



# Highway Tolls

No Japão, as cidades estão ligadas por uma rede de estradas. Esta rede consiste de  $N$  cidades e  $M$  estradas. Cada estrada liga um par de cidades distintas. Entre cada par de cidades há no máximo uma estrada. As cidades são numeradas de 0 a  $N - 1$  e as estradas estão numeradas de 0 a  $M - 1$ . Você pode conduzir em cada estrada em ambas direções. Você pode viajar de qualquer cidade para qualquer cidade usando as estradas.

Uma pedágio é cobrada por conduzir em cada estrada. A pedágio para uma estrada depende das condições de **trânsito** na estrada. O trânsito está ou **leve** ou **pesado**. Quando o trânsito está leve, a pedágio cobrada é de  $A$  yen. Quando o trânsito está pesado, a pedágio cobrada é de  $B$  yen. É garantido que  $A < B$ . Note que você sabe os valores de  $A$  e  $B$ .

Você tem uma máquina que, dadas as condições de trânsito em todas as estradas, calcula o menor custo total das pedágios que tem de pagar para viajar entre o par de cidades  $S$  e  $T$  ( $S \neq T$ ), com as condições de trânsito especificadas.

Contudo, a máquina é apenas um protótipo. Os valores de  $S$  e  $T$  são fixos (isto é, 'hardcoded' na máquina) e desconhecidos por você. Você gostaria de determinar  $S$  e  $T$ . Para tal, você quer especificar várias condições de trânsito à máquina e usar os valores de pedágios que ela retorna para deduzir  $S$  e  $T$ . Visto que especificar as condições de trânsito é custoso, você não quer usar a máquina muitas vezes.

## Detalhes de implementação

Você deve implementar a seguinte função:

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

- $N$ : o número de cidades.
- $U$  e  $V$ : arrays de comprimento  $M$ , onde  $M$  é o número de estradas que ligam cidades. Para cada  $i$  ( $0 \leq i \leq M - 1$ ), a estrada  $i$  liga as cidades  $U[i]$  e  $V[i]$ .
- $A$ : a pedágio para a estrada quando o trânsito está leve.
- $B$ : a pedágio para a estrada quando o trânsito está pesado.
- Esta função é chamada exatamente uma vez por caso de teste.
- Note que o valor de  $M$  é os comprimentos dos arrays e pode ser obtido como indicado na nota de implementação.

A função `find_pair` pode chamar a seguinte função:

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

- O comprimento de  $w$  deve ser  $M$ . O array  $w$  descreve as condições de trânsito.
- Para cada  $i$  ( $0 \leq i \leq M - 1$ ),  $w[i]$  representa as condições de trânsito na auto estrada  $i$ . O valor de  $w[i]$  deve ser ou 0 ou 1.
  - $w[i] = 0$  significa que o trânsito na estrada  $i$  está leve.
  - $w[i] = 1$  significa que o trânsito na estrada  $i$  está pesado.
- Esta função retorna o menor custo total das pedágios que tem de pagar para viajar entre as cidades  $S$  e  $T$ , segundo as condições de trânsito especificadas por  $w$ .
- Esta função pode ser chamada no máximo 100 vezes (por cada caso de teste).

`find_pair` deve chamar a seguinte função para reportar a resposta:

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

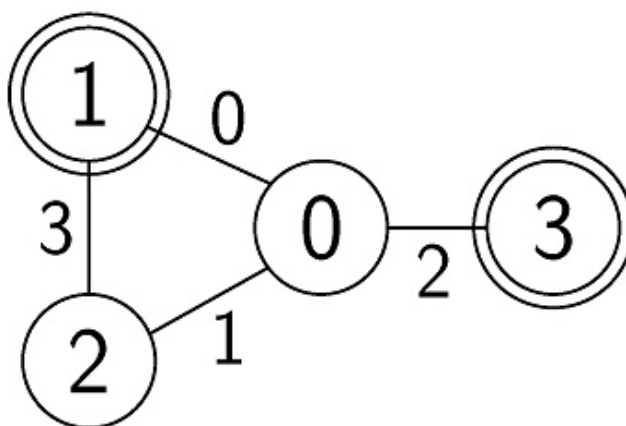
- $s$  e  $t$  devem ser o par  $S$  e  $T$  (a ordem não interessa).
- Esta função deve ser chamada exatamente uma vez.

Se alguma das condições acima não for satisfeita, o seu programa é avaliado com **Wrong Answer**. Caso contrário, o seu programa é avaliado como **Accepted** e a sua pontuação é calculada pelo número de chamadas a `ask` (ver Subtarefas).

## Exemplo

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

O avaliador chama `find_pair(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3)`.



Na figura acima, a aresta com número  $i$  corresponde à estrada  $i$ . Algumas possíveis chamadas a `ask` e os valores de retorno correspondentes seguem:

Chamada	Retorno
<code>ask([0, 0, 0, 0])</code>	2
<code>ask([0, 1, 1, 0])</code>	4
<code>ask([1, 0, 1, 0])</code>	5
<code>ask([1, 1, 1, 1])</code>	6

Para a chamada da função `ask([0, 0, 0, 0])`, o trânsito em cada estrada é leve e a pedágio em cada estrada tem custo 1. O caminho mais curto de  $S = 1$  para  $T = 3$  é  $1 \rightarrow 0 \rightarrow 3$ . A pedágio total para este caminho é 2. Assim, esta função retorna 2.

Para uma resposta correta, a função `find_pair` deve chamar `answer(1, 3)` ou `answer(3, 1)`.

O ficheiro `sample-01-in.txt` no arquivo zip em anexo corresponde a este exemplo. Outros inputs de exemplo estão disponíveis no arquivo.

## Restrições

- $2 \leq N \leq 90\,000$
- $1 \leq M \leq 130\,000$
- $1 \leq A < B \leq 1\,000\,000\,000$
- Para cada  $0 \leq i \leq M - 1$ 
  - $0 \leq U[i] \leq N - 1$
  - $0 \leq V[i] \leq N - 1$
  - $U[i] \neq V[i]$
- $(U[i], V[i]) \neq (U[j], V[j])$  e  $(U[i], V[i]) \neq (V[j], U[j])$  ( $0 \leq i < j \leq M - 1$ )
- Você pode viajar de cada cidade para qualquer outra cidade usando as estradas.
- $0 \leq S \leq N - 1$
- $0 \leq T \leq N - 1$
- $S \neq T$

Neste problema, o avaliador NÃO é adaptativo. Isto significa que  $S$  e  $T$  estão fixos no início de correr o avaliador e não dependem das perguntas feitas pela sua solução.

## Subtarefas

1. (5 pontos) um de  $S$  ou  $T$  é 0,  $N \leq 100$ ,  $M = N - 1$
2. (7 pontos) um de  $S$  ou  $T$  é 0,  $M = N - 1$
3. (6 pontos)  $M = N - 1$ ,  $U[i] = i$ ,  $V[i] = i + 1$  ( $0 \leq i \leq M - 1$ )
4. (33 pontos)  $M = N - 1$
5. (18 pontos)  $A = 1$ ,  $B = 2$
6. (31 pontos) Não há restrições adicionais

Suponha que o seu programa é avaliado como **Accepted** e faz  $X$  chamadas a `ask`. Então a sua pontuação  $P$  para o caso de teste, dependendo do seu número de subtarefa, é calculado da seguinte forma:

- Subtarefa 1.  $P = 5$ .
- Subtarefa 2. Se  $X \leq 60$ ,  $P = 7$ . Caso contrário  $P = 0$ .
- Subtarefa 3. Se  $X \leq 60$ ,  $P = 6$ . Caso contrário  $P = 0$ .
- Subtarefa 4. Se  $X \leq 60$ ,  $P = 33$ . Caso contrário  $P = 0$ .
- Subtarefa 5. Se  $X \leq 52$ ,  $P = 18$ . Caso contrário  $P = 0$ .
- Subtarefa 6.
  - Se  $X \leq 50$ ,  $P = 31$ .
  - Se  $51 \leq X \leq 52$ ,  $P = 21$ .
  - Se  $53 \leq X$ ,  $P = 0$ .

Note que a sua pontuação para cada subtarefa é o mínimo das pontuações para os casos de teste na subtarefa.

## Avaliador de exemplo

O avaliador de exemplo lê o input no seguinte formato:

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

Se o seu programa for avaliado como **Accepted**, o avaliador de exemplo imprime `Accepted: q`, onde  $q$  é o número de chamadas a `ask`.

Se o seu programa for avaliado como **Wrong Answer**, é imprimido `Wrong Answer: MSG`, onde `MSG` é uma de:

- `answered not exactly once`: Se a função `answer` não foi chamada exatamente uma vez.
- `w is invalid`: O comprimento de `w` dado a `ask` não foi  $M$  ou  $w[i]$  não é 0 ou 1 para algum  $i$  ( $0 \leq i \leq M - 1$ ).
- `more than 100 calls to ask`: A função `ask` foi chamada mais de 100 vezes.
- `{s, t} is wrong`: A função `answer` foi chamada com o par incorreto de `s` e `t`.