Permutation Game

Alice y Bob son amigos de la infancia y les encanta jugar juegos intelectuales. Hoy, están jugando un nuevo juego con grafos.

El juego incluye un grafo **conectado** (conexo) con m vértices, numerados del 0 al $m-1$, y e aristas, numeradas del 0 al $e-1$. La arista i-ésima conecta los vértices $u[i]$ y $v[i]$.

El juego también incluye una permutación $p[0], p[1], \ldots, p[n-1]$ de longitud n, donde $m \leq n$. Una permutación es un arreglo en el que cada número del 0 to $n-1$ aparece exactamente una vez, en algún orden. El puntaje de la permutación p es el número de índices i tales que $p[i] = i$.

El juego durará un máximo de 10^{100} turnos. En cada turno, ocurre lo siguiente:

1. Si Alice decide terminar el juego, este se detiene.
2. De lo contrario, Alice elige **índices distintos** $t[0], t[1], \ldots, t[m-1]$, donde $0 \leq t[i] < n$. Notar que, **no** es necesario que $t[0] < t[1] < \ldots < t[m-1]$.
3. Bob elige un índice $0 \leq j < e$ (una arista del grafo) e intercambia $p[t[u[j]]]$ y $p[t[v[j]]]$.

Alice busca maximizar el puntaje final de la permutación, mientras que Bob intenta minimizarlo.

Tu tarea es ayudar a Alice a jugar contra Bob, cuyos movimientos son simulados por el evaluador.

Definamos el *puntaje óptimo* como el puntaje final que se obtendría si ambos jugadores actuaran de forma óptima durante el juego.

Deberás determinar el puntaje óptimo de la permutación y después jugar el juego con Bob para obtener **al menos** ese puntaje óptimo después de algunos turnos.

Notar que la estrategia de Alice deberá funcionar sin importar que movimientos haga Bob, incluyendo aquellos no óptimos.

Detalles de implementación

Deberás implementar la siguiente función:

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

- m: el número de vértices en el grafo.

- e: el número de aristas en el grafo.
- u y v: arreglos de tamaño e describiendo las aristas en el grafo.
- n: la longitud de la permutación.
- p: un arreglo de tamaño n describiendo la permutación.
- Esta función se llama exactamente una vez.
- Esta función deberá retornar un entero – el puntaje óptimo del juego.

Dentro de la función anterior, podrás llamar a la siguiente función:

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

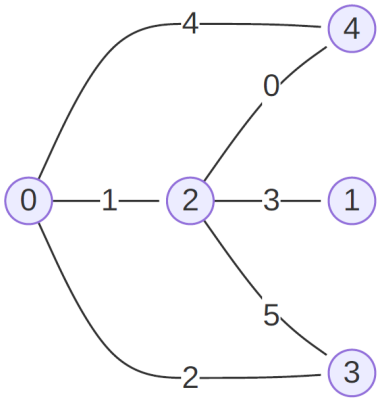
- t: un arreglo de tamaño m, que contiene índices distintos, donde $0 \le t[i] < n$ y $t[i] \ne t[j]$ para cualquiera $i \ne j$.
- Esta función retorna un único entero j que cumple que $0 \le j < e$.
- Esta función puede ser llamada múltiples veces.

Ejemplo

Considera la siguiente llamada:

```
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])
```

El grafo se ve de la siguiente forma:



y p es inicialmente $[8, 2, 7, 6, 1, 5, 0, 9, 3, 4]$.

Dados los límites anteriores, podemos demostrar que el puntaje óptimo es 1.

Supongamos que, Alice hace los siguientes 4 movimientos:

Parámetro de t para `Bob`	Valor de retorno de `Bob`	Índices correspondientes de p	p después del intercambio por 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	$5,2$	$[8,9,2,6,1,5,0,7,3,4]$

Notar que Alice y Bob no necesariamente hacen los movimientos óptimos. Dichos movimientos son mostrados por puro fin demostrativo. También notar que Alice podría terminar el juego inmediatamente, ya que el puntaje inicial de la permutación es 1.

Después de que Alice ha hecho todos los movimientos anteriores, el puntaje de la permutación se vuelve 3 ($p[2] = 2, p[5] = 5, p[7] = 7$).

Finalmente, la función `Alice()` retornará 1 – el puntaje óptimo de la permutación.

Notar que aunque Alice ha logrado un puntaje de 3 jugando con Bob, recibirás 0 puntos si en la función `Alice()` retornas 3 en lugar de 1.

Límites

- $2 \le m \le 400$
- $m - 1 \le e \le 400$
- $0 \le u[i], v[i] < m$
- $m \le n \le 400$
- $0 \le p[i] < n$
- El grafo está conectado, y no contiene aristas de un nodo a si mismo ó aristas múltiples.
- p es una permutación, es decir, $p[i] \ne p[j]$ para cualquier $i \ne j$.

Subtareas

1. (6 puntos) $m = 2$
2. (6 puntos) $e > m$
3. (10 puntos) $e = m - 1$
4. (24 puntos) $e = m = 3$
5. (24 puntos) $e = m = 4$
6. (30 puntos) $e = m$

Para cada subtarea, puedes obtener un puntaje parcial. Sea r el máximo ratio de $\frac{k}{n}$ de entre todos los casos de prueba de una subtarea, donde k es el número de turnos (es decir, llamadas a `Bob()`). Entonces, tu puntaje para esa subtarea es multiplicado por el siguiente número:

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

En particular, si resuelves el problema dentro de $3n$ turnos, obtendrás todos los puntos para esa subtarea. Usar más de $12n$ turnos resultará en obtener 0 para esa subtarea (mostrado como `Output isn't correct`).

Evaluador de ejemplo

El evaluador de ejemplo lee los datos de entrada en el siguiente formato:

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

El evaluador de ejemplo imprime los datos de salida en el siguiente formato:

- línea 1: permutación final de p
- línea 2: valor de retorno de `Alice()`
- línea 3: el puntaje real de la permutación
- línea 4: el número de turnos