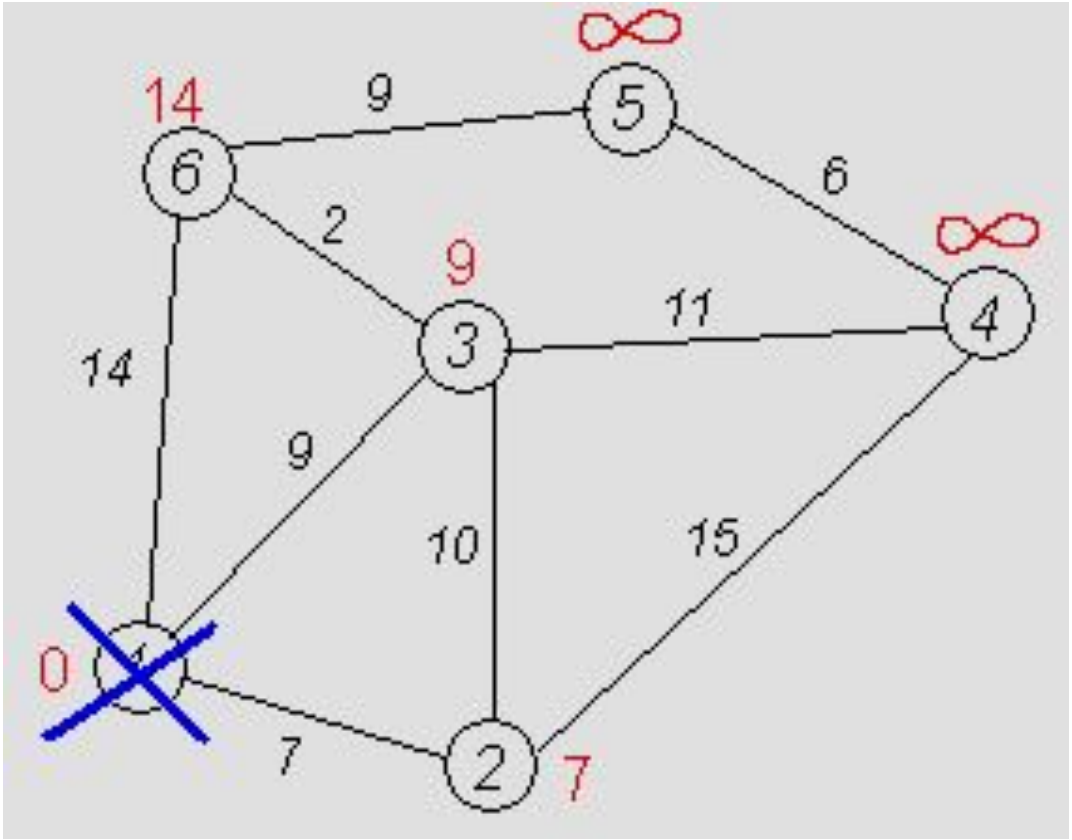




Аналогичную операцию проделываем с двумя другими соседями 1-й вершины — 3-й и 6-й.

Алгоритмы поиска путей в графе

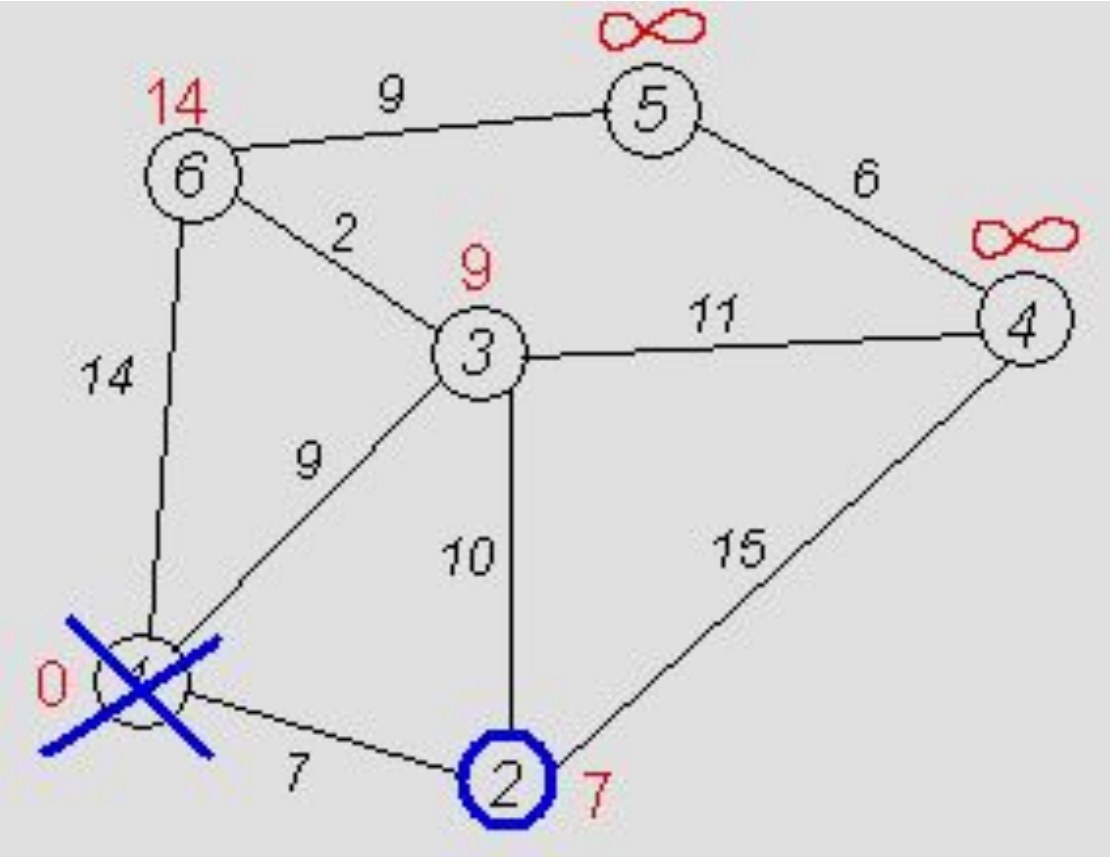
Алгоритм Дейкстры (пример)



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

Алгоритмы поиска путей в графе

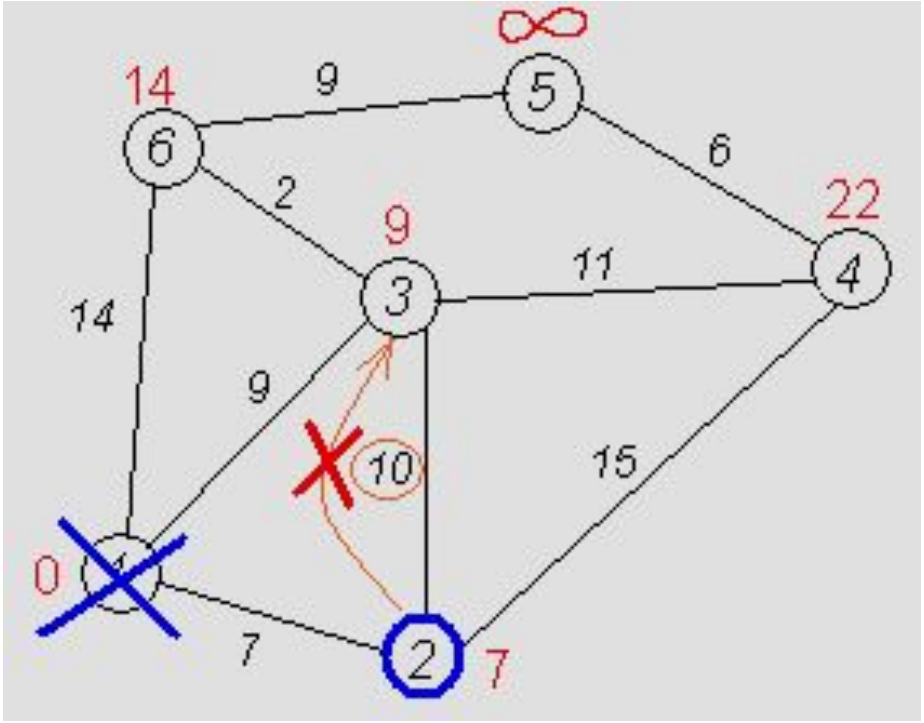
Алгоритм Дейкстры (пример)



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

Алгоритмы поиска путей в графе

Алгоритм Дейкстры (пример)



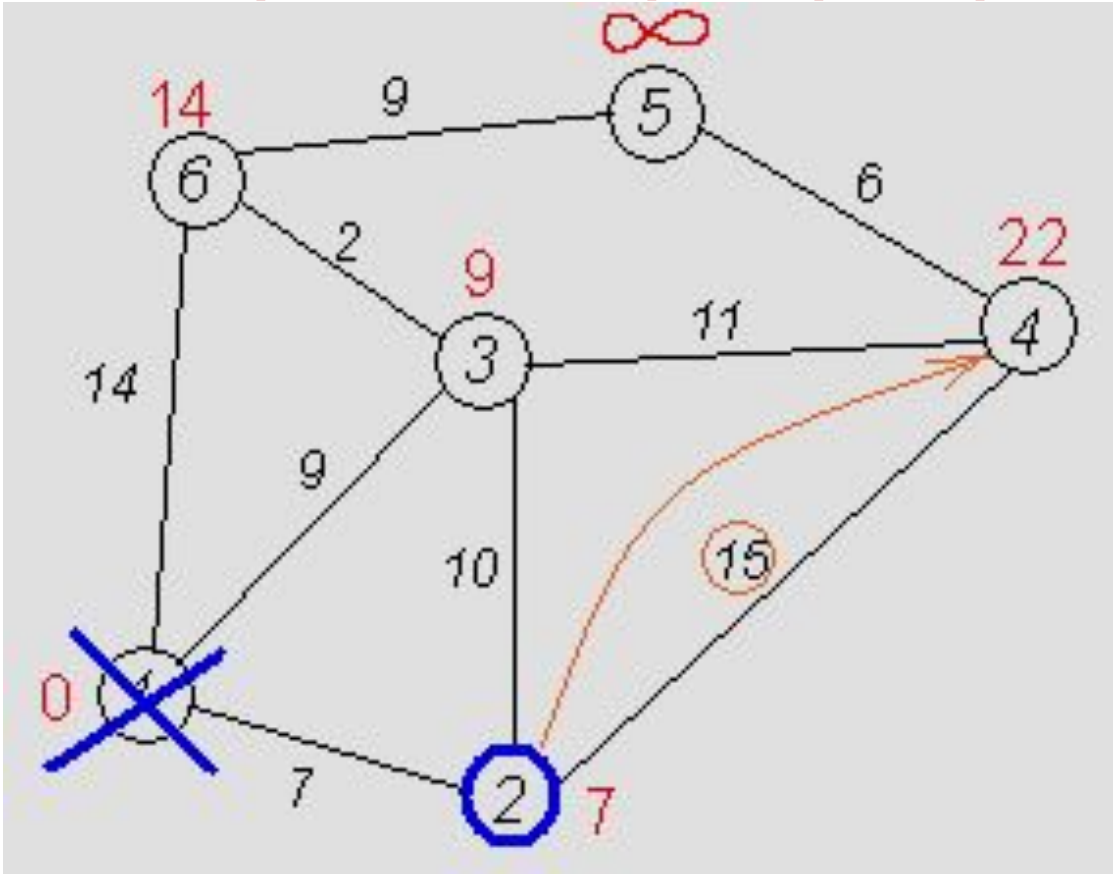
Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю. Соседями вершины 2 являются 1, 3, 4.

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. Если идти в неё через 2, то длина такого пути будет = 7 + 10 = 17. Но текущая метка третьей

Алгоритмы поиска путей в графе

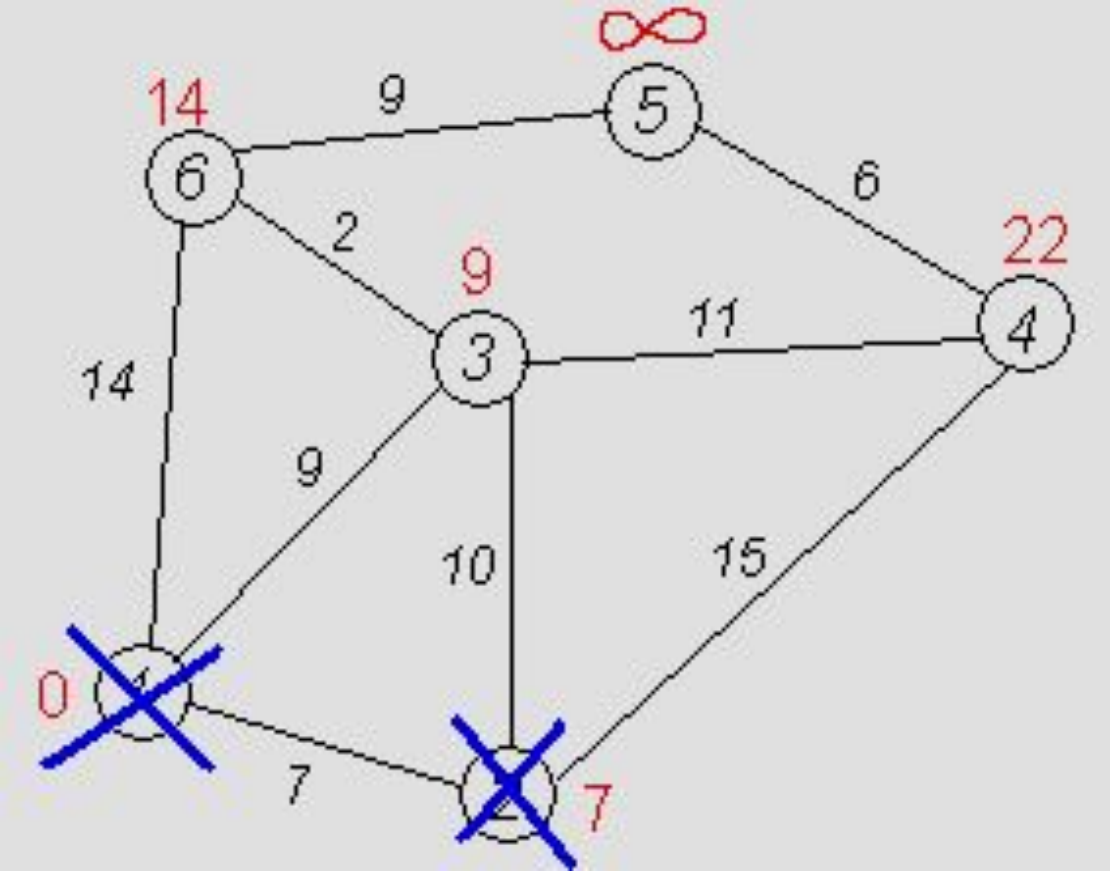
Алгоритм Дейкстры (пример)



Ещё один сосед вершины 2 — вершина 4. Если идти в неё через 2-ю, то длина такого пути будет = кратчайшее расстояние до 2 + расстояние между вершинами 2 и 4 = 7 + 15 = 22. Поскольку 22< , устанавливаем метку вершины 4 равной 22.

Алгоритмы поиска путей в графе

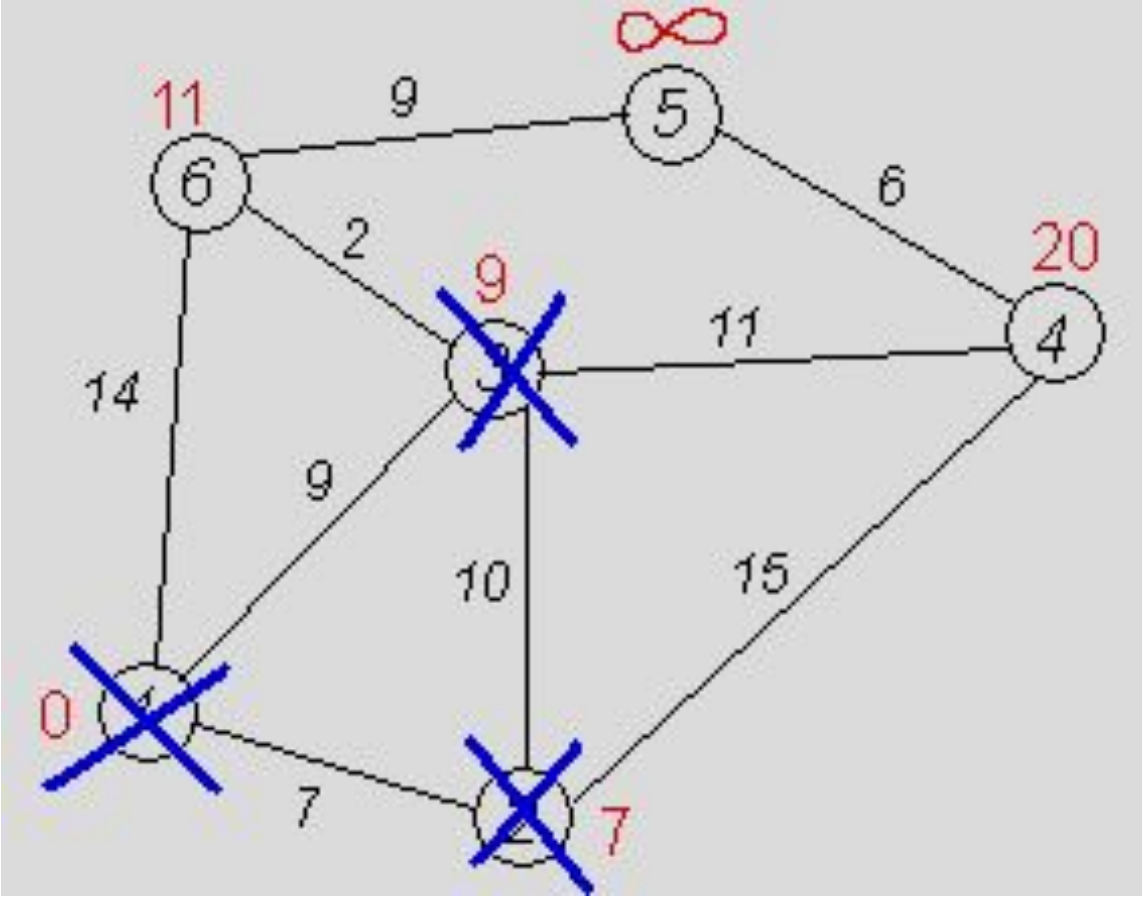
Алгоритм Дейкстры (пример)



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

Алгоритмы поиска путей в графе

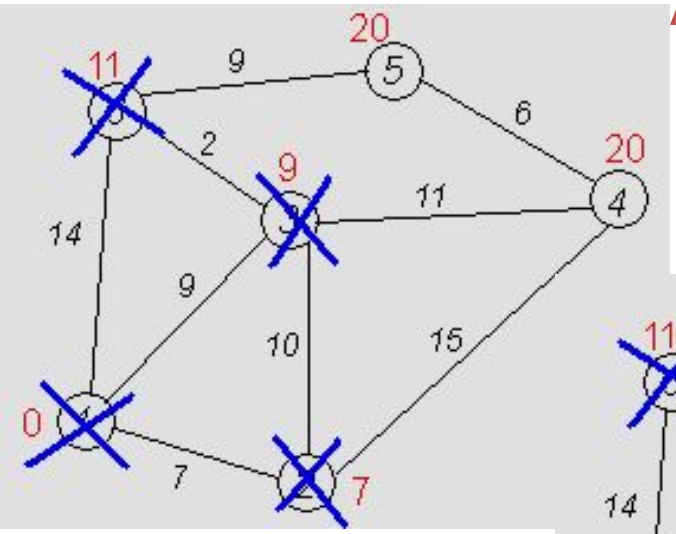
Алгоритм Дейкстры (пример)



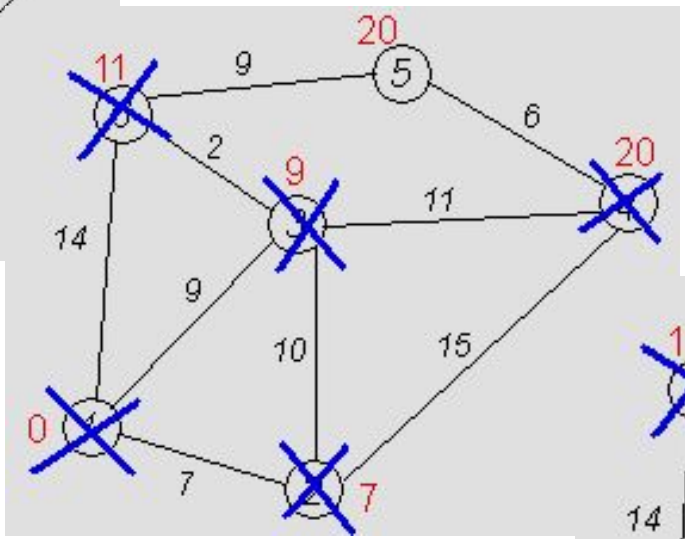
Третий шаг. Повторяем шаг алгоритма, выбрав вершину 3. После её «обработки» получим такие результаты:

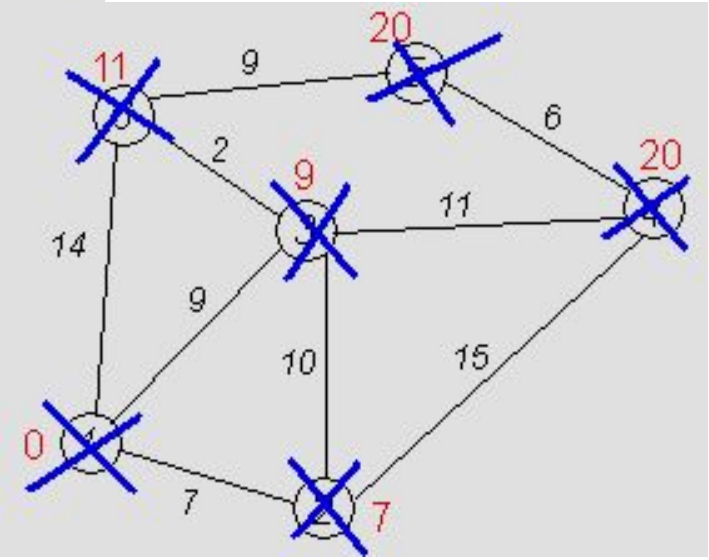
Алгоритмы поиска путей в графе

Алгоритм Дейкстры (пример)



Дальнейшие шаги. Повторяем шаг алгоритма для оставшихся вершин (Это будут по порядку 6, 4 и 5).





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 —

Алгоритмы поиска путей в графе

Путь минимальной суммарной длины во взвешенном графе с произвольными весами для всех пар вершин (алгоритм Флойда)

Функция находит пути минимального веса в графе *G=(V,E),* заданном весовой матрицей *w,* у которой элемент *wi j* равен весу ребра, соединяющего *i*-ю и *j*-ю вершины. При этом полагаем, что *W[i,i]=0* для всех *i*. Путь ищется для всех пар вершин графа. Для бесконечности используется число *GM,* его можно задавать в зависимости от конкретной задачи.

Алгоритм Флойда предполагает последовательное преобразование матрицы весов *W*. В конечном итоге получаем матрицу, элементы *d[i,j]* которой представляют из себя вес минимального пути соединяющего *i* и *j* вершины.

Алгоритмы поиска путей в графе

Нахождение *K* путей минимальной суммарной длины во взвешенном графе с неотрицательными весами (алгоритм Йена)

Алгоритм предназначен для нахождения *K* путей минимальной длины во взвешенном графе между вершинами *u1,u2*. Ищутся пути, которые не содержат петель.

Работа алгоритма начинается с нахождения кратчайшего пути, для этого будем использовать уже описанный алгоритм Дейкстры. Второй путь ищем, перебирая кратчайшие отклонения от первого, третий - кратчайшие отклонения от второго, и т.д.

Алгоритмы поиска путей в графе

Алгоритм:

1. Найти минимальный путь *P1=(v11,...,v1L[1])*. Положить *k = 2*. Включить *P1* в результирующий список.
2. Положить *MinW* равным бесконечности. Найти отклонение минимального веса от *(k–1)*-го кратчайшего пути *Pk-1* для всех *i=1,2,...,L[k-1]*, выполняя для каждого *i* шаги с 3-го по 6-й.
3. Проверить, совпадает ли подпуть, образованный первыми *i* вершинами пути *Pk-1*, с подпутем, образованным первыми *i* вершинами любого из путей *j=1,2,...,k–1*). Если да, положить *W[vk-1i,vji+1]* равным бесконечности, в противном случае ничего не изменять (чтобы в дальнейшем исключить получение в результате одних и тех же путей).4. Используя алгоритм нахождения кратчайшего пути, найти пути от *vk-1i* к *u2*, исключая из рассмотрения корни *(vk-11,...,vk-1i)* (чтобы исключить петли), для этого достаточно положить равными бесконечности элементы столбцов и строк матрицы *W*, соответствующие вершинам, входящим в корень.
5. Построить путь, объединяя корень и найденное кратчайшее ответвление; если его вес меньше *MinW*, то запомнить его.
6. Восстановить исходную матрицу весов *W* и возвратиться к шагу 3.
7. Поместить путь минимального веса (*MinW*), найденный в результате выполнения шагов с 3 по 6, в результирующий список. Если *k = K*, то алгоритм заканчивает работу, иначе увеличить *k* на единицу и вернуться к шагу 2.

Контрольные вопросы

1.Какими структурами данных можно представить в памяти древовидные структуры?

2. Перечислите основные алгоритмы поиска путей в графах?

3. Какие основные классами матриц предствляются графы в памяти компьютера?

Спасибо за внимание!