***Теория по практической работе №3***

***Алгоритм Дейкстры***

Каждой вершине приписывается вес – это вес пути от начальной вершины до данной. Также каждая вершина может быть выделена. Если вершина выделена, то путь от нее до начальной вершины кратчайший, если нет – то временный. Обходя граф, алгоритм считает для каждой вершины маршрут, и, если он оказывается кратчайшим, выделяет вершину. Весом данной вершины становится вес пути. Для всех соседей данной вершины алгоритм также рассчитывает вес, при этом ни при каких условиях не выделяя их. Алгоритм заканчивает свою работу, дойдя до конечной вершины, и весом кратчайшего пути становится вес конечной вершины.

Алгоритм Дейкстры:

1. Всем вершинам, за исключением первой, присваивается вес равный бесконечности, а первой вершине 0.
2. Все вершины не выделены.
3. Первая вершина объявляется текущей.
4. Вес всех невыделенных вершин пересчитывается по формуле: вес невыделенной вершины есть минимальное число из старого веса данной вершины, суммы веса текущей вершины и веса ребра, соединяющего текущую вершину с невыделенной.
5. Среди невыделенных вершин ищется вершина с минимальным весом. Если таковая не найдена, то есть вес всех вершин равен бесконечности, то маршрут не существует. Следовательно, выход. Иначе, текущей становится найденная вершина. Она же выделяется.
6. Если текущей вершиной оказывается конечная, то путь найден, и его вес есть вес конечной вершины.
7. Переход на п. 4.

В программной реализации алгоритма Дейкстры построим множество S вершин, для которых кратчайшие пути от начальной вершины уже известны. На каждом шаге к множеству S добавляется та из оставшихся вершин, расстояние до которой от начальной вершины меньше, чем для других оставшихся вершин. При этом будем использовать массив D, в который записываются длины кратчайших путей для каждой вершины. Когда множество S будет содержать все вершины графа, тогда массив D будет содержать длины кратчайших путей от начальной вершины к каждой вершине.

Помимо указанных массивов будем использовать матрицу длин C, где элемент C[i,j] –длина ребра (i,j), если ребра нет, то ее длина полагается равной бесконечности, то есть больше любой фактической длины ребер. Фактически матрица C представляет собой матрицу смежности, в которой все нулевые элементы заменены на бесконечность.

Для определения самого кратчайшего пути введем массив P вершин, где P[v] будет содержать вершину, непосредственно предшествующую вершине v в кратчайшем пути.



***Алгоритм Беллмана-Форда***

В отличие от [алгоритма Дейкстры](https://e-maxx.ru/algo/dijkstra), этот алгоритм применим также и к графам, содержащим рёбра отрицательного веса. Если граф содержит отрицательный цикл, то, понятно, кратчайшего пути до некоторых вершин может не существовать (по причине того, что вес кратчайшего пути должен быть равен минус бесконечности); впрочем, этот алгоритм можно модифицировать, чтобы он сигнализировал о наличии цикла отрицательного веса, или даже выводил сам этот цикл.

Считаем, что граф не содержит цикла отрицательного веса. Заведём массив расстояний d[0 \ldots n-1], который после отработки алгоритма будет содержать ответ на задачу. В начале работы мы заполняем его следующим образом: d[v] = 0, а все остальные элементы d[] равны бесконечности \infty.

Сам алгоритм Форда-Беллмана представляет из себя несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию вдоль каждого ребра (a,b) стоимости c. Релаксация вдоль ребра — это попытка улучшить значение d[b] значением d[a] + c. Фактически это значит, что мы пытаемся улучшить ответ для вершины b, пользуясь ребром (a,b) и текущим ответом для вершины a.

Утверждается, что достаточно n-1 фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе (повторимся, мы считаем, что циклы отрицательного веса отсутствуют). Для недостижимых вершин расстояние d[] останется равным бесконечности \infty.

Для алгоритма Форда-Беллмана, в отличие от многих других графовых алгоритмов, более удобно представлять граф в виде одного списка всех рёбер (а не n списков рёбер — рёбер из каждой вершины). В приведённой реализации заводится структура данных \rm edge для ребра. Входными данными для алгоритма являются числа n, m, список e рёбер, и номер стартовой вершины v. Все номера вершин нумеруются с 0 по n-1.

Восстановление путей

Для этого заведём ещё один массив p[0 \ldots n-1], в котором для каждой вершины будем хранить её "предка", т.е. предпоследнюю вершину в кратчайшем пути, ведущем в неё. В самом деле, кратчайший путь до какой-то вершины a является кратчайшим путём до какой-то вершины p[a], к которому приписали в конец вершину a.

Приведём реализацию Форда-Беллмана с восстановлением пути до какой-то заданной вершины t:

vector<edge> e;

const int INF = 1000000;

vector<int> d (ORDER, INF); //init dist

vector<int> p (ORDER, -1); //init path

d[v0]=0;

for (;;) {

bool any = false;

for (int j=0; j<m; j++)

if (d[e[j].a] < INF)

if (d[e[j].b] > d[e[j].a]+e[j].cost) {

d[e[j].b] = d[e[j].a]+e[j].cost;

any = true;

p[e[j].b] =e[j].a;

}

if (!any) break;

}

for(i=0; i<n ; i++)

{

if(i==v0)

continue;

j=i;

printf("v[%d] ",i);

while(j!=v0)

{

j=p[j];

printf("%d ",j);

}

printf("\n");

} // for i