高速道路の通行料金

日本では, N 個の都市が M 本の高速道路によってつながっている. 各高速道路は, 異なる 2 つの都市をつないでいる. ある都市の組をつなぐ高速道路はたかだか 1 本である. 都市には 0 から $N-1$ までの番号が付けられており, 高速道路には 0 から $M-1$ までの番号が付けられている. 高速道路は双方向に通ることができる. どの 2 つの異なる都市の間もいくつかの高速道路を使って移動することができる.

それぞれの高速道路を通るたびに, 通行料金が発生する. 通行料金は, そのときの高速道路の**交通量**によって変化する. 交通量は**少ない**か**多い**かのいずれかである. 交通量が少ないとき, 通行料金は A 円である. 交通量が多いとき, 通行料金は B 円である. ここで, $A < B$ であることが保証される. ここで, A と B は与えられる値であることに注意せよ.

あなたは, すべての高速道路の交通量のデータを与えると, その交通量で S 番目の都市から T 番目の都市へ ($S \neq T$) 高速道路で移動するときに必要な通行料金の総和の最小値を計算する装置を持っている.

しかし, この装置はまだ試作品である. 値 S と T は装置の内部で固定されているが, あなたはその値を知らない. あなたは S と T を特定しようと考えた. そのために, 装置に高速道路の交通量のデータを複数回与え, それぞれの計算結果を利用することで S と T を特定することにした. 交通量を具体的に指定するのはコストがかかるため, あまり装置を使わずに S と T を特定したい.

実装の詳細

あなたは, 以下のプロシージャを実装する必要がある:

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

- N: 都市の数である.
- U, V: 長さ M の配列である. M は都市をつなぐ高速道路の本数である. 各 i について ($0 \leqq i \leqq M-1$), i 番目の高速道路は U[i] 番目の都市と V[i] 番目の都市をつないでいる.
- A: 交通量が少ないときの通行料金である.
- B: 交通量が多いときの通行料金である.
- このプロシージャはそれぞれのテストケースに対し, ちょうど 1 回呼び出される.
- なお, ここで M は配列の長さを表す値であり, これは「実装上の注意」に記されている方法で取得することができる.

プロシージャ find_pair は以下の関数を呼び出すことができる:

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

- w の長さはちょうど M でなければならない. w は交通量を表す配列である.
- 各 i ($0 \leqq i \leqq M-1$)について, w[i] は i 番目の高速道路の交通量を表す. w[i] の値は 0 か 1 でなければならない.
  - w[i] = 0 であるとき, i 番目の高速道路の交通量が少ないということを表す.
  - w[i] = 1 であるとき, i 番目の高速道路の交通量が多いということを表す.
- この関数は, w で指定された交通量で, 都市 S から T へ高速道路で移動するときに必要な通行料金の総和の最小値を返す.
- この関数はそれぞれのテストケースにつき 100 回まで呼び出すことができる.

find_pair は, 答えを報告するために以下のプロシージャを呼び出す必要がある:

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

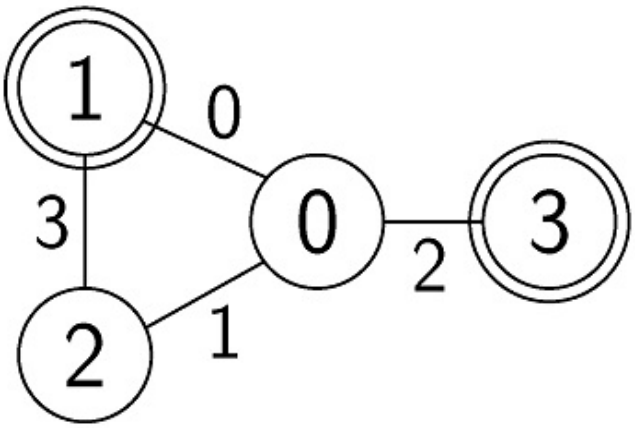
- s と t は S と T の組でなければならない (順不同).
- このプロシージャはちょうど 1 回呼び出されなければならない.

上記の条件のいずれかが満たされていない場合, あなたのプログラムは **Wrong Answer** と判定される. 条件がすべて満たされている場合, あなたのプログラムは **Accepted** と判定され, 関数 ask の呼び出し回数に応じて得点が計算される (「小課題」の節を参照).

入出力例

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

採点プログラムは find_pair(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3) を呼び出す.



上の図において, i と番号付けられた辺は i 番目の高速道路に対応する. やりとりの例を以下に示す.

呼び出し	返り値
`ask([0, 0, 0, 0])`	2
`ask([0, 1, 1, 0])`	4
`ask([1, 0, 1, 0])`	5
`ask([1, 1, 1, 1])`	6

最初の呼び出し `ask([0, 0, 0, 0])` において, それぞれの高速道路の交通量は少なく, その通行料金は 1 円である. $S = 1$ から $T = 3$ まで最小の費用で移動する経路は $1 \to 0 \to 3$ である. この経路での費用は 2 円である. よって, このとき関数は 2 を返す.

正しい解答をするためには, `find_pair` は `answer(1, 3)` もしくは `answer(3, 1)` をちょうど 1 回呼び出す必要がある.

zip 圧縮された添付パッケージ (attachment package) に入ったファイル `sample-01-in.txt` がこの例に対応している. 他の入出力例もこのパッケージで得られる.

制約

- $2 \leqq N \leqq 90\,000$.
- $1 \leqq M \leqq 130\,000$.
- $1 \leqq A < B \leqq 1\,000\,000\,000$.
- 各 $0 \leqq i \leqq M - 1$ に対して,
  - $0 \leqq U[i] \leqq N - 1$.
  - $0 \leqq V[i] \leqq N - 1$.
  - $U[i] \neq V[i]$.
- $(U[i], V[i]) \neq (U[j], V[j])$ かつ $(U[i], V[i]) \neq (V[j], U[j])$ $(0 \leqq i < j \leqq M - 1)$.
- どの 2 つの異なる都市の間もいくつかの高速道路を使って移動することができる.
- $0 \leqq S \leqq N - 1$.
- $0 \leqq T \leqq N - 1$.
- $S \neq T$.

この課題では, 採点プログラムは adaptive ではない. つまり, S と T は採点プログラムの実行開始時に固定されており, あなたのプログラムがする質問によって変化することはない.

小課題

1. (5 点) S もしくは T は 0 であり, $N \leqq 100, M = N - 1$.
2. (7 点) S もしくは T は 0 であり, $M = N - 1$.
3. (6 点) $M = N - 1, U[i] = i, V[i] = i + 1$ $(0 \leqq i \leqq M - 1)$.
4. (33 点) $M = N - 1$.
5. (18 点) $A = 1, B = 2$.
6. (31 点) 追加の制約はない.

あなたのプログラムが **Accepted** と判定され, ask を X 回呼び出したとする. そのとき, それぞれのテストケースの得点 P は, それぞれの小課題に応じて以下のように計算される.

- 小課題 1. $P = 5$.
- 小課題 2. $X \leqq 60$ であるとき, $P = 7$. そうでないとき, $P = 0$.
- 小課題 3. $X \leqq 60$ であるとき, $P = 6$. そうでないとき, $P = 0$.
- 小課題 4. $X \leqq 60$ であるとき, $P = 33$. そうでないとき, $P = 0$.
- 小課題 5. $X \leqq 52$ であるとき, $P = 18$. そうでないとき, $P = 0$.
- 小課題 6.
  - $X \leqq 50$ であるとき, $P = 31$.
  - $51 \leqq X \leqq 52$ であるとき, $P = 21$.
  - $53 \leqq X$ であるとき, $P = 0$.

各小課題の最終的な得点は, その小課題のそれぞれのテストケースでの得点の最小値となることに注意せよ.

採点プログラムのサンプル

採点プログラムのサンプルの入力形式は以下の通りである.

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

あなたのプログラムが **Accepted** と判定されるならば, 採点プログラムのサンプルは ask を呼び出した回数を q として, `Accepted: q` と出力する.

あなたのプログラムが **Wrong Answer** と判定されるならば, 採点プログラムのサンプルは `Wrong Answer: MSG` と出力する. MSG の部分には状況に応じて以下のいずれかのメッセージが入る.

- `answered not exactly once`: プロシージャ answer が呼び出された回数がちょうど 1 回ではなかった.
- `w is invalid`: 関数 ask に渡された配列 w の長さが M ではない, もしくはある i に対して `w[i]` が 0 でも 1 でもなかった ($0 \leqq i \leqq M - 1$).
- `more than 100 calls to ask`: 関数 ask が 100 回を超えて呼び出された.
- `{s, t} is wrong`: プロシージャ answer が間違った s と t の組で呼び出された.