



## Problema Speedrun

Header C++      `speedrun.h`

Marcel si sta allenando duramente per le speedruns, spera di ottenere un nuovo record mondiale (WR). Sfortunatamente, dopo svariati tentativi, non è riuscito nell'intento così ha deciso di imbrogliare!

La speedrun a cui sta giocando si chiama *Tree Souls III* e Marcel spera di ottenere un nuovo WR nella categoria *full-tree*.

Il gioco, come il nome suggerisce, si svolge in un albero: un grafo connesso di  $N$  vertici e  $N - 1$  archi.

Il gioco funziona nel seguente modo: all'inizio il giocatore si trova in un nodo qualsiasi dell'albero. Il giocatore può scegliere un nodo  $x$ : se esiste un arco tra il nodo in cui si trova e il nodo  $x$ , allora il giocatore si sposta nel nodo  $x$ , altrimenti non succede nulla. Il gioco **termina** quando Marcel visita ogni nodo dell'albero.

Marcel per imbrogliare ha deciso di sfruttare un bug del gioco che gli consente di impostare il seed con cui viene generato l'albero. Tuttavia, se si imparasse a memoria l'albero, sarebbe ovvio a tutti il tentativo di imbrogliare, così ha deciso segnarsi solo alcune informazioni per ogni nodo.

Le informazioni vengono codificate attraverso una stringa binaria di lunghezza  $l$ , dove  $l$  è un valore uguale per ogni nodo. Quando Marcel si trova in un nodo, può leggere la stringa binaria relativa al nodo.

## Protocollo di interazione

**Questo è un task di tipo communication!**

Devi implementare le seguenti funzioni:

```
void assignHints(int subtask, int N, int A[], int B[]);  
void speedrun(int subtask, int N, int start);
```

Durante l'esecuzione potrai usare le seguenti funzioni già implementate:

```
void setHintLen(int l);  
void setHint(int i, int j, bool b);  
int getLength();  
bool getHint(int j);  
bool goTo(int x);
```

Il tuo codice verrà eseguito **due volte**.

Nella **prima fase** il grader chiamerà la funzione `assignHints` esattamente una volta con i seguenti parametri:

- l'intero `subtask`: l'indice del subtask.
- l'intero `N`: il numero di nodi dell'albero.
- gli array `A` e `B` indicizzati da 1 a  $N - 1$ : per ogni  $1 \leq i \leq N - 1$  è presente un arco tra `A[i]` e `B[i]`.

Dovrai successivamente chiamare la funzione `setHintLen` dandogli come unico parametro la lunghezza delle informazione da salvare in ogni nodo.

Infine potrai chiamare la funzione `setHint` più volte con i seguenti parametri:

- l'intero  $i$ : l'indice del nodo.
- l'intero  $j$ : l'indice del stringa binaria.
- il booleano  $b$ : il valore da assegnare al  $j$ -esimo valore della stringa binaria del nodo  $i$ .

Nota che la stringa è indicizzata da 1 a  $l$ , dove  $l$  è il valore fornito alla funzione `setHintLen`.

In questa fase **non** devi chiamare le funzioni `getLength`, `getHint` e `goTo`.



Nella **seconda fase** il grader chiamerà la funzione **speedrun** esattamente una volta con i seguenti parametri:

- l'intero **subtask**: l'indice del subtask.
- l'intero **N**: il numero di nodi dell'albero.
- l'intero **start**: il nodo da cui Marcel parte.

Potrai usare le seguenti funzioni:

- **getLength()**: restituisce il valore  $l$  fornito a **setHintLen** nella precedente fase.
- **getHint(j)**: restituisce il bit della stringa di informazioni del nodo in cui Marcel si trova.
- **goto(x)**: se il valore  $x$  è un nodo adiacente al nodo attuale, allora Marcel si sposta nel nodo  $x$  e la funzione restituisce **true**, altrimenti la funzione restituisce **false**.

In questa fase **non** devi chiamare le funzioni **setHintLen** e **setHint**.

## Assunzioni

- $1 \leq N \leq 1000$
- Sia  $Q$  il numero di chiamate a **goTo** che hanno restituito **false**. Seguono i limiti su  $l$  e  $Q$  da rispettare per ottenere il punteggio di ogni subtask.

### Subtask 1 (21 punti)

- $l \leq N$
- $Q \leq 2000$
- Il punteggio di questo subtask è 21 se risolvi correttamente tutti i testcase che ne fanno parte, 0 altrimenti.

### Subtask 2 (8 punti)

- $l \leq 20$
- $Q \leq 2000$
- L'albero è una stella, ovvero esiste un nodo  $x$  ( $1 \leq x \leq N$ ) che è connesso a tutti gli altri.
- Il punteggio di questo subtask è 8 se risolvi correttamente tutti i testcase che ne fanno parte, 0 altrimenti.

### Subtask 3 (19 punti)

- $l \leq 20$
- $Q \leq 2000$
- Ogni nodo ha grado al più 2.
- Il punteggio di questo subtask è 19 se risolvi correttamente tutti i testcase che ne fanno parte, 0 altrimenti.

### Subtask 4 (12 punti)

- $l \leq 316$
- $Q \leq 32000$
- Il punteggio di questo subtask è 12 se risolvi correttamente tutti i testcase che ne fanno parte, 0 altrimenti.

### Subtask 5 (40 punti)

- $Q \leq 2000$
- Il punteggio  $s$  di ogni testcase è calcolato come segue. Se  $l > 40$ , allora  $s = 0$ . Se  $l \leq 20$ , allora  $s = 40$ . Altrimenti,  $s = 60 - l$ . Ovvero se  $l = 40$ , riceverai metà del punteggio massimo del testcase, e se  $l \leq 20$ , riceverai l'intero punteggio. Il punteggio dell'intero subtask è il minimo dei punteggi  $s$  ottenuti nei singoli testcase.



## Esempio di interazione

Nella **prima fase** (in cui puoi salvare informazioni nei nodi), la tua funzione `assignHints` verrà chiamata come segue:

```
assignHints(  
  /* subtask = */ 1,  
  /* N        = */ 5,  
  /* A        = */ {-, 1, 2, 3, 3},  
  /* B        = */ {-, 2, 3, 4, 5});
```

Questa funzione potrebbe, ad esempio, scegliere di impostare  $l = 2$ :

```
setHintLen(/* l = */ 2);
```

E di lasciare tutti i bit salvati a 0, ad eccezione del bit 1 del nodo 2, che imposta a 1:

```
setHint(  
  /* i = */ 2,  
  /* j = */ 1,  
  /* b = */ 1);
```

Infine, termina. La prima istanza del tuo programma a questo punto viene terminata, perdendo il valore di tutte le variabili nella sua memoria. A questo punto, le informazioni salvate sono "00" per tutti i nodi tranne il nodo 2, che contiene la stringa "10".

Nella **seconda fase** (fase di speedrun), la tua funzione `speedrun` viene chiamata:

```
speedrun(  
  /* subtask = */ 1,  
  /* N        = */ 5,  
  /* start    = */ 1);
```

La funzione inizia la speedrun sull'albero effettuando, ad esempio, le seguenti chiamate:

```
getLength(); // = 2  
getHint(1);  // = 0  
getHint(2);  // = 0  
goTo(2);     // = true  
getHint(1);  // = 1  
getHint(2);  // = 0
```

Le chiamate possono essere interpretate in questo modo: all'inizio, ti trovi nel nodo 1, ottieni il valore di  $l$  ( $l = 2$ ) e le informazioni salvate sul nodo 1 ("00"). Poi ti sposti con successo al nodo 2 e ottieni la sua stringa, "10". L'esecuzione continua come segue:

```
goTo(2);      // = false (non puoi spostarti dal nodo 2 a se stesso)  
goTo(5);      // = false (non esiste un arco tra il nodo 2 e il 5)  
goTo(3);      // = true  
goTo(4);      // = true  
goTo(3);      // = true  
goTo(5);      // = true
```

A questo punto hai visitato tutti i nodi, quindi la funzione `speedrun` può terminare. Al termine dell'esecuzione,  $Q = 2$ , perchè 2 delle chiamate a `goTo` hanno restituito `false`.