Highway Tolls

In Japan, cities are connected by a network of highways. This network consists of N cities and M highways. Each highway connects a pair of distinct cities. No two highways connect the same pair of cities. Cities are numbered from 0 through $N-1$, and highways are numbered from 0 through $M-1$. You can drive on any highway in both directions. You can travel from any city to any other city by using the highways.

A toll is charged for driving on each highway. The toll for a highway depends on the **traffic** condition on the highway. The traffic is either **light** or **heavy**. When the traffic is light, the toll is A yen (Japanese currency). When the traffic is heavy, the toll is B yen. It's guaranteed that $A < B$. Note that you know the values of A and B.

You have a machine which, given the traffic conditions of all highways, computes the smallest total toll that one has to pay to travel between the pair of cities S and T ($S \neq T$), under specified traffic conditions.

However, the machine is just a prototype. The values of S and T are fixed (i.e., hardcoded in the machine) and not known to you. You would like to determine S and T. In order to do so, you plan to specify several traffic conditions to the machine, and use the toll values that it outputs to deduce S and T. Since specifying the traffic conditions is costly, you don't want to use the machine many times.

實現細節

你需要實現下面的程序：

```
find_pair(int N, int[] U, int[] V, int A, int B)
```

- N: 城市的數目。
- U 及 V: 長度為 M 的陣列, 其中 M 為連接城市的高速公路。 對於每個 i ($0 \leq i \leq M-1$), 高速公路 i 連接城市 U[i] 及 V[i]。
- A: 交通順暢時高速公路的收費。
- B: 文通擠塞時高速公路的收費。
- 在每個測試用例中，這程序只會被調用一次。
- 請留意 M 的值為陣列的長度，並能如實現注意事項所示般獲取。

程序 find_pair 能調用以下函數:

```
int64 ask(int[] w)
```

- `w` 的長度一定為 M。 陣列 `w` 描述高速公路的交通狀況。
- 對於每個 i ($0 \le i \le M-1$), `w[i]` 描述高速公路 i 的交通狀況。 `w[i]` 的值一定為 0 或 1。
  - `w[i] = 0` 表示高速公路 i 交通順暢。
  - `w[i] = 1` 表示高速公路 i 交通擠塞。
- 該函數的返回值是在 `w` 所註明的交通狀況下，來往城市 S 及 T 每一單程所需的最少收費。
- 該函數最多只能被調用 100 次（對於每個測試用例）。

`find_pair` 應調用以下程序以報告答案：

```
answer(int s, int t)
```

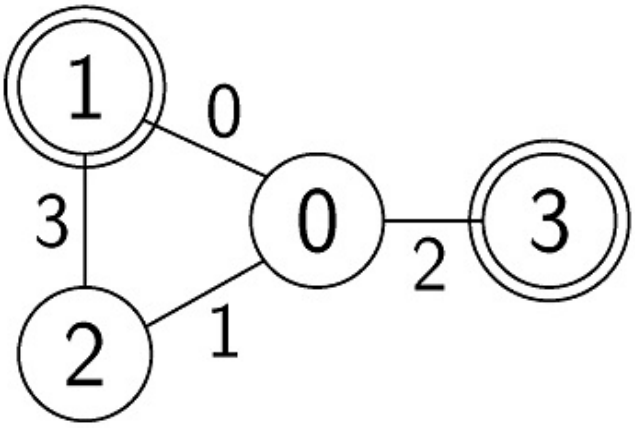
- `s` 及 `t` 一定為該對城市 S 及 T（兩者的先後次序並不重要）。
- 該程序一定會被調用及只會被調用一次。

如果不滿足上面的條件，你的程式將被判視 **`Wrong Answer`**。否則，你的程式將被判為 **`Accepted`**，而你的得分將根據 `ask` 的調用次數來計算（參見子任務）。

Example

Let $N = 4$, $M = 4$, $U = [0, 0, 0, 1]$, $V = [1, 2, 3, 2]$, $A = 1$, $B = 3$, $S = 1$, and $T = 3$.

The grader calls `find_pair(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3)`.



Some possible calls to `ask` and the corresponding return values are listed below:

Call	Return
`ask([0, 0, 0, 0])`	2
`ask([0, 1, 1, 0])`	4
`ask([1, 0, 1, 0])`	5
`ask([1, 1, 1, 1])`	6

For the function call `ask([0, 0, 0, 0])`, the traffic of every highway is light and the toll for it is 1. The cheapest route from $S = 1$ to $T = 3$ is $1 \to 0 \to 3$. The total toll for this path is 2. Thus, this function returns 2.

For a correct answer, the procedure `find_pair` should call `answer(1, 3)` or `answer(3, 1)`.

The file `sample-01-in.txt` in the zipped attachment package corresponds to this example. Other sample inputs are also available in the package.

Constraints

- $2 \leq N \leq 90\,000$
- $1 \leq M \leq 130\,000$
- $1 \leq A < B \leq 1\,000\,000\,000$
- For each $0 \leq i \leq M - 1$
  - $0 \leq U[i] \leq N - 1$
  - $0 \leq V[i] \leq N - 1$
  - $U[i] \neq V[i]$
- $(U[i], V[i]) \neq (U[j], V[j])$ and $(U[i], V[i]) \neq (V[j], U[j])$ $(0 \leq i < j \leq M - 1)$
- You can travel from any city to any other city by using the highways.
- $0 \leq S \leq N - 1$
- $0 \leq T \leq N - 1$
- $S \neq T$

In this problem, the grader is NOT adaptive. This means that S and T are fixed at the beginning of the running of the grader and they do not depend on the queries asked by your solution.

Subtasks

1. (5 points) one of S or T is 0, $N \leq 100$, $M = N - 1$
2. (7 points) one of S or T is 0, $M = N - 1$
3. (6 points) $M = N - 1$, $U[i] = i$, $V[i] = i + 1$ $(0 \leq i \leq M - 1)$
4. (33 points) $M = N - 1$
5. (18 points) $A = 1$, $B = 2$

6. (31 points) No additional constraints

Assume your program is judged as **`Accepted`**, and make X calls to `ask`. Then your score P for the test case, depending on its subtask number, is calculated as follows:

- Subtask 1. $P = 5$.
- Subtask 2. If $X \leq 60$, $P = 7$. Otherwise $P = 0$.
- Subtask 3. If $X \leq 60$, $P = 6$. Otherwise $P = 0$.
- Subtask 4. If $X \leq 60$, $P = 33$. Otherwise $P = 0$.
- Subtask 5. If $X \leq 52$, $P = 18$. Otherwise $P = 0$.
- Subtask 6.
  - If $X \leq 50$, $P = 31$.
  - If $51 \leq X \leq 52$, $P = 21$.
  - If $53 \leq X$, $P = 0$.

Note that your score for each subtask is the minimum of the scores for the test cases in the subtask.

Sample grader

The sample grader reads the input in the following format:

- line 1: $N\ M\ A\ B\ S\ T$
- line $2 + i$ ($0 \leq i \leq M - 1$): $U[i]\ V[i]$

If your program is judged as **`Accepted`**, the sample grader prints `Accepted: q`, with `q` the number of calls to `ask`.

If your program is judged as **`Wrong Answer`**, it prints `Wrong Answer: MSG`, where `MSG` is one of:

- `answered not exactly once`: The procedure `answer` was not called exactly once.
- `w is invalid`: The length of `w` given to `ask` is not M or `w[i]` is neither 0 nor 1 for some i ($0 \leq i \leq M - 1$).
- `more than 100 calls to ask`: The function `ask` is called more than 100 times.
- `{s, t} is wrong`: The procedure `answer` is called with an incorrect pair `s` and `t`.