Political Patricians (patricians)

Un censore romano sta monitorando N patrizi di dubbia etica politica. Ogni patrizio appartiene a **esattamente una** fazione politica segreta, ognuna delle quali conta uno o più patrizi. Il censore vuole determinare quali patrizi appartengono a quali fazioni.

Per farlo, il censore vuole organizzare ripetuti banchetti ai quali tutti gli N patrizi sono invitati. A ogni banchetto, i patrizi possono essere seduti a tavoli diversi e il censore controlla la disposizione dei posti a sedere. Un patrizio è *solitario* se durante il banchetto nessun altro patrizio della stessa fazione politica siede allo stesso tavolo.

Nulla sfugge all'occhio del censore, che dopo ogni banchetto sa esattamente quali patrizi erano solitari. Il tuo compito è aiutare il censore a determinare le fazioni politiche dei patrizi senza organizzare troppi banchetti.

Implementazione

Dovrai inviare un singolo file sorgente `.cpp`.

⚠ Tra gli allegati di questo problema troverai un template `patricians.cpp` con un'implementazione di esempio.

Dovrai implementare la seguente funzione:

C++
```cpp
vector<int> find_factions(int N)
```

- L'intero N rappresenta il numero di patrizi, i quali sono numerati da 0 a $N-1$.
- La funzione deve restituire un array P di N interi che rappresentano le fazioni politiche dei patrizi.

$P[i]$ deve essere uguale a $P[j]$ se e solo se i patrizi i e j appartengono alla stessa fazione politica. Non ci sono ulteriori vincoli sui valori degli interi di P.

Il tuo programma può chiamare la seguente funzione, già definita nel grader:

C++
```cpp
vector<bool> organize_banquet(const vector<int>& Q)
```

- L'array Q di N interi rappresenta la disposizione dei posti a sedere: il patrizio i sarà seduto al tavolo $Q[i]$.

Non ci sono vincoli sui valori degli interi di Q: ai tavoli può essere assegnato qualsiasi numero, l'unica informazione rilevante è che se $Q[i] = Q[j]$, allora il patrizio i e il patrizio j saranno seduti allo stesso tavolo (e sono seduti a tavoli diversi altrimenti).

- La funzione restituisce un array A di N booleani. $A[i]$ è vero se e solo se il patrizio i è solitario

al banchetto.

Il grader **non è adattivo** e ogni submission viene valutata sullo stesso insieme di casi di test. In altre parole, le fazioni politiche di ciascun patrizio sono fissate in anticipo in ogni caso di test e non dipendono dalle chiamate a `organize_banquet` effettuate dal tuo programma.

Grader di prova

Tra gli allegati di questo problema troverai una versione semplificata del grader utilizzato durante la valutazione, che puoi usare per testare le tue soluzioni in locale. Il grader di esempio legge i dati da `stdin`, chiama la funzione `find_factions` e scrive su `stdout` usando il seguente formato.

Il file di input è formato da 2 righe, contenenti:

- Riga 1: l'intero N.
- Riga 2: N interi che descrivono la fazione politica di ciascun patrizio.

L'output è costituito da una sola riga, contenente il punteggio ottenuto dalla tua soluzione nel caso di test. Se il punteggio è 0, stampa un breve messaggio su `stderr` con la ragione (risposta errata (cioè fazioni errate), query non valide o troppe query).

Assunzioni

- $4 \leq N \leq 2000$.

Puoi chiamare `organize_banquet` un massimo di 30000 volte.

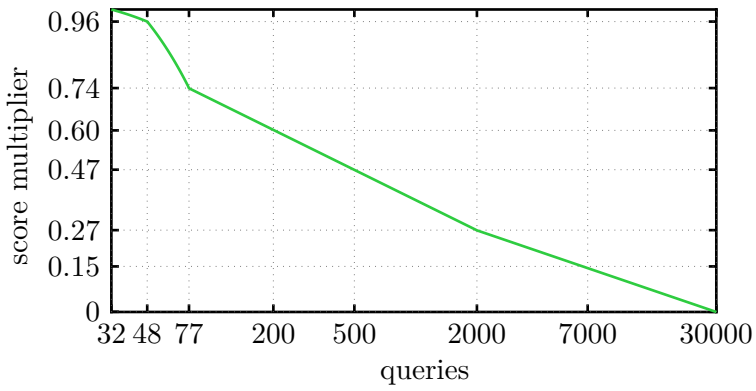
Assegnazione del punteggio

Il tuo programma verrà testato su diversi casi di test raggruppati in subtask. Il moltiplicatore di punteggio associato a un subtask sarà il minimo dei moltiplicatori di punteggio ottenuti in ciascuno dei casi di test.

- **Subtask 0** [**0 punti**]: Casi di esempio.
- **Subtask 1** [**19 punti**]: Ci sono esattamente due fazioni politiche.
- **Subtask 2** [**29 punti**]: Ogni fazione politica conta esattamente due patrizi.
- **Subtask 3** [**52 punti**]: Nessun vincolo aggiuntivo.

Il moltiplicatore di punteggio ottenuto in un caso di test è determinato come segue.

Numero di query	Moltiplicatore di punteggio
≤ 32	1.0
$33 \leq x < 48$	$1.08 - 0.0025x$
48	0.96
$48 < x < 77$	$1.32413 - 0.00758x$
77	0.74
$77 < x < 2000$	$1.366 - 0.1 \log_2(x)$
2000	0.27
$2000 < x \leq 30000$	$1.027 - 0.0691 \log_2(x)$



Il punteggio ottenuto su un subtask è il moltiplicatore di punteggio minimo ottenuto sui suoi casi di test, moltiplicato per il punteggio base del subtask.

Esempi di input/output

Grader	Soluzione
`find_factions(5)`	
	`organize_banquet([0,0,1,1,2])`
`return [1,1,1,1,1]`	
	`organize_banquet([0,0,0,1,1])`
`return [0,1,0,1,1]`	
	`organize_banquet([0,1,2,1,1])`
`return [1,0,1,0,1]`	
	`return [0,1,0,1,2]`

Grader	Soluzione
`find_factions(4)`	
	`organize_banquet([1,1,0,0])`
`return [1,1,1,1]`	
	`organize_banquet([1,1,1,2])`
`return [0,1,0,1]`	
	`organize_banquet([0,1,1,1])`
`return [1,0,1,0]`	
	`return [0,1,0,1]`

Spiegazione

Nel primo esempio, i patrizi 0 e 2 appartengono a una fazione politica, i patrizi 1 e 3 a una diversa, e il patrizio 4 alla sua fazione. Nel primo banchetto, tutti i patrizi sono solitari. Nel secondo banchetto, i patrizi 0 e 2 sono seduti assieme e quindi non solitari, così come i patrizi 1 e 3 nel terzo banchetto.