Schlüssel

Architekt Tim hat ein neues Escape-Spiel für Einzelspieler erfunden, mit n Räumen (nummeriert von 0 bis $n-1$). Zu Beginn liegt in jedem Raum genau ein Schlüssel. Jeder Schlüssel hat einen bestimmten Typ, gegeben durch eine Nummer aus 0 bis $n-1$ (inklusive). Der Schlüssel in Raum i hat den Typ $r[i]$. Es kann mehrere Schlüssel mit dem gleichen Typ geben; die Werte $r[i]$ sind also nicht zwingend alle unterschiedlich.

Zwischen den Räumen gibt es Gänge, die **in beiden Richtungen** benutzt werden können; insgesamt gibt es m Gänge, nummeriert von 0 bis $m-1$. Der Gang j verbindet zwei unterschiedliche Räume $u[j]$ und $v[j]$. Zwei Räume können durch mehrere Gänge miteinander verbunden sein.

Der einzige Spieler kann nun nach und nach die Schlüssel aufsammeln und sich durch die Gänge zwischen den Räumen bewegen. Der Spieler **benutzt** einen Gang j, wenn er durch diesen Gang von Raum $u[j]$ zu Raum $v[j]$ geht – oder umgekehrt. Der Spieler kann den Gang aber nur benutzen, wenn er einen Schlüssel mit dem zum Raum passenden Typ $c[j]$ hat.

Wenn der Spieler sich in einem Raum x befindet, kann der Spieler gemäss den Spielregeln die folgenden Aktionen ausführen:

- den Schlüssel aufsammeln, der in Raum x liegt und den Typ $r[x]$ hat – falls der Spieler den Schlüssel nicht bereits hat;
- den Gang j benutzen (falls entweder $u[j] = x$ oder $v[j] = x$), vorausgesetzt, der Spieler hat bereits einen Schlüssel vom Typ $c[j]$.

Hat der Spieler einmal einen Schlüssel aufgesammelt, behält er ihn bis zum Ende des Spiels.

Am Beginn des Spiels startet der Spieler ohne Schlüssel, in irgendeinem Raum s. Ein Raum t ist von einem Raum s **erreichbar**, wenn der Spieler in Raum s startet und gemäss der Spielregeln (siehe oben) zu Raum t gelangen kann.

Für jeden Raum i ($0 \leq i \leq n-1$) sei $p[i]$ die Anzahl der Räume, die von Raum i aus erreichbar sind. Tim möchte die Nummern derjenigen Räume wissen, für die diese Anzahl $p[i]$ minimal ist.

Implementierungsdetails

Implementiere die folgende Funktion:

```
int[] find_reachable(int[] r, int[] u, int[] v, int[] c)
```

- r: ein Array der Länge n. Für alle i ($0 \leq i \leq n-1$) hat der Schlüssel in Raum i den Typ $r[i]$.

- u, v: zwei Arrays der Länge m. Für alle j ($0 \le j \le m-1$) verbindet Gang j die Räume $u[j]$ und $v[j]$.
- c: ein Array der Länge m. Für alle j ($0 \le j \le m-1$) wird der Schlüsseltyp $c[j]$ benötigt, um den Gang j zu benutzen.
- Die Prozedur soll ein Array a der Länge n zurückgeben. Für alle $0 \le i \le n-1$ soll der Wert von $a[i]$ dann 1 sein, wenn $p[i]$ minimal ist (also für alle j mit $0 \le j \le n-1$, $p[i] \le p[j]$); ansonsten soll der Wert von $a[i]$ 0 sein.

Beispiele

Beispiel 1

Betrachte den folgenden Aufruf:

```
find_reachable([0, 1, 1, 2],
               [0, 0, 1, 1, 3], [1, 2, 2, 3, 1], [0, 0, 1, 0, 2])
```

Wenn der Spieler in Raum 0 startet, ist diese Folge von Aktionen möglich:

Aktueller Raum	Aktion
0	Nimm den Schlüssel von Typ 0
0	Benutze Gang 0 zu Raum 1
1	Nimm den Schlüssel von Typ 1
1	Benutze Gang 2 zu Raum 2
2	Benutze Gang 2 zu Raum 1
1	Benutze Gang 3 zu Raum 3

Raum 3 ist also von Raum 0 erreichbar. Auf ähnliche Art können wir Aktionsfolgen konstruieren, die zeigen, dass alle Räume von Raum 0 erreichbar sind. Daher ist $p[0] = 4$. Die folgende Tabelle zeigt die erreichbaren Räume für alle Starträume:

Startraum i	Erreichbare Räume	$p[i]$
0	$[0, 1, 2, 3]$	4
1	$[1, 2]$	2
2	$[1, 2]$	2
3	$[1, 2, 3]$	3

Der minimale Wert von $p[i]$ ist 2. Dieser Wert wird für die Räume $i = 1$ und $i = 2$ erreicht. Daher soll die Funktion $[0, 1, 1, 0]$ zurückgeben.

Beispiel 2

```
find_reachable([0, 1, 1, 2, 2, 1, 2],
               [0, 0, 1, 1, 2, 3, 3, 4, 4, 5],
               [1, 2, 2, 3, 3, 4, 5, 5, 6, 6],
               [0, 0, 1, 0, 0, 1, 2, 0, 2, 1])
```

Die folgende Tabelle zeigt die erreichbaren Räume:

Startraum i	Erreichbare Räume	$p[i]$
0	$[0, 1, 2, 3, 4, 5, 6]$	7
1	$[1, 2]$	2
2	$[1, 2]$	2
3	$[3, 4, 5, 6]$	4
4	$[4, 6]$	2
5	$[3, 4, 5, 6]$	4
6	$[4, 6]$	2

Der minimale Wert von $p[i]$ ist 2. Dieser wird für $i \in \{1, 2, 4, 6\}$ erreicht. Daher soll die Funktion $[0, 1, 1, 0, 1, 0, 1]$ zurückgeben.

Beispiel 3

```
find_reachable([0, 0, 0], [0], [1], [0])
```

Die folgende Tabelle zeigt die erreichbaren Räume:

Startraum i	Erreichbare Räume	$p[i]$
0	$[0, 1]$	2
1	$[0, 1]$	2
2	$[2]$	1

Der minimale Wert von $p[i]$ ist 1 und wird für $i = 2$ erreicht. Daher soll die Funktion $[0, 0, 1]$ zurückgeben.

Beschränkungen

- $2 \leq n \leq 300\,000$
- $1 \leq m \leq 300\,000$

- $0 \leq r[i] \leq n-1$ für alle $0 \leq i \leq n-1$
- $0 \leq u[j], v[j] \leq n-1$ und $u[j] \neq v[j]$ für alle $0 \leq j \leq m-1$
- $0 \leq c[j] \leq n-1$ für alle $0 \leq j \leq m-1$

Teilaufgaben

1. (9 Punkte) $c[j] = 0$ für alle $0 \leq j \leq m-1$ und $n, m \leq 200$
2. (11 Punkte) $n, m \leq 200$
3. (17 Punkte) $n, m \leq 2000$
4. (30 Punkte) $c[j] \leq 29$ (für alle $0 \leq j \leq m-1$) und $r[i] \leq 29$ (für alle $0 \leq i \leq n-1$)
5. (33 Punkte) Keine weiteren Beschränkungen.

Beispielgrader

Der Beispielgrader liest die Eingabe im folgenden Format:

- Zeile 1: n m
- Zeile 2: $r[0]$ $r[1]$ $\ldots$ $r[n-1]$
- Zeile $3+j$ ($0 \leq j \leq m-1$): $u[j]$ $v[j]$ $c[j]$

Der Beispielgrader gibt den Rückgabewert von `find_reachable` im folgenden Format aus:

- Zeile 1: $a[0]$ $a[1]$ $\ldots$ $a[n-1]$