Autostradowe opłaty

W Japonii miasta połączone są siecią autostrad. Sieć ta składa się z N miast oraz M autostrad. Każda autostrada łączy parę różnych miast. Żadne dwie autostrady nie łączą tej samej pary miast. Miasta ponumerowane są od 0 do $N-1$, natomiast autostrady od 0 do $M-1$. Każdą autostradą można przejechać w obu kierunkach. Z każdego miasta można dojechać do każdego innego, używając autostrad.

Za przejazd każdą autostradą pobierana jest opłata. Opłata ta zależy od **natężenia ruchu drogowego** na tej autostradzie. Natężenie ruchu na autostradzie może być zarówno **małe**, jak i **duże**. Kiedy natężenie jest małe, opłata wynosi A yenów (Japońska waluta). Kiedy natomiast natężenie jest duże, opłata wynosi B yenów. Zagwarantowane jest, że $A < B$. Zwróć uwagę, że wartości A i B są Tobie znane.

Posiadasz maszynę, która dla danego scenariusza natężeń ruchu drogowego na każdej autostradzie, oblicza najmniejszą możliwą sumaryczną opłatę, którą trzeba zapłacić, aby przejechać pomiędzy miastami S i T ($S \neq T$) zgodnie z ustalonymi natężeniami ruchu.

Maszyna jest jednak na razie tylko prototypem. Wartości S i T są stałe (tj. wpisane na sztywno w maszynie) i nie są Tobie znane. Chciałbyś ustalić S oraz T. Planujesz więc spreparować kilka scenariuszy natężenia ruchu drogowego dla maszyny i podać je maszynie, a następnie, na podstawie uzyskanych odpowiedzi maszyny, ustalić S oraz T. Nie chcesz używać maszyny zbyt wiele razy, jako że przygotowanie scenariuszy natężenia ruchu drogowego jest dość kosztowne.

Szczegóły implementacyjne

Powinieneś zaimplementować następującą procedurę:

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

- `N`: liczba miast.
- `U` oraz `V`: tablice długości M, gdzie M jest liczbą autostrad łączących miasta. Dla każdego i ($0 \leq i \leq M-1$), autostrada i łączy miasta `U[i]` oraz `V[i]`.
- `A`: opłata za przejazd autostradą, jeżeli natężenie ruchu jest małe.
- `B`: opłata za przejazd autostradą, jeżeli natężenie ruchu jest duże.
- Procedura ta jest wywoływana dokładnie raz dla każdego testu.
- Zauważ, że wartość M jest długością tablic i może zostać uzyskana, jak zostało to

opisane w *Uwagach implementacyjnych*.

Procedura `find_pair` może wywoływać następującą funkcję:

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

- Długość `w` musi być równa M. Tablica `w` określa scenariusz natężenia ruchu drogowego.
- Dla każdego i ($0 \leq i \leq M-1$) `w[i]` określa natężenie ruchu drogowego na i-tej autostradzie. Wartość `w[i]` musi być równa 0 lub 1.
  - `w[i] = 0` oznacza, że natężenie ruchu na i-tej autostradzie jest małe.
  - `w[i] = 1` oznacza, że natężenie ruchu na i-tej autostradzie jest duże.
- Funkcja ta zwraca najmniejszą sumaryczną opłatę za przejazd pomiędzy miastami S oraz T przy natężeniu ruchu określonym przez `w`.
- Funkcja ta może być wywołana co najwyżej 100 razy (dla każdego testu).

Aby udzielić odpowiedzi procedura `find_pair` powinna wywołać następującą procedurę :

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

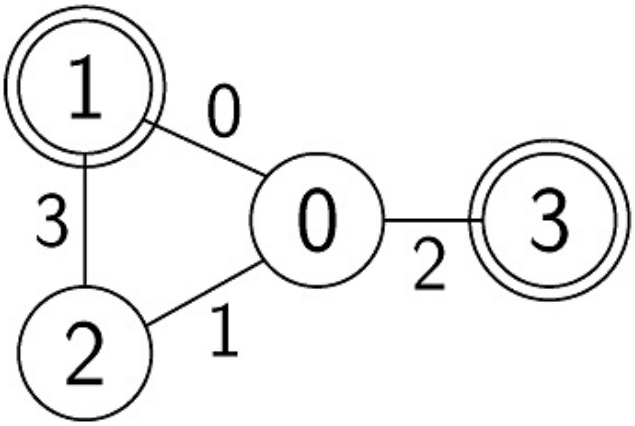
- `s` oraz `t` muszą być parą S oraz T (kolejność nie ma znaczenia).
- Procedura ta musi być wywołana dokładnie raz.

Jeżeli któryś z powyższych warunków nie jest spełniony, Twój program dostanie werdykt **`Wrong Answer`**. W przeciwnym wypadku, Twój program dostanie werdykt **`Accepted`** i Twój wynik będzie zależał od liczby wywołań funkcji `ask` (zobacz sekcję Podzadania).

Przykład

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

Sprawdzaczka wywołuje `find_pair(4, [0, 0, 0, 1], [1, 2, 3, 2], 1, 3)`.



Na rysunku powyżej, krawędź o numerze i odpowiada i-tej autostradzie. Możliwe wywołania funkcji `ask` oraz odpowiadające im wyniki zostały przedstawione poniżej:

Wywołanie	Zwrócona wartość
`ask([0, 0, 0, 0])`	2
`ask([0, 1, 1, 0])`	4
`ask([1, 0, 1, 0])`	5
`ask([1, 1, 1, 1])`	6

Dla wywołania funkcji `ask([0, 0, 0, 0])`, natężenie ruchu na każdej autostradzie jest małe i opłata za przejazd dla każdej autostrady wynosi 1. Najtańsza trasa z $S = 1$ do $T = 3$ to $1 \to 0 \to 3$. Sumaryczna opłata za przejazd tą trasą wynosi 2. Stąd zwrócona wartość wynosi 2.

Aby poprawnie odpowiedzieć, procedura `find_pair` powinna wywołać procedurę `answer(1, 3)` albo `answer(3, 1)`.

Plik `sample-01-in.txt` w załączonym pakiecie w formacie zip odpowiada temu przykładowi. Inne przykładowe wejścia również są dostępne w tym pakiecie.

Ograniczenia

- $2 \le N \le 90\,000$
- $1 \le M \le 130\,000$
- $1 \le A < B \le 1\,000\,000\,000$
- Dla każdego $0 \le i \le M - 1$
  - $0 \le U[i] \le N - 1$
  - $0 \le V[i] \le N - 1$
  - $U[i] \neq V[i]$
- $(U[i], V[i]) \neq (U[j], V[j])$ oraz $(U[i], V[i]) \neq (V[j], U[j])$ $(0 \le i < j \le M - 1)$

- Z każdego miasta możesz przejechać do każdego innego używając autostrad.
- $0 \leq S \leq N - 1$
- $0 \leq T \leq N - 1$
- $S \neq T$

W tym zadaniu sprawdzaczka NIE jest adaptacyjna. Oznacza to, że S oraz T są ustalone przy uruchomieniu programu sprawdzającego i nie zależą od zapytań zadanych przez Twój program.

Podzadania

1. (5 punktów) jedno z S i T jest równe 0, $N \leq 100$, $M = N - 1$
2. (7 punktów) jedno z S i T jest równe 0, $M = N - 1$
3. (6 punktów) $M = N - 1$, $U[i] = i$, $V[i] = i + 1$ $(0 \leq i \leq M - 1)$
4. (33 punkty) $M = N - 1$
5. (18 punktów) $A = 1$, $B = 2$
6. (31 punktów) Brak dodatkowych ograniczeń.

Zakładając, że Twój program dostał werdykt **`Accepted`** oraz wykonał X wywołań funkcji `ask`, Twój wynik P dla danego testu jest zależny od numeru podzadania i jest obliczany w następujący sposób:

- Podzadanie 1. $P = 5$.
- Podzadanie 2. Dla $X \leq 60$, $P = 7$. W przeciwnym wypadku $P = 0$.
- Podzadanie 3. Dla $X \leq 60$, $P = 6$. W przeciwnym wypadku $P = 0$.
- Podzadanie 4. Dla $X \leq 60$, $P = 33$. W przeciwnym wypadku $P = 0$.
- Podzadanie 5. Dla $X \leq 52$, $P = 18$. W przeciwnym wypadku $P = 0$.
- Podzadanie 6.
  - Dla $X \leq 50$, $P = 31$.
  - Dla $51 \leq X \leq 52$, $P = 21$.
  - Natomiast dla $53 \leq X$, $P = 0$.

Pamiętaj, że Twój wynik dla każdego podzadania to minimum z wyników dla poszczególnych testów w tym podzadaniu.

Przykładowa sprawdzaczka

Przykładowa sprawdzaczka wczytuje wejście w następującym formacie:

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

Jeżeli Twój program dostanie werdykt **`Accepted`**, przykładowa sprawdzaczka wypisuje `Accepted: q`, gdzie `q` to liczba wywołań funkcji `ask`.

Jeżeli Twój program dostanie werdykt **`Wrong Answer`**, sprawdzaczka wypisuje `Wrong Answer: MSG`, gdzie znaczenie `MSG` jest następujące:

- `answered not exactly once`: Procedura `answer` nie została wywołana dokładnie raz.
- `w is invalid`: Długość argumentu `w` funkcji `ask` nie jest równa M, bądź `w[i]` nie jest równe 0 lub 1 dla pewnego i ($0 \leq i \leq M-1$).
- `more than 100 calls to ask`: Funkcja `ask` została wywołana więcej niż 100 razy.
- `{s, t} is wrong`: Procedura `answer` została wywołana z nieprawidłową parą `s` i `t`.