Лекция по курсу
«Алгоритмы и структуры данных» /
«Технологии и методы программирования»

Поиск кратчайших путей:
алгоритм Дейкстры, алгоритм Беллмана-Форда

Ориентированные ациклические графы:
топологическая сортировка, поиск кратчайших путей

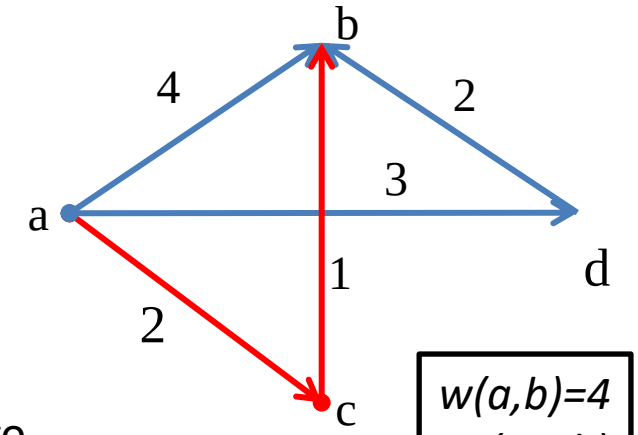
Мясников Е.В.

Поиск кратчайших путей

Пусть задан взвешенный ориентированный граф ***G=<V,E>***
с веществennознозначной весовой функцией ***w(u,v)***,
которая для любого ребра ***(u,v)*** $\in$ ***E*** возвращает вес соответствующего ребра.

Вес пути ***P***, задаваемого последовательностью вершин ***P=(v_1,v_2,…v_n)*** в таком графе, рассчитывается как сумма весов входящих в него ребер:

$$w(P) = \sum_{i=1}^{n-1} w(v_i, v_{i+1})$$



w(a,b)=4
P=(a,c,b)
w(P) = 3

Будем считать, что если некоторая вершина ***v*** недостижима из вершины ***u***, то вес соответствующего пути равен бесконечности (∞).

Кратчайшим путем из вершины ***u*** в вершину ***v*** будем называть любой путь из ***u*** в ***v***, вес которого является минимальным среди всех возможных путей из ***u*** в ***v***.

Кратчайший путь необязательно является единственным.

Поиск кратчайших путей: варианты постановки задачи

Задача нахождения кратчайшего пути может быть поставлена по-разному:

- как задача поиска кратчайшего пути из заданной исходной вершины s до всех остальных вершин;

- как задача о кратчайшем пути в заданный пункт назначения из всех остальных вершин (задача сводится к первой путем изменения направления ребер графа);

- как задача поиска кратчайшего пути между заданной парой вершин (решение первой задачи содержит и решение рассматриваемой);

- как задача поиска между всеми парами вершин.

Следует отметить, что для *невзвешенного графа поиск кратчайшего пути может выполняться с использованием алгоритма поиска в ширину*.

Поиск кратчайших путей: инициализация

Для веса текущего пути, ведущего из исходной вершины ***s*** в некоторую вершину ***u***, будем использовать обозначение d_u.
На каждом шаге описываемых далее алгоритмов величина d_u будет представлять собой ***оценку веса кратчайшего пути сверху***, сходящуюся к истинному значению по мере выполнения алгоритмов.

Для того чтобы алгоритмы поиска кратчайшего пути имели практическую ценность, они должны вычислять не просто вес такого пути, но и определять последовательность входящих в его состав вершин. Для этой цели введем для каждой вершины u предшественника $Пред_u$ – вершины из которой в кратчайшем пути осуществляется переход в вершину ***u***. В случае если предшественник вершины ***u*** не указан $Пред_u = NULL$.

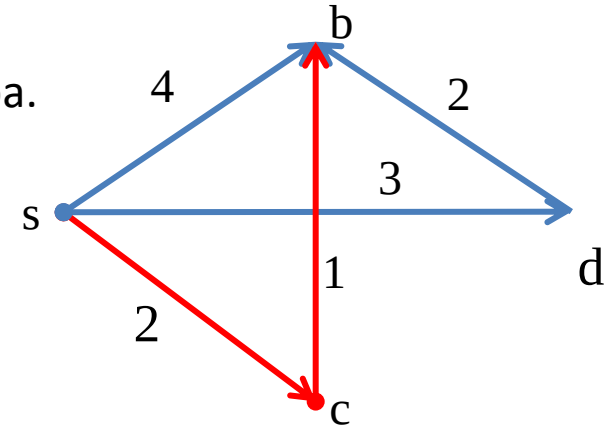
Инициализация : в текущие оценки кратчайших путей записывается бесконечность, а предшественники еще не определены. Исключение составляет лишь исходная вершина s, вес пути для которой равен нулю:

Инициализация():
    Для каждой вершины $u \in V$:
        $d_u := \infty$
        $Пред_u := NULL$
    $d_s := 0$

Ослабление (Relax) ребер

Рассматриваемые алгоритмы поиска опираются на утверждение о том, что искомые кратчайшие пути содержат другие кратчайшие пути.

Представим себе, что на некотором этапе найден некоторый путь из исходной вершины ***s*** в вершину ***v***. В основе алгоритмов поиска кратчайших путей лежат попытки улучшения текущего найденного пути путем прохода к вершине ***u*** через другие соседние с ней вершины.
Формально попытка такого улучшения реализуется через так называемую процедуру ***ослабления*** (англ., relax) ребра.



Ослабление(u,v):

Если ($d_v > d_u + w(u,v)$):

$d_v := d_u + w(u,v)$

$Пред_v := u$

Суть процедуры ослабления заключается в проверке, *возможно ли улучшить (уменьшить) оценку* ***d_v*** *веса пути к вершине* ***v*** *пройдя к вершине* ***v*** *через вершину* ***u***.

if (d_b=4) > (d_c+w(c,b)=3):
d_b=3

В последнем случае вес пути можно рассчитать как ***$d_u + w(u,v)$***.
Если этот новый путь оказывается лучше, чем найденный ранее, мы обновляем текущую оценку пути ***d_v*** и записываем вершину ***u*** как предшественника вершины ***v***, показывая тем самым, что в вершину ***v*** мы попадаем из вершины ***u***.

Алгоритм Дейкстры

```
Алгоритм Дейкстры():
    Инициализация()
    S := ∅
    Q := V
    Пока ( Q не пуста ):
            u := Q.извлечь_мин()
            S := S ∪ {u}
            Для каждой вершины v смежной с u:
                Ослабление ( u, v )
```

Поиск кратчайшего пути ведется во взвешенном ациклическом графе, веса в котором неотрицательны, т.е. ***w(u,v) >= 0***

Здесь используется список вершин ***S***, кратчайшие пути до которых уже вычислены, а также очередь вершин ***Q*** с приоритетами. Элементы в очереди ***Q*** считаются упорядоченными по возрастанию текущей оценки кратчайшего пути от исходной вершины ***s***.

Изначально множество ***S*** пусто, а в очередь ***Q*** записывается все множество вершин графа. Пока вершины в очереди ***Q*** не исчерпаны, из нее извлекается очередная вершина ***u***, оценка кратчайшего пути до которой минимальна, и заносится в список вершин ***S***. После этого выполняется ослабление всех исходящих из вершины ***u*** ребер.

Алгоритм Дейкстры

Алгоритм Дейкстры():

Инициализация()

S := ∅

Q := V

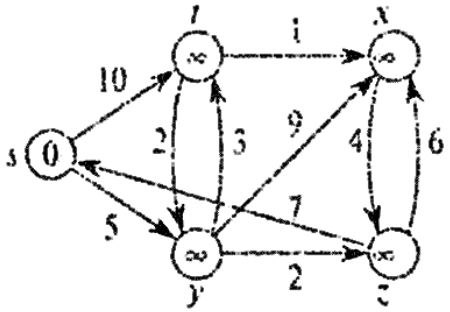
Пока (Q не пуста):

u := Q.извлечь_мин()

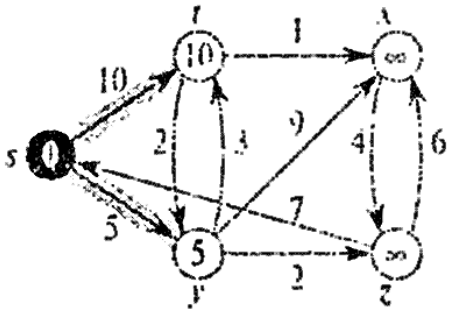
S := S ∪ {u}

Для каждой вершины v смежной с u:

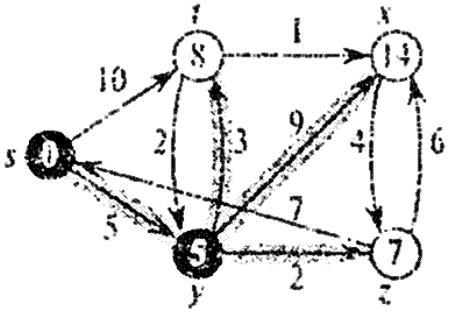
Ослабление (u, v)



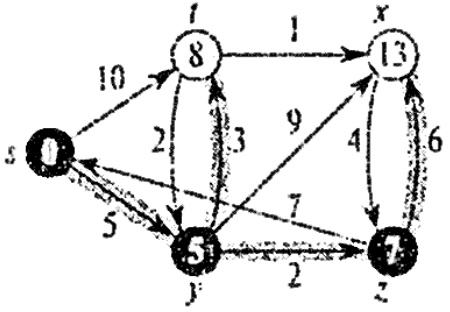
Q={s:0, t:∞, x:∞, y:∞, z :∞}



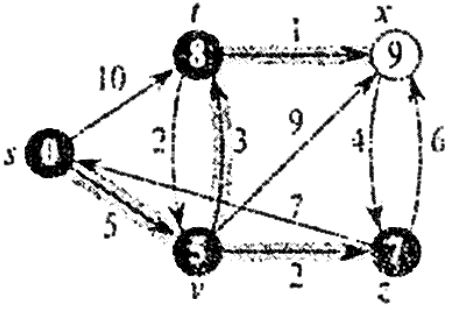
Q={y:5, t:10, x:∞, z :∞}



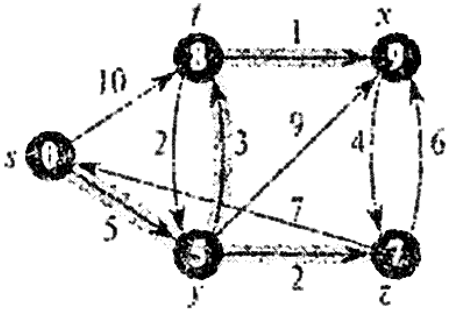
Q={z:7, t:8, x:14}



Q={t:8, x:13}



Q={x:9}



Q={}

Алгоритм Беллмана-Форда

Алгоритм пригоден и для графов с отрицательными весами, позволяет выявить, существсвует ли достижимый из исходной вершины цикл с отрицательным весом.

```
Алгоритм_Беллмана_Форда():
        Инициализация()
        Для i := 1 до (|V|-1):
                Для каждого ребра (u,v)∈E:
                        Ослабление ( u, v )
        Для каждого ребра (u,v)∈E:
                Если ( d_v > d_u + w(u,v) ):
                        Вернуть Ложь
        Вернуть Истину
```

Основная часть алгоритма, выполняемая после процедуры инициализации, представляет собой выполняемое в цикле ослабление всех ребер графа. При этом количество проходов выбирается таким образом, чтобы гарантировано получить истинные оценки кратчайших путей для всех вершин (в том случае, если это возможно). Во второй части алгоритма выносится решение о наличии в графе циклов с отрицательными весами. Такое решение принимается, если оценка пути для некоторой вершины v может быть улучшена, хотя после выполнения основной части алгоритма предполагалось, что все кратчайшие пути уже найдены. Возникновение такой ситуации говорит о наличии в графе цикла с отрицательным весом.

Алгоритм Беллмана-Форда

Алгоритм_Беллмана_Форда():

Инициализация()

Для i := 1 до (|V|-1):

Для каждого ребра (u,v)∈E:

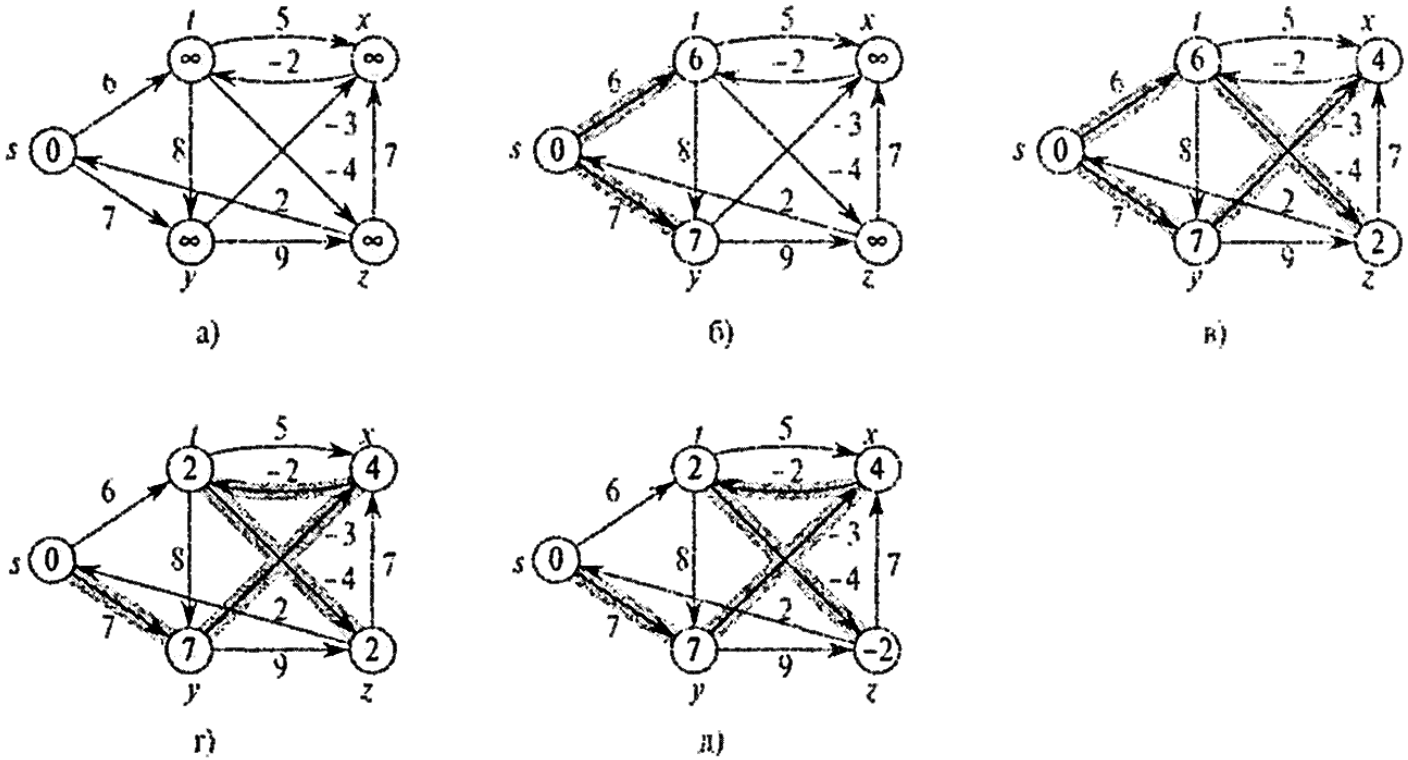
Ослабление (u, v)

Для каждого ребра (u,v)∈E:

Если ($d_v > d_u + w(u,v)$):

Вернуть Ложь

Вернуть Истину



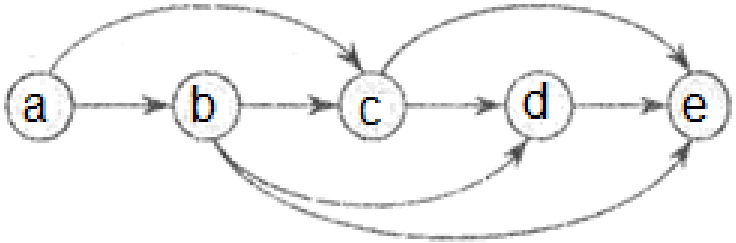
Ориентированные ациклические графы

ОРИЕНТИРОВАННЫЕ АЦИКЛИЧЕСКИЕ ГРАФЫ

Направленным циклом в ориентированном графе называют начинающуюся и заканчивающуюся в одной и той же вершине последовательность вершин, в которой для любых двух последовательных вершин существует дуга из предыдущей вершины в последующую.

Ориентированными ациклическими графами называют ориентированные графы, в которых отсутствуют направленные циклы.

На практике ориентированные ациклические графы широко используются для моделирования процессов, представляющих собой последовательности зависящих друг от друга действий:
освоение учебных дисциплин,
выполнение строительных и отделочных работ,
компиляция исходных текстов программ и т.п.



Понятно, что в таких задачах хочется иметь возможность упорядочить вершины графа таким образом, чтобы сначала шли вершины, связанные с процессами, которые должны быть выполнены раньше, а в конце – процессы, соответствующие более поздним стадиям решения задачи.

Сделать это можно с использованием т.н. топологической сортировки графа.

ТОПОЛОГИЧЕСКАЯ СОРТИРОВКА ВЕРШИН

Топологической сортировкой назовем упорядочивание вершин ориентированного ациклического графа согласно частичному порядку, задаваемому ребрами графа на множестве его вершин.

Предположим, что существует функция, возвращающая для некоторой вершины $u \in V$ начальные вершины входящих в u ребер, именуется *НачРебер(u)*.

Очевидно, что вершина, для которой множество *НачРебер(u)* пусто, может быть первой вершиной в топологической сортировке.

Представим себе, что найденную первую вершину можно вычеркнуть вместе с исходящими из нее ребрами, и искать очередную вершину для сортировки аналогичным образом, приходим к следующему алгоритму топологической сортировки.

АЛГОРИТМ ТОПОЛОГИЧЕСКОЙ СОРТИРОВКИ

Пусть *V* – неупорядоченное множество вершин графа,
V' – формируемое множество вершин в порядке топологической сортировки.
Алгоритм топологической сортировки (алгоритм Кана) :

Топологическая сортировка(V):
V' = ∅
Для всех вершин u∈V:
S_u = НачРебер(u)
Пока |V'| < |V|:
Найти вершину u∈V такую, что u∉V' и S_u=∅
Если вершина u найдена:
Добавить u в V'
Для всех v∈V:
Sv := Sv \ {u}
Иначе:
Вернуть ∅
Вернуть V'

ПОИСК КРАТЧАЙШЕГО ПУТИ: АЛГОРИТМ

Поиск кратчайших путей в ориентированном ациклическом графе может быть выполнен даже при наличии отрицательных весов, т.к. циклы в нем отсутствуют. С использованием алгоритма топологической сортировки, алгоритм поиска кратчайших путей для ориентированного ациклического графа выглядит следующим образом:

```
Поиск_кратчайших_путей():
        V' = Топологическая_сортировка(V)
        Инициализация()
        Для каждой вершины u∈V' в порядке топологической сортировки:
                Для каждой вершины v смежной с u:
                        Ослабление(u, v)
```

ПОИСК КРАТЧАЙШЕГО ПУТИ: ПРИМЕР

Поиск_кратчайших_путей():

V’ = Топологическая_сортировка(V)

Инициализация()

Для каждой вершины u∈V’ в порядке топологической сортировки:

Для каждой вершины v смежной с u:

Ослабление(u, v)