**Алгоритм Флойда**

**Алгоритм Флойда (или Флойда-Уоршелла, Floyd–Warshall)** позволяет найти кратчайшее расстояние между любыми двумя вершинами в графе, при этом веса ребер могут быть как положительными, так и отрицательными. Данный алгоритм также использует идею динамического программирования.

Будем считать, что в графе n вершин, пронумерованных числами от 0 до n−1. Граф задан матрицей смежности, вес ребра i−j хранится в wij. При отсутствии ребра i−j значение wij=+∞, также будем считать, что wii=0.

Пусть значение aijk 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 aij0 - это длина кратчайшего пути из i в j, который вообще не содержит промежуточных вершин, то есть состоит только из одного ребра i−j, поэтому aij0=wij. Значение aij1=wij равно длине кратчайшего пути, который может проходить через промежуточную вершину с номером 0, путь с весом aij2 может проходить через промежуточные вершины с номерами 0 и 1 и т. д. Путь с весом aijn может проходить через любые промежуточные вершины, поэтому значение aijn равно длине кратчайшего пути из i в j.

Алгоритм Флойда последовательно вычисляет aij0, aij1, aij2, …, aijn, увеличивая значение параметра k. Начальное значение - aij0=wij.

Теперь предполагая, что известны значения aijk−1 вычислим aijk. 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 aijk−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 ai,k−1k−1+ak−1,jk−1. Из двух рассматриваемых вариантов необходимо выбрать вариант наименьшей стоимости, поэтому:

aijk=min(aijk−1,ai,k−1k−1+ak−1,jk−1)

**Пример на языКЕ C++**

**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]);**

**Пример на языке Python**

**A = [[[INF for j in range(n)] for i in range(n)] for k in range(n + 1)]**

**for i in range(n):**

**for j in range(n):**

**A[0][i][j] = W[i][j]**

**for k in range(1, n + 1):**

**for i in range(n):**

**for j in range(n):**

**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.

**Пример на языКЕ C++**

**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]);**

**Пример на языке Python**

**A = [[W[i][j] for j in range(n)] for i in range(n)]**

**for k in range(n):**

**for i in range(n):**

**for j in range(n):**

**A[i][j] = min(A[i][j], A[i][k] + A[k][j])**

Очевидно, что сложность такого алгоритма O(n3).

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:

**Пример на языКЕ C++**

**vector < vector <int > > A = W;**

**vector < vector <int > > Prev(n, vector<int>(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];**

**}**

**Пример на языке Python**

**A = [[W[i][j] for j in range(n)] for i in range(n)]**

**Prev = [[None for j in range(n)] for i in range(n)]**

**for k in range(n):**

**for i in range(n):**

**for j in range(n):**

**if A[i][k] < INF and A[k][j] < INF and 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:

**Пример на языКЕ C++**

**vector <int> Path;**

**while (j != -1)**

**{**

**Path.push\_back(j);**

**j = Prev[i][j];**

**}**

**reverse(Path.begin(), Path.end());**

**Пример на языке Python**

**Path = []**

**while j is not None:**

**Path.append(j)**

**j = Prev[i][j]**

**Path = Path[::-1]**