Поиск путей на графах

Граф (V, E), V – множество вершин, E – множество рёбер (дуг).
Виды задач о кратчайшем пути:

- Задача о кратчайшем пути из исходной вершины во все остальные.
- Задача о кратчайшем пути из заданной вершины v в заданную вершину u.
- Задача о кратчайшем пути между всеми парами вершин.

Популярные алгоритмы для решения задачи поиска кратчайших путей на графах:

- Алгоритм Дейкстры находит кратчайший путь из исходной вершины графа до всех остальных при использовании матрицы смежности $O(V^2)$. Алгоритм работает только для графов без рёбер отрицательного веса.
- Алгоритм Беллмана - Форда находит кратчайшие пути из исходной вершины графа до всех остальных $O(V \cdot E)$. Вес рёбер может быть отрицательным.
- Алгоритм A* находит маршрут с наименьшей стоимостью от одной вершины (начальной) к другой (целевой, конечной), используя алгоритм поиска по первому наилучшему совпадению на графе.
- Алгоритм Флойда - Уоршелла находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа.
- Алгоритм Джонсона находит кратчайшие пути между всеми парами вершин взвешенного ориентированного графа.
- Алгоритм Ли (волновой алгоритм) основан на методе поиска в ширину $O(E)$. Находит путь между вершинами s и t графа (s не совпадает с t), содержащий минимальное количество промежуточных вершин (рёбер). Основное применение - трассировки электрических соединений на кристаллах микросхем и на печатных платах. Так же используется для поиска кратчайшего расстояния на карте в стратегических играх.

Вычислительная сложность алгоритмов поиска путей зависит от способа представления графа.

Применение

- Картографические сервисы – алгоритмы нахождения кратчайшего пути на графе применяются для нахождения путей между физическими объектами на таких картографических сервисах, как карты Google или OpenStreetMap.
- Недетерминированная машина – если представить недетерминированную абстрактную машину как граф, где вершины описывают состояния, а дуги определяют возможные переходы, тогда алгоритмы поиска кратчайшего пути могут быть применены для поиска оптимальной последовательности решений. Например, если вершинами являются состояния Кубика Рубика, а дуга представляет собой одно действие над кубиком, тогда алгоритм может быть применён для поиска решения с минимальным количеством ходов.

- Сети дорог – задача поиска кратчайшего пути на графе широко используется при определении наименьшего расстояния в сети дорог. Сеть дорог можно представить в виде графа с положительными весами. Вершины являются дорожными развязками, а ребра дорогами, которые их соединяют. Веса рёбер могут соответствовать протяжённости данного участка, времени необходимому для его преодоления или стоимости путешествия по нему. Ориентированные ребра можно использовать для представления односторонних улиц. В таком графе можно ввести характеристику, которая указывает на то, что одни дороги важнее других для длительных путешествий (например, автомагистрали).
- Задача коммивояжера, цикл Гамильтона, минимальный цикл Гамильтона.
- Цикл Эйлера

Алгоритм Дейкстры

Автор – голландский учёный Эдсгер Дейкстра (1959 год). Алгоритм находит кратчайшие пути от заданной вершины графа до всех остальных и работает только с графами без рёбер отрицательного веса. Однако легко избавиться от рёбер отрицательного веса путём сложения весов всех рёбер с модулем наименьшего веса.

Суть алгоритма заключается в расстановке расстояний на вершины, начиная от исходной вершины a до всех смежных, образующих фронт волны, расходящейся от исходной вершины. На каждом шаге алгоритма фронт волны расширяется путём замены самой близкой к исходной a фронтальной вершины смежными ей вершинами, которые алгоритм ещё не посетил. Заменённая вершина заносится в список посещённых. Все посещённые вершины снабжаются расстоянием от исходной до неё. Алгоритм останавливается, когда все вершины посещены. После завершения алгоритма каждая вершина будет содержать минимальное расстояние от исходной вершины до неё и собственно путь в виде списка вершин.

Постановка задачи: Дан взвешенный ориентированный граф $G(V,E)$ без дуг отрицательного веса. Найти кратчайшие пути от некоторой вершины a графа G до всех остальных вершин этого графа.

Дано:

Пусть граф $G(V,E)$ задан списком узлов Node, хранящих имя вершины v и список смежных вершин u с весами рёбер $w_{v,u}$. Обозначим:

$w_{v,u}$ – вес ребра, соединяющего вершины v и u;

a – начальная вершина;

U – список посещённых вершин;

F – сортированный список вершин фронта;

d_u – длина кратчайшего пути из a в u;

p_u – кратчайший путь из a в u в виде списка вершин;

v, u – вершины графа;

max – максимальное число (целое беззнаковое или вещественное).

Описание алгоритма

1. Снабдим каждую вершину u меткой, содержащей расстояние d_u и путь к ней p_u. Исходная вершина имеет метку 0. Остальные вершины – метку бесконечности, которая при программировании может быть представлена максимальным в разрядной сетке числом *max*.
2. Помещаем исходную вершину в список вершин фронта *F*.
3. Вырезаем первую вершину *v* из списка *F*, помещаем её в список *U*.
4. Устанавливаем меткам вершин $u \notin U$, смежных *v*, минимальные значения расстояний d_u и путь p_u:

$d_u > d_v + w_{v,u}$ $\quad d_u = d_v + w_{v,u};$

$p_u = (p_v, u);$

}

5. Помещаем в список *F* вершины *u*, таким образом, чтобы список *F* сохранил свойство упорядоченности по увеличению значений меток расстояния.
6. Если список *F* пуст, то останавливаем алгоритм. Иначе – переходим к п.3.

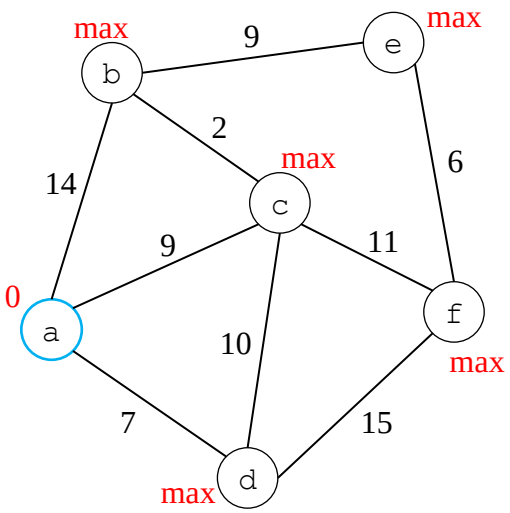

Рис.1. Исходный размеченный граф

Пример.

Шаг 1. Выполняем п.1 и 2 алгоритма: расставим метки расстояний на исходном графе (рис.1). Они красного цвета. Помещаем исходную вершину в список вершин фронта *F*. Фронт волны – синий.

$U = ();$

$F = (a);$

$d_a = 0;\ \ p_a = (a);$

$d_b = max;\ \ p_b = ();$

$d_c = max;\ \ p_c = ();$

$d_d = max;\ \ p_d = ();$

$d_e = max;\ \ p_e = ();$

$d_f = max;\ \ p_f = ();$

Шаг 2. Выполняем п.3 алгоритма. Цвет посещённых вершин – оранжевый.

$U = (a);$

$F = ();$

Выполняем п.4 алгоритма:

$d_b = 14;\ \ p_b = (a, b);$

$d_c = 9;\ \ p_c = (a, c);$

$d_d = 7;\ \ p_d = (a, d);$

Выполняем п.5 алгоритма (рис.2):

$F = (d, c, b);$

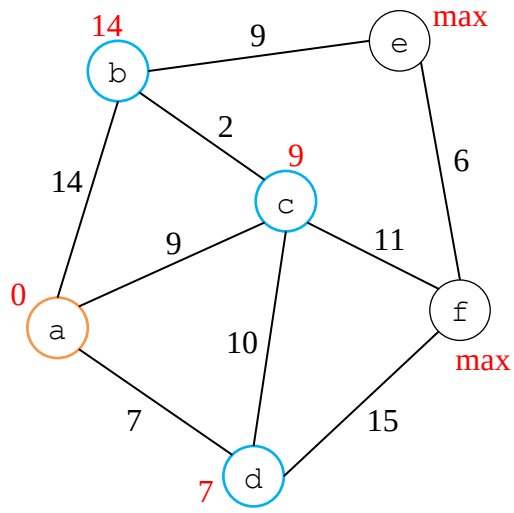

Рис.2.

Так как список F не пуст, переходим к п.3 алгоритма.

Шаг 3. Выполняем п.3 алгоритма для вершины d:

$F = (c, b);$

$U = (a, d);$

Выполняем п.4 алгоритма. Расширяем фронт волны из вершины d. Метка вершины c не изменяется, так как 9<17. Метка вершины f изменяется:

$d_f = 7 + 15 = 22;\ \ p_f = (a, d, f);$

Выполняем п.5 алгоритма (рис.3):

$F = (c, b, f);$

Так как список F не пуст, опять переходим к п.3 алгоритма.

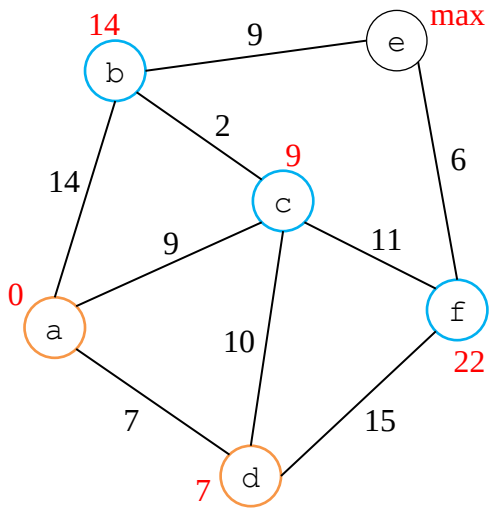

Рис.3.

Шаг 4. Выполняем п.3 алгоритма для вершины c.

$F = (b, f);$

$U = (a, d, c);$

Выполняем п.4 алгоритма (рис.4). Фронт волны из вершины c мы расширить не можем, так как все смежные вершины уже есть во фронте. Метки вершин b и f изменятся, так как путь к ним через вершину c короче:

$d_b = 9 + 2 = 11;\ \ p_b = (a, c, b);$

$d_f = 9 + 11 = 20;\ \ p_f = (a, c, f);$

п.5 алгоритма выполнять не надо, так нет новых вершин, смежных к c, которые бы расширили фронт волны.

Так как список F не пуст, опять переходим к п.3 алгоритма.

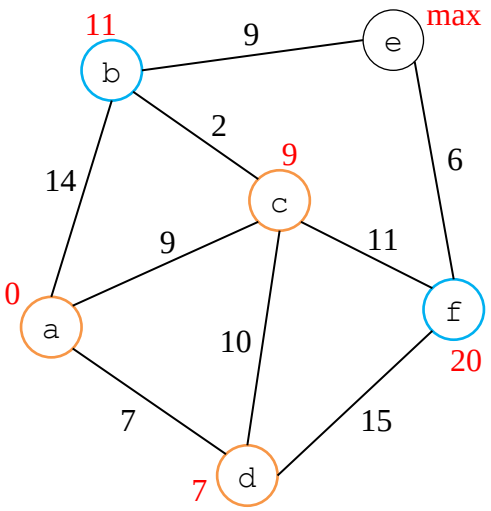

Рис.4.

Шаг 5. Выполняем п.3 алгоритма для вершины b.

$F = (f);$

$U = (a, d, c, b);$

Выполняем п.4 алгоритма (рис.5). Расширяем фронт волны из вершины b. Метка вершины e изменяется:

$d_e = 11 + 9 = 20;\ \ p_e = (a, c, b, e);$

Выполняем п.5 алгоритма:

$F = (e, f);$

Так как список F не пуст, опять переходим к п.3 алгоритма.

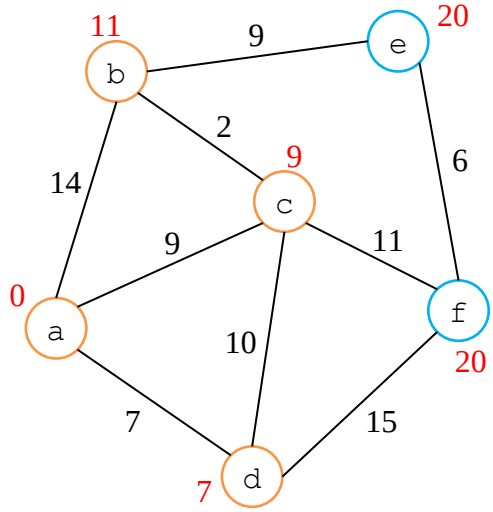

Рис.5.

Шаг 6. Выполняем п.3 алгоритма для

вершины e.

$F = (f);$

$U = (a, d, c, b, e);$

Выполняем п.4 алгоритма (рис.6).

Фронт волны из вершины e мы расширить не можем – нет новых вершин. Метка вершины f не изменяется, так как путь в f из вершины с короче, чем из вершины e.

Выполняем п.5 алгоритма:

$F = (f);$

Так как список F не пуст, опять переходим к п.3 алгоритма.

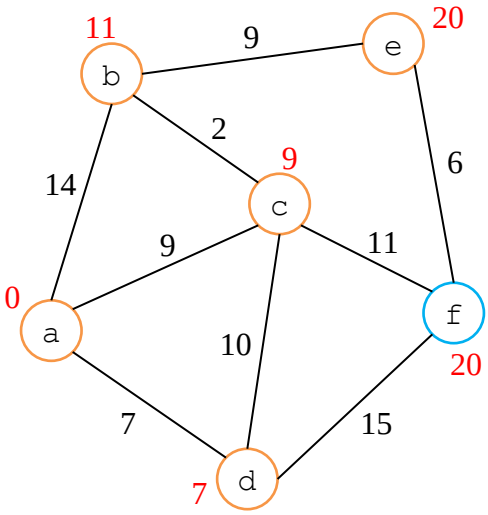

Рис.6.

Шаг 7. Выполняем п.3 алгоритма для вершины f.

$F = ();$

$U = (a, d, c, b, e, f);$

Выполняем п.4 алгоритма (рис.7).

Фронт волны из вершины f мы расширить не можем – нет новых вершин.

п.5 алгоритма выполнять не надо, так нет новых вершин, смежных к f, которые бы расширили фронт волны.

Так как список F пуст, завершаем алгоритм.

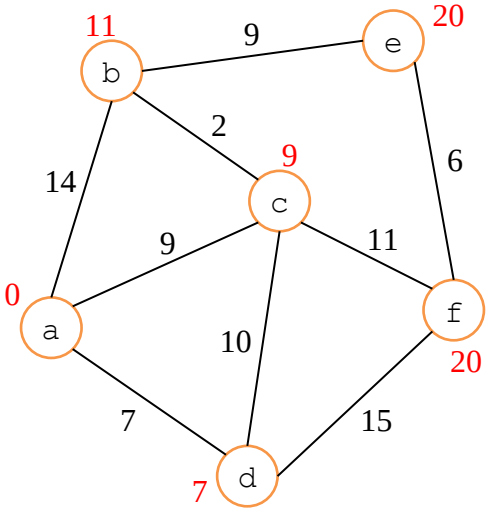

Рис.7.

В результате имеем минимальные расстояния из вершины a до всех остальных вершин графа, а также минимальные пути:

$d_a = 0;\ \ p_a = (a);$

$d_b = 11;\ \ p_b = (a, c, b);$

$d_c = 9;\ \ p_c = (a, c);$

$d_d = 7;\ \ p_d = (a, d);$

$d_e = 20;\ \ p_e = (a, c, b, e);$

$d_f = 20;\ \ p_f = (a, c, f);$

Алгоритм Беллмана-Форда

Алгоритм решает ту же задачу, что и алгоритм Дейкстры, но работает на графах с отрицательными весами и позволяет обнаруживать отрицательные циклы. Отрицательный цикл – это цикл с суммарным отрицательным весом. Алгоритм проще в реализации по сравнению с алгоритмом Дейкстры $O(V^2)$, но имеет большую вычислительную сложность $O(V \cdot E)$, так как количество дуг в графе, как правило, больше чем вершин: $E > V$.

Постановка задачи: Дан взвешенный ориентированный граф $G(V,E)$, возможно с дугами отрицательного веса. Найти кратчайшие пути от некоторой вершины a графа G до всех остальных вершин этого графа.

Дано:

Пусть граф $G(V,E)$ задан матрицей смежности M, хранящей веса дуг $w_{v,u}$.

Обозначим:

$w_{v,u}$ – вес ребра, соединяющего вершины v и u;

a – начальная вершина;

d – массив расстояний от вершины a до любой другой вершины;

v, u – вершины графа;

max – максимальное число (целое беззнаковое или вещественное).

Описание алгоритма

1. Создадим одномерный массив d, присвоим всем элементам максимальное число max, а элементу a присвоим ноль, так как путь из вершины a до неё же самой равен нулю:

```
for each v ∈ V
    do d[v] = max
d[a] = 0
```

2. $|V|-1$ раз выполнить действия: для каждой дуги (u,v), имеющей вычисленное расстояние до исходящей вершины $d[u] < max$, обновить расстояние $d[v]$ до вершины v в случае, если d[v]>d[u]+w(u,v):

```
for i=1 to |V|-1
    do for each (u,v)∈E и d[u]<max
        do if d[v] > d[u] + w(u, v)
           then d[v] = d[u] + w(u, v)     // релаксация дуги
```

3. Проверим наличие отрицательных циклов. Для этого попытаемся провести релаксацию хотя бы одной дуги. Если релаксация получится, то граф содержит отрицательный цикл:

```
for each (u,v)∈E
    do if d[v] > d[u] + w(u, v)     // релаксация возможна?
        then return false    // есть отрицательный цикл
```

4. Если отрицательный цикл не найден, то возвращаем массив d, в котором находятся длины кратчайших путей от вершины a до остальных вершин:

```
return d
```

Алгоритм выполняет $|V|-1$ шагов. Этого достаточно, так как самый длинный путь содержит не более чем $|V|-1$ дуг.

Пример поиска длин кратчайших путей (рис.8).

1. Исходный граф на рис 8,a. Начальная вершина s. Она имеет расстояние 0. До остальных вершин расстояние max, которое помечено знаком бесконечности. Алгоритм сделает 4 шага, так как граф содержит 5 вершин.

2. На первом шаге (рис 8,b) найдено две дуги (s,t) и (s,y), у которых исходящая вершина имеет расстояние, меньшее max. Поэтому вершины t и y помечаются новыми расстояниями 6 и 7 соответственно.

3. На втором шаге (рис 8,c), кроме дуг (*s*,*t*) и (*s*,*y*), найдено ещё четыре дуги, исходящие из вершин t и y , расстояние до которых нам уже известно. Поэтому вершины x и z помечаются новыми расстояниями 4 и 2 соответственно.
4. На третьем шаге (рис 8,d) обновлено расстояние до вершины *t*.
5. На четвёртом шаге (рис 8,e) обновлено расстояние до вершины *z*.

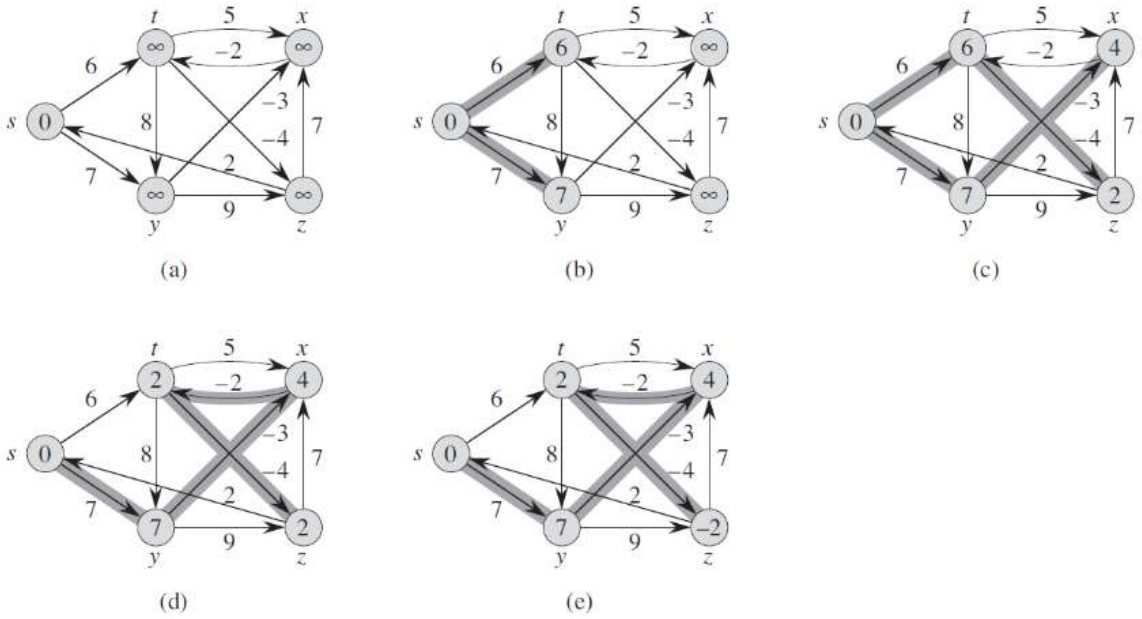

Рис.8.

Если алгоритм Беллмана-Форда не сошелся после $|V|-1$ итераций (т.е. дальнейшие релаксации возможны), то в графе есть циклы с отрицательным весом.
У алгоритма Беллмана-Форда есть интересная особенность: его можно выполнять параллельно: релаксацию каждой дуги можно выполнять независимо на каждой из вершин.
Описанный алгоритм вычисляет только длины путей, но не сами пути. Для сохранения минимальных путей предлагается в дополнительный одномерный массив записывать вершину-предшественник той вершины, расстояние до которой было сохранено. На рис.8 серой тенью выделены те дуги, по которым для некоторой вершины можно определить предшествующую ей вершину. Обратите внимание как при переходе рис.8,c→рис.8,d предшественник вершины t был заменён новым предшественником – вершиной x. До этой замены предшественником была вершина s.

ЛР:
1. Разработать программу поиска всех путей из заданной вершины на основе алгоритма Дейкстры. Представление графа выбрать самостоятельно.
2. Разработать программу поиска всех путей из заданной вершины на основе алгоритма Беллмана-Форда. Представление графа выбрать самостоятельно.