# Кратчайшие пути

## Постановка задачи

**Задача о кратчайшем пути** - задача поиска кратчайшего пути от заданной вершины до заданной или до всех остальных.

Существует множество алгоритмов поиска кратчайшего пути:

- **Алгоритм Дейкстры** находит кратчайший путь от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.
- **Алгоритм Форда-Беллмана** - в отличие от алгоритма Дейкстры, способен корректно обработать отрицательный вес.
- **Алгоритм A*** - эвристический алгоритм поиска.

## Алгоритм Дейкстры (случай неотрицательных весов)

### Идея алгоритма

1. Каждый раз, когда мы хотим посетить новый узел, мы выберем узел с наименьшим известным расстоянием.
2. Как только мы переместились в узел, мы проверяем каждый из соседних узлов. Мы вычисляем расстояние от соседних узлов до корневых узлов, суммируя стоимость ребер, которые ведут к этому новому узлу.
3. Если расстояние до узла меньше известного расстояния, мы обновим самое короткое расстояние.

На каждом шаге существует множество уже обработанных вершин и еще не обработанных.

### Псевдокод

Инициализация:

```
for (v ∈ V)
    DIST[v] = ∞
    PREV[v] = ∅
DIST[v' ∈  V] = 0  // стартовая вершина
// Первый шаг:
H ← MakeQueue() // формирование очереди с приоритетами для вершины v'. Все инцидентные вершины попадают сюда
while (H ≠ ∅)
    v ← min(H) // из очереди с приоритетами выбирается минимальный DIST[v]
    for (vu ∈  E) //для каждого инцидентного ребра
        if (DIST[u] > DIST[v] + w(v,u) // если расстояние до вершины больше, чем то, по которому мы проходим - условие релаксации
```

```
            DIST[u] ← DIST[V] + w(v,u)
            PREV[u] ←
UpdatePriorities(H)
```

### Сложность алгоритма

Худший случай - каждый путь содержит в себе все остальные ($v$ вершин). Если каждый такой путь будет хранится к каждой вершине, память будет $v^2$. Для оптимизации в каждой вершине хранится не весь путь, а только предыдущую вершину, из которой можно попасть в текущую.

Общая сложность алгоритма:

- $O(v^2)$ при работе на массиве;
- $O( ln(v) )$ при работе на куче.

Если количество ребер небольшое, выгоднее использовать реализацию на куче, если же ребер намного больше, чем вершин, лучше использовать работу на массиве.

### Применение алгоритма

Одно из применений алгоритма - маршрутизация. Например, алгоритм OSPF (Open Shortest Pass First). Каждый маршрутизатор строит некоторый граф и использует алгоритм Дейкстры в чистом виде.

1. Очередь приоритетов пуста. Работа алгоритма закончена.

## Алгоритм Флойда-Уоршелла вычисления расстояний между всеми парами вершин

Алгоритм Флойда — Уоршелла — алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного графа без циклов с отрицательными весами с использованием метода динамического программирования.

### Идея алгоритма

Будем считать, что в графе $n$ вершин, пронумерованных числами от $0$ до $n - 1$. Граф задан матрицей смежности, вес ребра $i - j$ хранится в $w_{ij}$. При отсутствии ребра $i - j$ значение $w_{ij} = + \infty$, также будем считать, что $w%%_{%%ii} = 0$.

Пусть значение $a_{ij}^{k}$ равно длине кратчайшего пути из вершины $i$ в вершину $j$, при этом путь может заходить в промежуточные вершины только с номерами меньшими $k$ (не считая начала и конца пути). То есть $a_{ij}^{0}$ - это длина кратчайшего пути из $i$ в $j$, который вообще не содержит промежуточных вершин, то есть состоит только из одного ребра $i - j$, поэтому $a_{ij}^{0} = w{ij}$. Значение $a_{ij}^{1} = w{ij}$ равно длине кратчайшего

пути, который может проходить через промежуточную вершину с номером 0, путь с весом $a_{ij}^{2}$ может проходить через промежуточные вершины с номерами 0 и 1 и т. д. Путь с весом $a{ij}^{n}$ может проходить через любые промежуточные вершины, поэтому значение $a_{ij}^{n}$ равно длине кратчайшего пути из $i$ в $j$.

Алгоритм Флойда последовательно вычисляет $a_{ij}^{0}, a_{ij}^{1}, a_{ij}^{2}, \dots, a_{ij}^{n}$, увеличивая значение параметра $k$. Начальное значение - $a_{ij}^{0} = w{ij}$.

Теперь предполагая, что известны значения $a_{ij}^{k - 1}$ вычислим $a_{ij}^{k}$. Кратчайший путь из вершины $i$ в вершину $j$, проходящий через вершины с номерами, меньшими, чем $k$ может либо содержать, либо не содержать вершину с номером $k - 1$. Если он не содержит вершину с номером $k - 1$, то вес этого пути совпадает с $a_{ij}^{k - 1}$. Если же он содержит вершину $k - 1$, то этот путь разбивается на две части: $i - (k - 1)$ и $(k - 1) - j$. Каждая из этих частей содержит промежуточные вершины только с номерами, меньшими $k - 1$, поэтому вес такого пути равен $a_{i, k - 1}^{k - 1} + a_{k - 1, j}^{k - 1}$. Из двух рассматриваемых вариантов необходимо выбрать вариант наименьшей стоимости, поэтому:

$$ a_{ij}^{k} = \min ( a_{ij}^{k - 1}, a_{i, k - 1}^{k - 1} + a_{k - 1, j}^{k - 1} ) $$

```
int A[n + 1][n][n];
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
        A[0][i][j] = W[i][j];
}
for (int k = 1; k <= n; ++k) {
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            A[k][i][j] = min(A[k-1][i][j], A[k-1][i][k-1] + A[k-1][k-1][j]);
}
```

Внешний цикл в этом алгоритме последовательно перебирает все вершины, затем пытается улучшить пути из $i$ в $j$, разрешив им проходить через выбранную вершину. Упростим этот алгоритм, избавившись от «трехмерности» массива A: будем только хранить значение кратчайшего пути из $i$ в $j$ в $A[i][j]$, а при улучшении пути будем записать новую длину пути также в $A[i][j]$. Также изменим определение цикла по переменной $k$, заменив значение $k - 1$ на $k$.

```
int A[n][n];
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
        A[i][j] = W[i][j];
}
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            A[i][j] = min%%([%%i][j], A[i][k] + A[k][j]);
}
```

Очевидно, что **сложность** такого алгоритма $O(n^3)$.

Обратите внимание, что при наличии ребер отрицательного веса значения $A[i][j]$ могут уменьшатся. Поэтому может оказаться, что значение $A[i][j]$ было равно $INF$, а затем оно уменьшилось благодаря наличию ребер отрицательного веса. В результате значение $A[i][j]$ оказалось меньше $INF$ (например, за счет объединения пути длиной $INF$ и пути отрицательного веса), но при этом все равно пути между вершинами $i$ и $j$ нет. Поэтому нужно либо ставить дополнительные проверки на то, что $A[i][k]$ и $A[k][j]$ не равны $INF$, либо значения, которые незначительно меньше $INF$, также считать отсутствием пути.

Алгоритм Флойда некорректно работает при наличии цикла отрицательного веса, но при этом если путь от $i$ до $j$ не содержит цикла отрицательного веса, то вес этого пути будет найден алгоритмом правильно. Также при помощи данного алгоритма можно определить наличие цикла отрицательного веса: если вершина $i$ лежит на цикле отрицательного веса, то значение $A[i][i]$ будет отрицательным после окончания алгоритма.

Для восстановления ответа необходим двумерный массив предшественников. Будем считать, что в $Prev[i][j]$ хранится номер вершины, являющейся предшественником вершины $j$ на кратчайшем пути из вершины $i$. Тогда при обновлении значения $A[i][j]$ нужно также обновить предшественника. А именно, если путь $i - j$ был обновлен на путь, проходящий через вершину $k$, то теперь предшественником вершины $j$ на пути из $i$ становится вершина, которая была ее предшественником на пути из $k$, то есть необходимо присвоить $Prev[i][j]=Prev[k][j]$.

Запишем алгоритм, который сохраняет предшественников, а также добавим проверки на существование пути:

```
vector < vector > A = W;
vector < vector > Prev(n, vector(n, -1));
for (int k = 0; k < n; ++k)
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (A[i][k] < INF && A[k][j] < INF && A[i][k] + A[k][j] <
A[i][j]) {
                A[i][j] = A[i][k] + A[k][j]; Prev[i][j] = Prev[k][j];
            }
```

Восстановление пути из $i$ в $j$ аналогично ранее рассмотренным алгоритмам, только необходимо учесть двумерность массива Path:

```
vector Path;
while (j != -1) {
    Path.push_back(j); j = Prev[i][j];
}
reverse(Path.begin(), Path.end());
```

From:
https://se.moevm.pro/ - **МОЭВМ Вики [se.moevm.pro]**

Permanent link:
**https://se.moevm.pro/doku.php/courses:algorithms_building_and_analysis:materials:shortest_ways**

Last update: