IOI '11 P4 - Crocodile's Underground City

Time limit: 2.0s **Memory limit:** 256M

Archaeologist Benjamas is running for her life after investigating the mysterious Crocodile's Underground City. The city has N chambers. There are M bidirectional corridors, each connecting a different pair of distinct chambers. Running through different corridors may require different amounts of time. Only K of the N chambers are exit chambers that allow her to escape. Benjamas starts in chamber 0. She wants to reach an exit chamber as quickly as possible.

The Crocodile gatekeeper wants to prevent Benjamas from escaping. From his den, he controls secret doors that can block any *single* corridor. That is, whenever he blocks a new corridor, the previously blocked one has to be reopened.

Benjamas's situation can be described as follows: Each time she tries to leave a chamber, the Crocodile gatekeeper may choose to block one of the corridors adjacent to it. Benjamas then chooses and follows one of the unblocked corridors to the next chamber. Once Benjamas enters a corridor, the Crocodile gatekeeper may not block it until Benjamas reaches the other end. Once she enters the next chamber, the gatekeeper may again choose to block one of the outgoing corridors (possibly the corridor that Benjamas just followed), and so on.

She would like to have a simple escape plan in advance. More precisely, she would like to have a set of instructions that tell her what to do when she gets to a chamber. Let A be one of the chambers. If it is an exit chamber, no instructions are needed–obviously, she can escape the city. Otherwise, the instruction for chamber A should have one of the following forms:

- "If you ever reach chamber A, take the corridor leading to chamber B. However, if that corridor is blocked, then take the corridor leading to chamber C."
- "Don't bother about chamber A; according to this escape plan you cannot possibly reach it."

Note that in some cases (for example, if your plan directs Benjamas to run in a cycle) the gatekeeper may be able to prevent Benjamas from reaching an exit. An escape plan is *good* if Benjamas is guaranteed to reach an exit chamber after a finite amount of time, regardless of what the gatekeeper does. For a good escape plan, let T be the smallest time such that after time T, Benjamas is *guaranteed* to reach an exit. In that case, we say that *the good escape plan takes time* T.

Your task

Write a procedure `travel_plan(N, M, R, L, K, P)` that takes the following parameters:

- N – the number of chambers. The chambers are numbered 0 through $N-1$.
- M – the number of corridors. The corridors are numbered 0 through $M-1$.
- R – a two-dimensional array of integers representing the corridors. For $0 \leq i < M$, corridor i connects two distinct chambers $R[i][0]$ and $R[i][1]$. No two corridors join the same pair of chambers.
- L – a one-dimensional array of integers containing the times needed to traverse the corridors. For $0 \leq i < M$, the value $1 \leq L[i] \leq 1\,000\,000\,000$ is the time Benjamas needs to run through the i^{th} corridor.
- K – the number of exit chambers. You may assume that $1 \leq K < N$.
- P – a one-dimensional array of integers with K distinct entries describing the exit chambers. For $0 \leq i < K$, the value $P[i]$ is the number of the i^{th} exit chamber. Chamber 0 will never be one of the exit chambers.

Your procedure must return the smallest time T for which there exists a good escape plan that takes time T.

You may assume that each non-exit chamber will have at least two corridors leaving it. You may also assume that in each test case there is a good escape plan for which $T \leq 1\,000\,000\,000$.

Examples

Example 1

Consider the case shown in Figure 1, where $N = 5$, $M = 4$, $K = 3$, and

$$R = \begin{matrix} 0 & 1 \\ 0 & 2 \\ 3 & 2 \\ 2 & 4 \end{matrix} \qquad L = \begin{matrix} 2 \\ 3 \\ 1 \\ 4 \end{matrix} \qquad P = \begin{matrix} 1 \\ 3 \\ 4 \end{matrix}$$

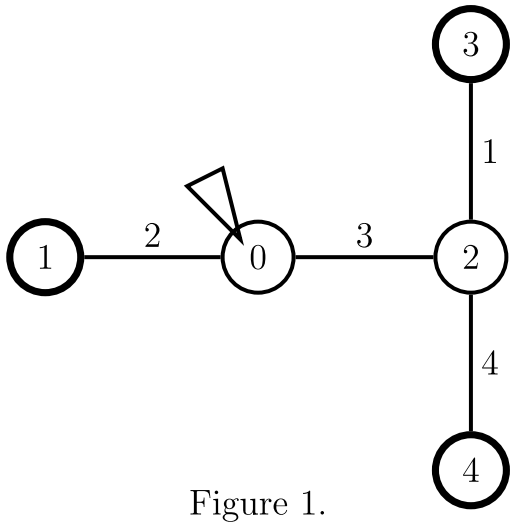

Figure 1.

Chambers are shown as circles, and corridors connecting them are shown as lines. Exit chambers are shown as thick-bordered circles. Benjamas starts at chamber 0 (marked by a triangle). An optimal escape plan is the following one:

- If you ever reach chamber 0, take the corridor leading to chamber 1. However, if that corridor is blocked, then take the corridor leading to chamber 2.
- If you ever reach chamber 2, take the corridor leading to chamber 3. However, if that corridor is blocked, then take the corridor leading to chamber 4.

In the worst case, Benjamas will reach an exit chamber in 7 units of time. Hence, `travel_plan` should return 7.

Example 2

Consider the case shown in Figure 2, where $N = 5$, $M = 7$, $K = 2$, and

$$R = \begin{matrix} 0 & 2 \\ 0 & 3 \\ 3 & 2 \\ 2 & 1 \\ 0 & 1 \\ 0 & 4 \\ 3 & 4 \end{matrix} \qquad L = \begin{matrix} 4 \\ 3 \\ 2 \\ 10 \\ 100 \\ 7 \\ 9 \end{matrix} \qquad P = \begin{matrix} 1 \\ 3 \end{matrix}$$

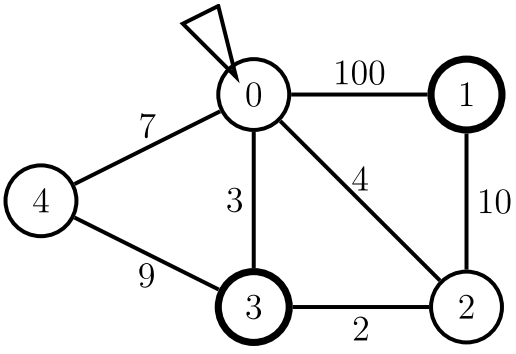

Figure 2.

Here is an optimal escape plan:

- If you ever reach chamber 0, take the corridor leading to chamber 3. However, if that corridor is blocked, then take the corridor leading to chamber 2.
- If you ever reach chamber 2, take the corridor leading to chamber 3. However, if that corridor is blocked, then take the corridor leading to chamber 1.
- Don't bother about chamber 4; according to this escape plan you cannot possibly reach it.

Benjamas will reach one of the exit chambers no later than after 14 units of time. Therefore, `travel_plan` should return 14.

Subtasks

Subtask 1 (46 points)

- $3 \leq N \leq 1\,000$
- The underground city is a tree. That is, $M = N - 1$ and for each pair of chambers i and j there is a sequence of corridors connecting i and j.
- Every exit chamber is connected to exactly one other chamber.
- Any other chamber is connected directly to three or more other chambers.

Subtask 2 (43 points)

- $3 \leq N \leq 1\,000$
- $2 \leq M \leq 100\,000$

Subtask 3 (11 points)

- $3 \leq N \leq 100\,000$
- $2 \leq M \leq 1\,000\,000$

Implementation Details

Interface (API)

```
int travel_plan(int N, int M, int R[][2], int W[], int K, int E[]);
```