Permainan Permutasi

Alice dan Bob adalah teman masa kecil, dan mereka senang bermain permainan asah otak. Hari ini, mereka sedang bermain permainan baru yang melibatkan graf.

Dalam permainan ini, terdapat sebuah graf **terhubung** yang terdiri dari m buah verteks, dinomori dari 0 hingga $m-1$; dan e buah *edge*, dinomori dari 0 hingga $e-1$. *Edge* ke-i menghubungkan verteks $u[i]$ dan $v[i]$.

Dalam permainan ini terdapat juga sebuah permutasi p dari $[0, 1, \ldots, n-1]$, dengan $m \leq n$. Sebuah permutasi dari $[0, 1, \ldots, n-1]$ merupakan suatu *array* dengan panjang n sedemikian sehingga setiap bilangan bulat dari 0 hingga $n-1$ muncul tepat sekali di *array* tersebut. **Nilai** dari sebuah permutasi p adalah banyaknya indeks i sehingga $p[i] = i$.

Permainan ini berlangsung selama paling banyak 10^{100} putaran. Pada setiap putaran, terjadi hal-hal berikut:

1. Jika Alice memilih untuk menghentikan permainan, maka permainan berhenti.
2. Selebihnya, Alice memilih indeks-indeks $t[0], t[1], \ldots t[m-1]$ yang **saling berbeda** di mana $0 \leq t[i] < n$. Perhatikan bahwa **tidak perlu** terpenuhi bahwa $t[0] < t[1] < \ldots < t[m-1]$.
3. Bob memilih sebuah indeks $0 \leq j < e$ dari *edge* pada graf, lalu menukar $p[t[u[j]]$ dan $p[t[v[j]]]$.

Alice ingin memaksimalkan nilai akhir dari permutasi p, sedangkan Bob ingin meminimalkannya.

Tugas Anda adalah membantu Alice bermain melawan Bob, yang langkahnya akan disimulasikan oleh *grader*.

Didefinisikan *nilai optimal* sebagai nilai akhir permutasi p jika Alice dan Bob keduanya bermain secara optimal.

Anda harus menentukan nilai optimal dari permainan ini, lalu Anda juga harus bermain melawan Bob untuk mencapai nilai yang **setidaknya** sebesar nilai optimal tersebut setelah sejumlah putaran.

Perhatikan bahwa apa pun langkah yang dilakukan Bob, strategi Alice tetap harus bekerja, termasuk jika Bob melakukan langkah yang tidak optimal.

Detail Implementasi

Anda harus mengimplementasikan prosedur berikut:

```
int Alice(int m, int e, std::vector<int> u, std::vector<int> v,
          int n, std::vector<int> p)
```

- m: banyaknya verteks pada graf.
- e: banyaknya *edge* pada graf.
- u, v: *array* sepanjang e yang menyatakan *edge-edge* pada graf.
- n: panjang permutasi p.
- p: *array* sepanjang n yang merupakan permutasi dari $[0, 1, \ldots, n-1]$.
- Prosedur ini dipanggil tepat sekali untuk setiap kasus uji.
- Prosedur ini harus mengembalikan sebuah bilangan bulat yang menyatakan nilai optimal permainan ini.

Prosedur di atas dapat memanggil fungsi berikut:

```
int Bob(std::vector<int> t)
```

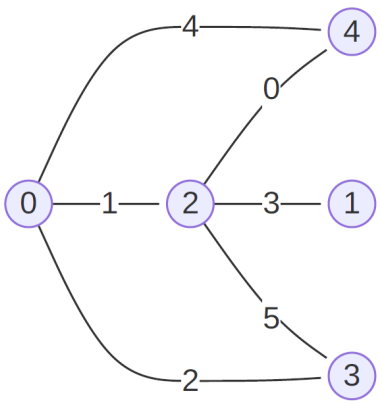
- t: *array* sepanjang m yang berisi indeks-indeks yang saling berbeda, dengan $0 \le t[i] < n$ dan $t[i] \ne t[j]$ jika $i \ne j$.
- Fungsi ini mengembalikan suatu bilangan bulat j yang memenuhi $0 \le j < e$, yang menyatakan pilihan Bob.
- Fungsi ini dapat dipanggil beberapa kali.

Contoh

Misalkan terjadi pemanggilan berikut:

```
Alice(5, 6, [4, 0, 3, 1, 4, 2], [2, 2, 0, 2, 0, 3],
      10, [8, 2, 7, 6, 1, 5, 0, 9, 3, 4])
```

Graf yang diberikan adalah sebagai berikut.



Nilai awal p adalah $[8, 2, 7, 6, 1, 5, 0, 9, 3, 4]$.

Dari informasi di atas, dapat dibuktikan bahwa nilai optimal permainan adalah 1.

Misalkan Alice melakukan 4 langkah sebagai berikut:

Argumen t pada `Bob`	Nilai kembalian `Bob`	Indeks p yang terpengaruh	p setelah penukaran Bob
$[3, 1, 5, 2, 0]$	5	$5, 2$	$[8, 2, 5, 6, 1, 7, 0, 9, 3, 4]$
$[9, 3, 7, 2, 1]$	0	$1, 7$	$[8, 9, 5, 6, 1, 7, 0, 2, 3, 4]$
$[5, 6, 7, 8, 9]$	1	$5, 7$	$[8, 9, 5, 6, 1, 2, 0, 7, 3, 4]$
$[7, 5, 2, 3, 6]$	3	$5, 2$	$[8, 9, 2, 6, 1, 5, 0, 7, 3, 4]$

Perhatikan bahwa langkah-langkah Alice dan Bob di atas belum tentu optimal. Langkah-langkah tersebut ditunjukkan semata-mata untuk memperagakan jalannya permainan. Perhatikan juga bahwa Alice bisa saja langsung menghentikan permainan, karena nilai awal permutasi p sudah 1.

Setelah Alice melakukan langkah-langkah tersebut, nilai sesungguhnya dari permutasi p adalah 3 ($p[2] = 2, p[5] = 5, p[7] = 7$).

Akhirnya, fungsi `Alice()` mengembalikan 1, yakni nilai optimal permainan.

Perhatikan bahwa walaupun Alice mendapatkan nilai 3 saat bermain dengan Bob, Anda akan mendapatkan nilai 0 jika nilai kembalian `Alice()` adalah 3 dan bukan 1.

Batasan

- $2 \le m \le 400$
- $m - 1 \le e \le 400$
- $0 \le u[i], v[i] < m$
- $m \le n \le 400$
- $0 \le p[i] < n$
- Graf yang diberikan terhubung dan tidak memiliki *self-loop* maupun *multiple edges*.
- p merupakan suatu permutasi; dengan kata lain, $p[i] \ne p[j]$ untuk $i \ne j$.

Subsoal

1. (6 poin) $m = 2$
2. (6 poin) $e > m$
3. (10 poin) $e = m - 1$
4. (24 poin) $e = m = 3$
5. (24 poin) $e = m = 4$

6. (30 poin) $e = m$

Untuk setiap subsoal, Anda bisa mendapatkan poin parsial. Misalkan r merupakan rasio maksimum dari $\frac{k}{n}$ di antara semua kasus uji dalam suatu subsoal, dimana k merupakan banyaknya putaran (banyaknya pemanggilan `Bob()`). Nilai Anda pada subsoal tersebut akan dikalikan dengan bilangan berikut:

Kondisi	Pengali
$12 \leq r$	0
$3 < r < 12$	$1 - \log_{10}(r - 2)$
$r \leq 3$	1

Khususnya, jika Anda menyelesaikan suatu subsoal dengan paling banyak $3n$ putaran, maka Anda akan mendapatkan poin penuh pada subsoal tersebut. Jika Anda menggunakan lebih dari $12n$ putaran, maka Anda akan mendapatkan nilai 0 untuk subsoal tersebut (yang akan ditunjukkan sebagai `Output isn't correct`).

Contoh *Grader*

Contoh *grader* membaca masukan dengan format berikut:

- Baris 1: $m\ e$
- Baris $2 + i$ $(0 \leq i \leq e - 1)$: $u[i]\ v[i]$
- Baris $2 + e$: n
- Baris $3 + e$: $p[0]\ p[1]\ \ldots\ p[n - 1]$

Contoh *grader* mencetak keluaran dengan format berikut:

- Baris 1: permutasi p pada akhir permainan
- Baris 2: nilai kembalian `Alice()`
- Baris 3: nilai sesungguhnya pada akhir permainan
- Baris 4: banyaknya putaran