**Problem** (Single-Source Shortest Paths)**.** *You are given a directed or undirected weighted graph with $n$ vertices and $m$ edges. All edge weight are non-negative. You are also given a source vertex $s$. Find the lengths of the shortest paths from $s$ to all other vertices, and provide a way to obtain the paths themselves.*

Let us construct an array $d$. For every vertex $v$ we will store the current length of the shortest path from $s$ to $v$ at $d_v$. Initially $d_s = 0$ and $d_v = +\infty$ for $v \neq s$. Besides $d$, we will also store the boolean array $u$ denoting whether or not every vertex was already used by the algorithm.

The algorithm itself will consist of $n$ iterations. On every iteration we will select the yet unused vertex $v$ with the smallest value of $d_v$ via linear search in $O(n)$. We then try to improve the shortest paths via edges $(v, w)$ outgoing from $v$, that's it $d_w = \min(d_w, d_v + d(v, w))$. Once the iterations terminate, the array $d$ will contain actual distances.

The paths can be easily restored if we also store array $p$ of parents on the shortest path, that's it we set $p_w = v$ if $d_v + d(v, w) < d_w$. The path then will be $(s, \dots p_{p_{p_v}}, p_{p_v}, p_v, v)$ and can be easily constructed in the reverse order.

*Proof.* The main proposition is that once some vertex $v$ gets used the corresponding label $d_v$ contains the correct distance and won't change anymore. This fact can be easily proven by induction, and the key component of the algorithm is that we always use the vertex with the smallest current distance, which clearly cannot decrease as edge weights are non-negative. □

Note that this version of the Dijkstra algorithm runs in $O(n^2 + m)$ which is optimal for dense graphs. The graph is typically called *dense* if $m \sim n^2$.

The graph is typically called *sparse* if $m \ll n^2$. The bottleneck for sparse graphs in the above algorithm is, of course, the $O(n^2)$ term. As it comes from slow minimum selection in an array, it becomes pretty clear that a better data structure is needed.

Two possible choices are a set and a heap (a.k.a. priority queue). In practice, a heap of pairs $(d_v, v)$ outperforms the corresponding set, because of the linear underlying structure (vs AVL- or RB-tree), despite being unable to easily delete an element.

The resulting complexity becomes $O(m \log m)$ for priority queue and $O(m \log n)$ for set, which may look better, but in fact $\log m \leq 2 \log n$ as $m \leq n^2$, and the hidden constant of a set is decisively larger than that of a heap.

Dijkstra cannot be adapted to handle the case of the negative weights. There is, however, another well-known algorithm for that purpose. We are talking, of course, about the Ford—Bellman algorithm. This algorithm also deals with an array $d$ of shortest distances found so far, and also consists of iterations, namely $n - 1$ of them this time.

On every iteration we again make the same attempt to improve the distance $d_w$ to $d_v + d(v, w)$ by using an edge $(v, w)$. The only difference is that this time around we use all edges on every iteration, because the negative ones can come in handy after their original discovery.

*Proof.* The main proposition is that after $k$ iterations all paths of length at most $k$ (measured in the number of edges) were found. This fact can be easily proven by induction, and the key component of the algorithm is that we always use all the edges. The final result follows as no path can be longer that $n - 1$ edges (except for the case of a negative cycle). Finally, the negative cycle case can be easily detected if the array $d$ changes on the $n$-th iteration. □

Unfortunately, Ford—Bellman runs in $O(VE)$, which is hardly suitable for many practical problems. Fortunately, Ford—Bellman often stops changing anything long before $n - 1$ iterations. Let's head on to practical problems now.

# Problem A. Dijkstra

| | |
|---|---|
| Input filename: | `dijkstra.in` |
| Output filename: | `dijkstra.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 Mb |

You are given a directed weighted graph. Find the shortest path from one vertex to another.

## Input file format

First line of the input file contains three numbers: the number $1 \le n \le 2000$ of vertices, and the indices $1 \le s, d \le n$ of the source and destination vertices respectively.

The following $n$ lines contain the adjacency matrix of a graph, with $-1$ denoting the absence of an edge, and non-negative numbers denoting the distance between the pair of vertices. It is guaranteed that the main diagonal contains all zeros.

## Output file format

Output the requested distance, or $-1$ if there is no path from source to destination.

## Sample tests

| dijkstra.in | dijkstra.out |
|---|---|
| 3 1 2<br>0 -1 2<br>3 0 -1<br>-1 4 0 | 6 |

# Problem B. Distance Between Vertices

| | |
|---|---|
| Input filename: | `distance.in` |
| Output filename: | `distance.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 Mb |

You are given an undirected weighted graph. Find the path of the minimal weight from one vertex to another.

## Input file format

First line of the input file contains four positive integers: the numbers $1 \le n \le 70000$ of vertices and $1 \le m \le 200000$ edges, as well as the indices $1 \le s \ne d \le n$ of the source and destination vertices respectively.

The following $m$ lines contain the triples of positive integers $1 \le u_i, v_i \le n$ and $0 \le w_i \le 100000$, denoting the indices of the endpoints of an undirected edge, together with the weight of this edge.

## Output file format

First string must contain one positive integer — the requested weight. The second string must contain the path itself.

If there is no path from source to destination output a single line containing $-1$.

## Sample tests

| distance.in | distance.out |
|---|---|
| 4 4 1 3<br>1 2 1<br>3 4 5<br>3 2 2<br>4 1 4 | 3<br>1 2 3 |

# Problem C. Shortest Path

| | |
|---|---|
| Input filename: | `path.in` |
| Output filename: | `path.out` |
| Time limit: | 2 seconds |
| Memory limit: | 256 Mb |

You are given a directed weighted graph and a vertex $s$. Find the distances of the shortest paths from $s$ to all other vertices.

## Input file format

First line of the input file contains three positive integers: the numbers $1 \le n \le 2000$ of vertices and $1 \le m \le 5000$ edges, as well as the index $1 \le s \le n$ of the source vertex.

The following $m$ lines contain the triples of positive integers $1 \le b_i, e_i \le n$ and $-10^{15} \le w_i \le 10^{15}$, denoting the indices of beginning and the end vertices of an edge, together with the weight of the edge.

## Output file format

Output $n$ lines, with the distance from $s$ to the corresponding vertex. Output `*` if there is no path, and `-` if there is no shortest path.

## Sample tests

| path.in | path.out |
|---|---|
| 6 7 1<br>1 2 10<br>2 3 5<br>1 3 100<br>3 5 7<br>5 4 10<br>4 3 -18<br>6 1 -1 | 0<br>10<br>-<br>-<br>-<br>* |