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 the 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.

Implementation details

You should implement the following procedure:

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

- **N**: the number of cities.
- **U** and **V**: arrays of length $M$, where $M$ is the number of highways connecting cities. For each $i$ ($0 \leq i \leq M - 1$), the highway $i$ connects the cities $U[i]$ and $V[i]$.
- **A**: the toll for a highway when the traffic is light.
- **B**: the toll for a highway when the traffic is heavy.
- This procedure is called exactly once for each test case.
- Note that the value of $M$ is the lengths of the arrays, and can be obtained as indicated in the implementation notice.
The procedure \texttt{find\_pair} can call the following function:

\begin{verbatim}
int64 ask(int[] w)
\end{verbatim}

- The length of \( w \) must be \( M \). The array \( w \) describes the traffic conditions.
- For each \( i \) (\( 0 \leq i \leq M - 1 \)), \( w[i] \) gives the traffic condition on the highway \( i \). The value of \( w[i] \) must be either 0 or 1.
  - \( w[i] = 0 \) means the traffic of the highway \( i \) is light.
  - \( w[i] = 1 \) means the traffic of the highway \( i \) is heavy.
- This function returns the smallest total toll for travelling between the cities \( S \) and \( T \), under the traffic conditions specified by \( w \).
- This function can be called at most 100 times (for each test case).

\texttt{find\_pair} should call the following procedure to report the answer:

\begin{verbatim}
answer(int s, int t)
\end{verbatim}

- \( s \) and \( t \) must be the pair \( S \) and \( T \) (the order does not matter).
- This procedure must be called exactly once.

If some of the above conditions are not satisfied, your program is judged as \textbf{Wrong Answer}. Otherwise, your program is judged as \textbf{Accepted} and your score is calculated by the number of calls to \texttt{ask} (see Subtasks).

**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 \texttt{find\_pair}(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3).

In the figure above, the edge with number \( i \) corresponds to the highway \( i \). Some possible calls to ask and the corresponding return values are listed below:

\begin{figure}[ht]
\centering
\includegraphics[width=0.5\textwidth]{figure.png}
\caption{Highway (2 of 4)}
\end{figure}
For the function call \( ask([\theta, \theta, \theta, \theta]) \), the traffic of each highway is light and the toll for each highway is 1. The cheapest route from \( S = 1 \) to \( T = 3 \) is \( 1 \to 0 \to 3 \). The total toll for this route is 2. Thus, this function returns 2.

For a correct answer, the procedure \texttt{find_pair} should call \texttt{answer(1, 3)} or \texttt{answer(3, 1)}.

The file \texttt{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 makes \( X \) 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 \( \text{answer} \) 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 \( \text{ask} \) is called more than 100 times.
- **\( \{s, t\} \) is wrong**: The procedure \( \text{answer} \) is called with an incorrect pair \( s \) and \( t \).