**Алгоритм Форда-Беллмана**

**Алгоритм Форда-Беллмана** позволяет найти кратчайшие пути из одной вершины графа до всех остальных, даже для графов, в которых веса ребер могут быть отрицательными. Тем не менее, в графе не должно быть циклов отрицательного веса, достижимых из начальной вершины, иначе вопрос о кратчайших путях является бессмысленным. При этом алгоритм Форда-Беллмана позволяет определить наличие циклов отрицательного веса, достижимых из начальной вершины.

Алгоритм Форда-Беллмана использует динамическое программирование. Введем функцию динамического программирования:

F[k][i] — длина кратчайшего пути из начальной вершины до вершины i, содержащего не более k ребер.

Начальные значения зададим для случая k=0. В этом случае F[0][start] = 0, а для всех остальных вершин i F[0][i] = INF, то есть путь, состоящий из нуля ребер существует только от вершины start до вершины start, а до остальных вершин пути из нуля ребер не существует, что будем отмечать значением INF.

Далее будем вычислять значения функции F увеличивая число ребер в пути k, то есть вычислим кратчайшие пути, содержащие не более 1 ребра, кратчайшие пути, содержащие не более 2 ребер и т. д. Если в графе нет циклов отрицательного веса, то кратчайший путь между любыми двумя вершинами содержит не более n−1 ребра (n - число вершин в графе), поэтому нужно вычислить значения F[n-1][i], которые и будут длинами кратчайших путей от вершины start до вершины i).

Рассмотрим, как вычисляется значение F[k][i]. Пусть есть кратчайший маршрут из вершины start до вершины i, содержащий не более k ребер. Пусть последнее ребро этого маршрута есть ребро j-i. Тогда путь до вершины j содержит не более k-1 ребра и является кратчайшим путем из всех таких путей, значит, его длина равна F[k-1][j], а длина пути до вершины i равна F[k-1][j] + W[j][i], где W[j][i] есть вес ребра j-i. Дальше необходимо перебрать все вершины j, которые могут выступать в качестве предыдущих, и выбрать минимальное значение F[k-1][j] + W[j][i].

Получаем следующий алгоритм:

**Реализация на языке C++**

**const int INF = 1e9;**

**vector <vector<int> > F(N, vector<int>(n, INF));**

**F[0][start] = 0;**

**for (int k = 1; k < N; ++k)**

**for (int i = 0; i < N; ++i)**

**{**

**F[k][i] = F[k - 1][i];**

**for (int j = 0; j < N; ++j)**

**if (F[k - 1][j] + W[j][i] < F[k][i])**

**F[k][i] = F[k - 1][j] + W[j][i];**

**}**

**Реализация на языке Python**

**INF = 10 \*\* 9**

**F = [[INF] \* N for i in range(N)]**

**F[0][start] = 0**

**for k in range(1, N):**

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

**F[k][i] = F[k - 1][i]**

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

**if F[k - 1][j] + W[j][i] < F[k][i]:**

**F[k][i] = F[k - 1][j] + W[j][i]**

Очевидно, что сложность такого алгоритма O(n3).

Теперь модифицируем этот алгоритм. Прежде всего, сделаем массив F одномерным - «склеим» значения F[k][i] для разных значений k, будем хранить в массиве F[i] кратчайшее известное расстояние до вершины i, улучшая его по ходу. Получим следующий код:

**Реализация на языке C++**

**const int INF = 1e9;**

**vector <int> F(N, INF);**

**F[start] = 0;**

**for (int k = 1; k < N; ++k)**

**for (int i = 0; i < N; ++i)**

**for (int j = 0; j < N; ++j)**

**if (F[k - 1][j] + W[j][i] < F[k][i])**

**F[k][i] = F[k - 1][j] + W[j][i];**

**Реализация на языке Python**

**INF = 10 \*\* 9**

**F = [INF] \* N**

**F[start] = 0**

**for k in range(1, N):**

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

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

**if F[j] + W[j][i] < F[i]:**

**F[i] = F[j] + W[j][i]**

Последние две строчки есть ни что иное, как релаксация ребра j-i, как это делается в алгоритме Дейкстры. А два последних цикла по вершинам j и i с релаксацией ребра j-i просто являются релаксацией всех ребер в графе. Но если граф «разреженный», то его удобно хранить не в виде матрицы смежности, а в виде списков смежности, тогда перебор всех ребер в графе можно осуществить быстрее, чем перебирая все пары вершин.

**Реализация алгоритма Форда-Беллмана с использованием списка ребер**

Алгоритм Форда-Беллмана удобно реализовывать, если граф хранится "списком ребер", то есть в одном массиве хранится информация обо всех ребрах графа, так как основной цикл алгоритма Форда-Беллмана заключается в переборе всех ребер графа.

Рассмотрим реализацию алгоритма Форда-Беллмана с использованием списка ребер на языке C++.

Информация о ребрах графа будет храниться в одном векторе. Каждое ребро будет храниться в массиве из трех элементов: начало ребра, конец ребра, вес ребра. Организовать считывание графа можно следующим образом:

**int n, m;**

**cin >> n >> m;**

**vector < vector <int> > edges(m, vector<int>(3));**

**for (int i = 0; i < m; ++i)**

**cin >> edges[i][0] >> edges[i][1] >> edges[i][2];**

Тогда перебрать все ребра графа, можно организовав цикл по всем элементам вектора edges:

**const int INF = 1e9;**

**vector <int> F(N, INF);**

**F[start] = 0;**

**for (int k = 1; k < N; ++k)**

**for (auto & edge: edges)**

**{**

**int start = edge[0];**

**int finish = edge[1];**

**int weight = edge[2];**

**if (F[start] + weight < F[finish])**

**F[finish] = F[start] + weight;**

**}**

Рассмотрим реализацию алгоритма Форда-Беллмана с использованием списков смежности на языке Python.

Пусть граф задан списками смежности, а вес ребра j-i хранится в словаре W[j, i], где ключ — это кортеж из j, i, а значение — вес ребра. Такой способ хранения графа рассматривался в виде одной из возможных реализаций алгоритмы Дейкстры.

Тогда перебрать все ребра графа, можно организовав цикл по всем ключам словаря W и алгоритм Форда-Беллмана можно записать в виде:

**INF = 10 \*\* 9**

**F = [INF] \* N**

**F[start] = 0**

**for k in range(1, N):**

**for j, i in W.keys():**

**if F[j] + W[j, i] < F[i]:**

**F[i] = F[j] + W[j, i]**

То есть по сути алгоритм Форда-Беллмана можно сформулировать так:

1. Проинициализировать массив F значениями F[start] = 0, F[i] = INF для остальных i.
2. Пройтись по всем ребрам j-i графа, пытаясь срелаксировать ребро j-i.
3. Пункт 2 повторить n-1 раз.

Сложность такого алгоритма равна O(nm), где n - число вершин, m - число ребер графа. Видно, что для плотных графов (где m≈n2) сложность близка к сложности предыдущего варианта алгоритма, а вот для разреженных графов (где m≈n) такой алгоритм будет существенно быстрее.

Восстановление пути делается точно так же, как в алгоритме Дейкстры. Для этого необходимо запоминать предка для каждой вершины, обновляя его при успешной релаксации ребра:

**Реализация на языке C++**

**const int INF = 1e9;**

**vector <int> F(N, INF);**

**vector <int> prev(N, -1);**

**F[start] = 0;**

**for (int k = 1; k < N; ++k)**

**for (auto & edge: edges)**

**{**

**int start = edge[0];**

**int finish = edge[1];**

**int weight = edge[2];**

**if (F[start] + weight < F[finish])**

**{**

**F[finish] = F[start] + weight;**

**pref[finish] = start;**

**}**

**}**

**Реализация на языке Python**

**INF = 10 \*\* 9**

**F = [INF] \* N**

**Prev = [None] \* N**

**F[start] = 0**

**for k in range(1, N):**

**for j, i in W.keys():**

**if F[j] + W[j, i] < F[i]:**

**F[i] = F[j] + W[j, i]**

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

Сама процедура восстановления пути совпадает с аналогичной процедурой алгоритма Дейкстры.

Алгоритм Форда-Беллмана можно остановить, если на очередном шаге ни одно ребро не было срелаксировано:

**Реализация на языке C++**

**const int INF = 1e9;**

**vector <int> F(N, INF);**

**F[start] = 0;**

**bool stop = false;**

**for (int k = 1; k < N && !stop; ++k)**

**{**

**stop = true;**

**for (auto & edge: edges)**

**{**

**int start = edge[0];**

**int finish = edge[1];**

**int weight = edge[2];**

**if (F[start] + weight < F[finish])**

**{**

**F[finish] = F[start] + weight;**

**stop = false;**

**}**

**}**

**}**

**Реализация на языке Python**

**INF = 10 \*\* 9**

**F = [INF] \* N**

**F[start] = 0**

**Stop = False**

**k = 1**

**while k < N and not Stop:**

**k += 1**

**Stop = True**

**for j, i in W.keys():**

**if F[j] + W[j, i] < F[i]:**

**F[i] = F[j] + W[j, i]**

**Stop = False**

Также алгоритм Форда-Беллмана можно использовать для проверки того, есть ли в графе цикл отрицательного веса, достижимый из начальной вершины. Для этого нужно еще раз попробовать срелаксировать все ребра. Если хотя бы одна релаксация возможна, то граф содержит цикл отрицательного веса, достижимый из начальной вершины.

**Реализация на языке C++**

**bool cycle\_found = false;**

**for (auto & edge: edges)**

**{**

**int start = edge[0];**

**int finish = edge[1];**

**int weight = edge[2];**

**if (F[start] + weight < F[finish])**

**{**

**cycle\_found = true;**

**break;**

**}**

**}**

**Реализация на языке Python**

**CycleFound = False**

**for j, i in W.keys():**

**if F[j] + W[j, i] < F[i]:**

**CycleFound = True**

**break**