Statement of the Problem

Given a directed or undirected weighted graph with $n$vertices and $m$edges. Weights of all edges are non-negative.Contains some starting vertex s. Required to find the length of the shortest paths from vertex $s$to all other vertices, as well as provide a way to output the shortest paths themselves.

This problem is called "the problem of the shortest paths from a single source" (single-source shortest paths problem).

Algorithm

Here we describe an algorithm that offered Dutch researcher **Dijkstra** (Dijkstra) in 1959

Head of an array $d[]$in which each vertex $v$will store the current length $d[v]$of the shortest path $s$in v. Initially $d[s] = 0$, and for all other vertices, this length is equal to infinity (when implemented on a computer usually as infinity simply choose a sufficiently large number, certainly more possible path length):

$$d[v] = \infty, v \neq s$$

Furthermore, for each vertex $v$we store, it also marked or not, i.e. 's have a boolean array $u[]$. Initially, all vertices are labeled, ie

$$u[v] = \mathrm{false}$$

Dijkstra algorithm itself consists of **iterations** . On the next iteration selected vertex with the lowest value among the yet marked, ie:n $v$$d[v]$

$$d[v] = \min_{p:\ u[p]=\mathrm{false}} d[p]$$

(It is understood that on the first iteration will be selected starting vertex s.)

Selected so the top $v$notes marked. Further, in the current iteration, the vertex v produced **relaxation** : Browse all the edges (v, to)emanating from the vertex v, and for each such vertex toalgorithm tries to improve the value $d[to]$. Let the length of this edge is equal len, then a relaxation code looks like:

$$d[to] = \min(d[to], d[v] + \mathrm{len})$$

At the ends the current iteration, the algorithm proceeds to the next iteration (again selects the lowest peak value d, the relaxation produced therefrom, etc.). In the end, after $n$iterations, all vertices will be labeled, and the algorithm completes its work. It is argued that the values found $d[v]$are the desired length of the shortest paths $s$in v.

It is worth noting that if not all the vertices reachable from the vertex s, then the values $d[v]$for them will remain endless. It is clear that the last few iterations of the algorithm will just choose these peaks, but no useful work to produce these iterations will not (because an infinite distance can not prorelaksirovat other, even infinite distance too). Therefore, the algorithm can be immediately stopped as soon as the selected vertices taken top with infinite distance.

Restoration paths . Of course, you usually need to know not only the length of the shortest paths, but also to get yourself way. We show how to maintain sufficient information for subsequent recovery of the shortest path $s$to any vertex. It's enough to so-called **ancestors array** : array $p[]$, in which each vertex $v \neq s$is stored node number $p[v]$, which is the penultimate in the shortest path to the top v. Here we use the fact that if we take the shortest path to some vertex v, and then remove from the last vertex of the path, you get way, ending a vertex $p[v]$, and this will be the shortest path for the top $p[v]$. So if we have this array of ancestors, the shortest path can be restored to him, each time taking just ancestor of the current node, until we come to the starting vertex s - so we obtain the desired shortcut, but written in reverse order. Thus, the shortest path $P$to the top $v$is:

$$P = (s, \ldots, p[p[p[v]]], p[p[v]], p[v], v)$$

Necessary to understand how to build the array ancestors. However, this is very simple: at every successful relaxation, ie when the selected vertex $v$is improving distance to a vertex to, we write that ancestor vertex $to$is a vertex v:

$$p[to] = v$$

Proof

The main assertion , which is based on the correctness of Dijkstra's algorithm is as follows. It is alleged that after some vertex $v$becomes marked, the current distance to it $d[v]$is the shortest, and therefore more will not change.

Proof by induction will produce. For the first iteration of its validity is obvious - for the top $s$we have $d[s] = 0$, which is the length of the shortest path to it. Suppose now that this statement holds for all previous iterations, ie all already labeled vertices, we prove that it is not disturbed after the current iteration. Let v- vertex selected in the current iteration, ie vertex, which is going to mark the algorithm. We prove that $d[v]$indeed is the length of the shortest path to it (we denote this length through $l[v]$).

Consider the shortest path $P$to the top v. Clearly, this path can be divided into two ways: P_1consisting only of vertices labeled (at least the starting vertex $s$is in the way) and the rest of the way P_2(it may also include a marked vertex, but certainly begins with untagged). We denote $p$the first vertex of the path P_2, and through q- the last point of the path P_1.

We first prove our assertion for the top p, ie prove equality $d[p] = l[p]$. However, it is almost clear: in fact on one of the previous iterations, we chose the top $q$and perform the relaxation of it. Since (by virtue of the choice of the vertex p) to the shortest path $p$is the shortest path to $q$the edge, plus (p,q), when you run the relaxation of $q$the quantity $d[p]$actually set to the required value.

Due to the non-negativity values edges length of the shortest path $l[p]$(and she just proved equal $d[p]$) does not exceed the length of $l[v]$the shortest path to the top v. Given that $l[v] \le d[v]$(because Dijkstra's algorithm could not find a shorter route than it is at all possible), we finally obtain the relation:

$$d[p] = l[p] \le l[v] \le d[v]$$

On the other hand, since both p, and v- unmarked vertex, then, since in the current iteration been chosen vertex v, and not the top p, then we get another inequality:

$$d[p] \ge d[v]$$

From these two inequalities we conclude equality $d[p] = d[v]$, and then found out before this relationship and get:

$$d[v] = l[v]$$

QED.

Implementation

Thus, Dijkstra's algorithm is $n$iterative, each of which is selected unlabeled vertex with the lowest value $d[v]$, this vertex is labeled and then iterates through all the edges emanating from a given vertex, and along each edge is an attempt to improve the value $d[]$on the other end of the edge.

Time of the algorithm consists of:

- n Just search the top with the lowest value $d[v]$ among all unmarked vertices, ie of $O(n)$ vertices
- m times tries relaxations

At the simplest implementation of these operations will be spent, a top search $O(n)$ operations, and one relaxation - $O(1)$ operations, and the final **asymptotic behavior** of the algorithm is:

$$O(n^2 + m)$$

Implementation :

```
const int INF = 1000000000;

int main() {
     int n;
     ... чтение n ...
     vector < vector < pair<int,int> > > g (n);
     ... чтение графа ...
     int s = ...; // стартовая вершина

     vector<int> d (n, INF),  p (n);
     d[s] = 0;
     vector<char> u (n);
     for (int i=0; i<n; ++i) {
          int v = -1;
          for (int j=0; j<n; ++j)
               if (!u[j] && (v == -1 || d[j] < d[v]))
                    v = j;
          if (d[v] == INF)
               break;
          u[v] = true;

          for (size_t j=0; j<g[v].size(); ++j) {
               int to = g[v][j].first,
                    len = g[v][j].second;
               if (d[v] + len < d[to]) {
                    d[to] = d[v] + len;
                    p[to] = v;
               }
          }
     }
}
```

Here the graph $g$is stored in the form of adjacency lists: for each vertex $v$list $g[v]$ contains a list of edges emanating from that vertex, ie a list of pairs $\mathrm{pair} < \mathrm{int}, \mathrm{int} >$, where the first element of the pair - the vertex which an edge, and the second element - the weight of the edge.

After reading infest arrays distances $d[]$, marks $u[]$and ancestors $p[]$. Then executed n iterations. At each iteration, first is the summit v, which has the smallest distance $d[]$ of unmarked vertices. If the distance to the selected vertex $v$is equal to infinity, then the algorithm stops. Otherwise, the vertex is marked as tagged, and scans all edges emanating from a given vertex, and run along each edge relaxation. If relaxation is successful (ie, the distance $d[to]$changes), the distance is recalculated $d[to]$and stored ancestor $p[]$.

After all the iterations in the array $d[]$are the length of the shortest paths to all vertices, and in the array $p[]$- the ancestors of all vertices (except the homepage s).Restore the path to any vertex $t$as follows:

```
vector<int> path;
for (int v=t; v!=s; v=p[v])
     path.push_back (v);
path.push_back (s);
reverse (path.begin(), path.end());
```

Literature

- Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein. **Algorithms: Design and analysis of** [2005]
- Edsger Dijkstra. **A note on two problems in connexion with Graphs** [1959]