Minimum Cost Trees

Input file name: `standard input`

Output file name: `standard output`

Time limit: 10 s

Memory limit: 1024 MB

In this optimization problem, you have to construct a stable information transmission channel in a communication network.

From here on we will use some notions of graph theory. First occurrences of terms will be highlighted in *italics*. If you are not familiar with some definitions, you may learn them by this link.

A communication network is a *directed graph* G; each *vertex* of that graph can send a signal to its neighbors along directed *edges* (aka *arcs*). The particular vertex s is called the information source, and several other vertices $\{t_i\}$ are called receivers. The goal is to find a set of *paths* from the source to all the terminals. These paths make up a *directed tree* $T \subseteq G$ from its *root* to its *leaves*.

Each directed edge of G has two numeric parameters:

- `delay`, the time needed for a signal to pass through this edge;
- `cost`, the cost of adding this edge to a directed tree.

The main goal of this problem is to construct a reliable data transfer scheme that would be able to deliver information to receivers even in the case of some of its parts failing. For this purpose we have to find two directed trees T_1, T_2 without common edges; if in the real system a part of one tree stops functioning properly, the other tree would be able to pass the signal.

The two trees must meet the following constraints:

1. do not have common edges;
2. be able to transfer a signal within the `delay` time amount that is not greater than a universal bound D;
3. have as least total `cost` of both trees as possible.

Formal statement

A directed graph $G = (V, E)$ is given. Each edge $e \in E$ has two corresponding values:

- d_e, the time it takes for a signal to pass through this edge;
- c_e, the cost of adding this edge to a tree.

The source vertex $s \in V$ and the set of terminals $\{t_i \in V,\ t_i \neq s\}$ are specified. We will call a *tree T directed from the source to the terminals* the minimal set of edges containing paths from s to each of t_i. For such a tree T, all t_i are reachable from the vertex s, and we get an undirected *tree* (a *connected* graph without *cycles*) if we remove orientation of all edges of T.

You are required to (in decreasing order of importance):

1. find two directed trees T_1, T_2 from s to $\{t_i\}$ which do not have common edges;
2. ensure that every path P from s to $\{t_i\}$ in both trees satisfies the constraint $\sum\limits_{e \in P} d_e \leq D$,
3. minimize $\sum\limits_{e \in T_1 \sqcup T_2} c_e$.

Graph structure

In each test case, the graph has a specific structure that guarantees the existence of two trees directed from the source to the terminals.

We describe the graph construction iteratively.

1. A *simple cycle* is a cycle in the usual sense of graph theory, containing 4 to 20 vertices.
   Example: vertices $(56, 55, 63, 64)$ form a simple cycle in the right part of the figure.
2. A *complex cycle* is a union of 3 to 10 simple cycles that are united by a common forming path L. The united simple cycles do not have common vertices. Every simple cycle has at least two vertices in common with the forming path L, which go in a row and form a simple path.
   Example: in the right part of the figure, path $L = (6, 51, 52, 53, 54, 55, 56, 57, 58, 59, 7)$ unites three cycles: $(59, 58, 65, 66)$, $(56, 55, 63, 64)$, and $(53, 52, 51, 60, 61, 62)$. These cycles have common sections with the forming path L which are $(59, 58)$, $(56, 55)$, and $(53, 52, 51)$, respectively.
3. A *core* is a *2-edge-connected* graph (that is, a graph which remains connected after any edge is removed) to which 2 to 20 complex cycles are attached. Attached complex cycles do not have common vertices. Each complex cycle has exactly two vertices in common with the core: the start and end vertices of the forming path of this complex cycle, and these two vertices are also *adjacent* to each other.
   Example: vertices $(0, 1, 2, 3, 4, 5, 6, 7)$ form a core to which three complex cycles are attached by pairs of vertices $(2, 3)$, $(4, 5)$, and $(6, 7)$.

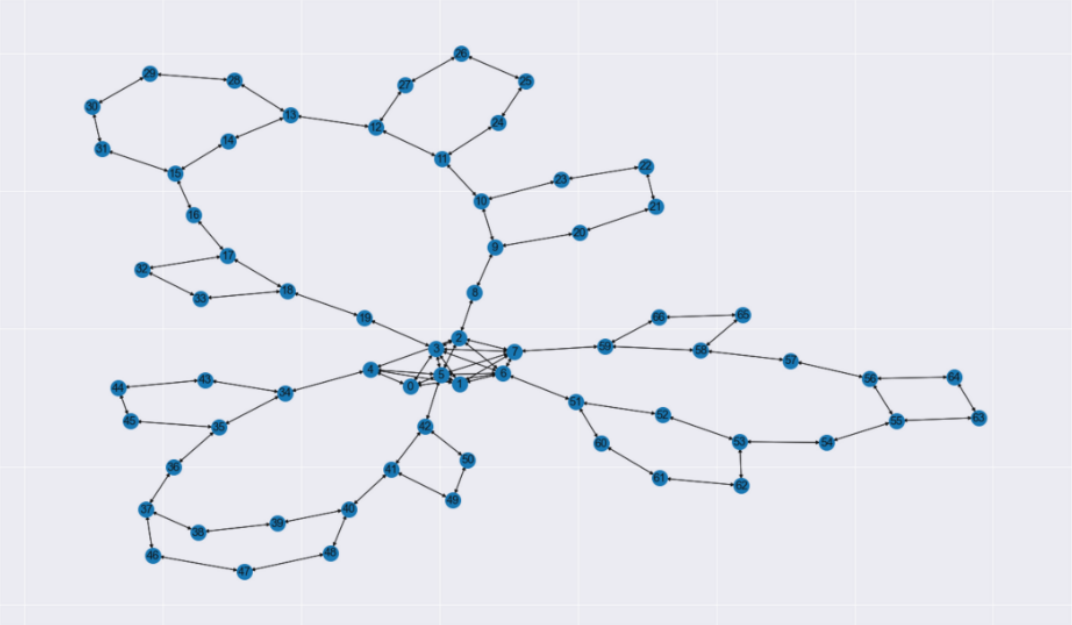
4. A *multicore* is a graph composed of one or more cores. If there are no more than 4000 vertices in the graph, then the multicore consists of a single core. If there are more than 4000 vertices, then a multicore is made up of $q \geq 2$ cores in one of three ways:
    - In each core i, two adjacent vertices $v_{i,1}$, $v_{i,2}$ are chosen, then $2q$ edges of the cycle $(v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2}, \ldots, v_{q,1}, v_{q,2})$ are added. The order of the cores is arbitrary.
    - In each core i, two (not necessarily adjacent) vertices $v_{i,1}$, $v_{i,2}$ are chosen, then q edges of two cycles $(v_{1,1}, v_{2,1}, \ldots, v_{q,1})$ and $(v_{1,2}, v_{2,2}, \ldots, v_{q,2})$ are added. The order of the cores is arbitrary, but in both cycles the cores go in the same order.
    - Combined option: in each core i, two adjacent vertices $v_{i,1}$, $v_{i,2}$ are chosen, then three cycles are added: one cycle from the first option and two cycles from the second option.

The resulting graph is a multicore with all its cores and complex and simple cycles attached to them. Exception: tests 1–2 from the problem statement have a simplified structure for clarity.

In addition, the following guarantees are given:

- The existence of two disjoint trees directed from the source to the terminals is guaranteed, but it is not guaranteed that they fit into the specified limit by `delay`.
- If there is a directed edge (u, v) with the parameter values `cost` and `delay` in the graph, then there is also an edge (v, u) with the same parameter values.

The graph from the 3rd test is shown below on the figure (click to enlarge). It consists of a core, 10 simple and 3 complex cycles.



Input

The first line of the input consists of a single integer n, the number of vertices ($3 \leq n \leq 60\,000$).

The second line consists of a single integer s, the index of the source vertex ($0 \leq s \leq n - 1$).

The third line consists of a single integer k, the number of the terminals ($1 \leq k \leq \min\{n - 1, 30\}$).

The fourth line consists of k integers $t_1, t_2, \ldots, t_k$, the indices of the terminals ($0 \leq t_i \leq n - 1, t_i \neq s$). All t_i in the line are different.

The fifth line consists of a single integer D, the `delay` bound for all paths from s to t_i ($1 \leq D \leq 1\,000\,000$).

The sixth line consists of a single integer m, the number of edges ($3 \leq m \leq 120\,000$).

The next m lines describe the graph edges. The $(i + 6)$-th line consists of four integers a_i, b_i, c_i, d_i ($0 \leq a_i < b_i \leq n - 1, 1 \leq c_i \leq 200, 1 \leq d_i \leq 4000$). These integers define two directed edges (a_i, b_i) and (b_i, a_i), both having cost c_i and delay d_i.

The graph in every test case is guaranteed to have the above-described structure.

Output

In the first line, print the integer f, the number of directed trees found ($1 \leq f \leq 2$).

After that, print f directed trees. Each tree needs to be described as follows:

In the first line of each tree description, print the integer w, the number of directed edges in a tree.

In the next w lines print tree edges in the following way: in the line $(i+1)$ print two integers (a_i, b_i), the start and end vertices of a directed edge. The edge (a_i, b_i) must exist in the input graph and be printed only once.

The set of printed edges must form a valid directed tree from s to all t_i.

Examples

standard input	standard output
3 0 2 2 1 980 3 0 1 29 415 0 2 35 460 1 2 45 520	2 2 0 1 1 2 2 0 2 2 1
10 9 2 8 7 3418 13 0 2 182 1321 0 3 140 1165 1 3 164 1511 1 2 185 1581 2 3 44 440 2 4 83 760 3 6 54 667 4 5 81 739 4 7 30 376 5 6 69 714 6 9 34 474 7 8 37 546 8 9 33 401	2 2 8 7 9 8 5 4 7 5 4 6 5 9 6 7 8

Note

Please note that using two opposite edges in different trees is acceptable. In the first example, T_1 uses the edge $(1, 2)$, and in T_2 the opposite edge $(2, 1)$ is used. In the second example the edge $(7, 8)$ is used in that way.

Scoring system

Points are scored **for each test case separately** according to the following levels:

1. A single tree — **5 points**.
2. A single tree that meets the `delay` constraint — **10 points**.
3. Two edge-disjoint trees — **20 points**.
4. Two edge-disjoint trees, one of which meets the `delay` constraint — **40 points**.
5. Two edge-disjoint trees, both meeting the `delay` constraint — **100 points**.

The scoring algorithm:

- The achieved level with the highest number $level$ is determined.
- The unconditioned points x for reaching $level$ are awarded.
- The bonus points are awarded according to the formula: $0.25 \cdot x \cdot \left(1 - \sqrt{1 - \frac{bestSum}{participantSum}}\right)$. Here $bestSum$ is the least sum of edge costs obtained on this test by solutions that reached $level$ (but not a higher one); $participantSum$ is the sum of edge costs obtained on this test by the participant.

ATTENTION!!! The LAST solution which passed example test cases will be taken as the final solution of the participant! All solutions prior to the final one will not be retested at the end of the contest and will not be taken into account.

During the contest, $bestSum$ is selected among all submitted solutions (for each test and for each level separately). During the final testing, $bestSum$ will be selected from the final solutions of all participants and only from these solutions.