



Aufgabe Speedrun

C++ header `speedrun.h`

Marcel hat kürzlich angefangen, sich für Speedruns zu interessieren, und hat sich als Ziel gesetzt, einen neuen WR (Weltrekord) aufzustellen. Leider schafft er es nicht, die neu entdeckten Kniffe zu meistern, deshalb versucht er, sich über Cheaten einen unfairen Vorteil zu verschaffen.

Das Game, das ihn interessiert, nennt sich *Tree Souls III*. Er möchte einen WR in der Kategorie *full-tree%* aufstellen. Wie der Name schon impliziert, findet das Game auf einem Baum statt. (Ein Baum ist ein Graph von N Knoten, nummeriert von 1 bis N , der genau $N - 1$ Kanten hat und in dem es keine Zyklen gibt.)

Das Game funktioniert wie folgt: Der Spielercharakter wird an einem beliebigen Knoten innerhalb des Baums platziert. Er kann eine Ganzzahl x auswählen und versuchen, vom Knoten, in dem er sich momentan befindet, zum Knoten x zu laufen. Gibt es tatsächlich eine Kante zwischen x und dem aktuellen Knoten, geht er nach x , ansonsten passiert nichts. Der Speedrun ist gültig, wenn er jeden Knoten mindestens einmal besucht hat.

Nun zum Cheaten... Weil er den Edge-Teleportation-Glitch nicht beherrscht, setzt er einen fixen Seed für die Welt, in der er seinen Speedrun ausführen möchte. Weil es sofort auffliegen würde, wenn er sich den ganzen Tree merkt, möchte er stattdessen kleine Hinweise in den Knoten verstecken. Ein Hinweis ist ein Bitstring der Länge ℓ (das gleiche ℓ gilt für alle Knoten). Den Hinweis eines Knotens kann er nur anschauen, wenn er sich auf diesem Knoten befindet.

Interaktives Protokoll

Dies ist eine Encoding-Decoding-Aufgabe!

Dein Code wird zweimal ausgeführt, einmal für das Encoden (die *Hinweis-Setzen-Phase*), und ein zweites Mal für das Decoden (die *Speedrun-Phase*).

Du sollst folgende Funktionen implementieren (aber dafür keine `main`-Funktion).

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

Und dabei kannst du folgende Funktionen aufrufen:

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

Hinweise-Setzen-Phase. Dies ist die erste Ausführung deines Codes. In dieser Phase wird der Grader die Funktion `assignHints` genau einmal aufrufen. Du bekommst *subtask*, den Index der ausgeführten Teilaufgabe; N , die Grösse des Baums; und die Kanten in zwei Arrays A und B wie folgt: für jedes $i = 1, \dots, N - 1$, gibt es eine Kante zwischen $A[i]$ und $B[i]$ (beachte, dass die Indizes 1-basiert sind). Anschliessend sollst du `setHintLen` genau einmal aufrufen, mit der Länge der Hinweise, für den du dich als Parameter entschieden hast. Dannach kannst du die Funktion `setHint(i, j, b)` mehrere Male aufrufen, um das j -te Bit im Hinweis von Knoten i auf b zu setzen. Die Bits sind nummeriert von 1 bis ℓ . Die Hinweise sind anfänglich gefüllt mit Nullen. In dieser Phase ist es nicht erlaubt, `getLength`, `getHint` oder `goTo` aufzurufen. Nachdem du alle Hinweise gesetzt hast, soll sich `assignHints` beenden.

Speedrun-Phase. Dies ist die zweite Ausführung deines Codes. In dieser Phase wird der Grader die Funktion `speedrun(subtask, N, start)` genau einmal ausführen, wobei es dir den Startknoten `start`,



in welchem der Spielercharakter startet, sowie den Wert von N mitteilt. Ebenfalls erhältst du den Index der Teilaufgabe über den Parameter *subtask*. Danach kannst du die Funktion `getLength()` aufrufen, welche dir ℓ , zurückgibt, der Länge der Hinweise, welche von deinem Programm in der ersten Phase gesetzt wurden; mit der Funktion `getHint(j)` findest du heraus, was der Wert des j -ten Bits des Hinweises ist, vom Knoten, an dem du dich momentan befindest; und dann gibt es noch die Funktion `goTo` um dich zu bewegen. Der Parameter der Funktion `goTo` ist der Index des Knotens, zu dem du dich bewegen möchtest; gibt es eine Kante zwischen deiner momentan Position und diesem Knoten, dann gibt die Funktion `true` zurück und bewegt dich zu dem Knoten. Ansonsten gibt sie `false` zurück und du bleibst du deinem momentanen Knoten stehen. In dieser Phase darfst du weder `setHint` noch `setHintLen` aufrufen. Die Funktion `speedrun` sollte sich irgendwann beenden, nachdem du alle Knoten des Baums besucht hast.

Limits

- $1 \leq N \leq 1000$
- Sei Q die Anzahl Aufrufe von `goTo`, welche `false` zurückgegeben haben. Was nun folgt sind die Bedingungen an ℓ und Q , die eingehalten werden sollten, um Punkte für die jeweiligen Teilaufgaben zu erhalten.

Teilaufgabe 1 (21 Punkte)

- $\ell \leq N$
- $Q \leq 2000$
- Du erhältst 21 Punkte, falls du alle Testfälle korrekt löst, ansonsten 0 Punkte.

Teilaufgabe 2 (8 Punkte)

- $\ell \leq 20$
- $Q \leq 2000$
- Der Baum ist ein Stern; d.h. es gibt einen Knoten x ($1 \leq x \leq N$) welcher mit jedem anderen Knoten verbunden ist.
- Du erhältst 8 Punkte, falls du alle Testfälle korrekt löst, ansonsten 0 Punkte.

Teilaufgabe 3 (19 Punkte)

- $\ell \leq 20$
- $Q \leq 2000$
- Jeder Knoten hat einen Grad von höchstens 2.
- Du erhältst 19 Punkte, falls du alle Testfälle korrekt löst, ansonsten 0 Punkte.

Teilaufgabe 4 (12 Punkte)

- $\ell \leq 316$
- $Q \leq 32000$
- Du erhältst 12 Punkte, falls du alle Testfälle korrekt löst, ansonsten 0 Punkte.

Teilaufgabe 5 (40 Punkte)

- $Q \leq 2000$
- Dein Score s von jedem Test wird wie folgt berechnet. Falls $\ell > 40$, dann ist $s = 0$. Falls $\ell \leq 20$, dann ist $s = 40$. Ansonsten ist $s = 60 - \ell$. Zum Beispiel, falls $\ell = 40$, dann erhältst du die Hälfte der Punkte für den Teste, und falls $\ell \leq 20$ erhältst du die volle Punktzahl. die Punktzahl für die Teilaufgabe ist das Minimum der Werte von s aller Tests in dieser Teilaufgabe.

Beispiel-Interaktion

In der ersten Ausführung (Hinweis-Setz-Phase) wird deine Funktion wie folgt aufgerufen:



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

Die Funktion könnte sich in der Folge dazu entscheiden, $\ell = 2$ zu setzen.

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

Dann möchte die Funktion alle Bits der Hinweise auf 0 belassen, bis auf eines: das Bit 1 von Knoten 2 soll auf 1 gesetzt werden:

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

Anschliessend beendet sich die Funktion. Die erste Instanz deines Programms wird nun beendet, und entsprechend gehen alle Daten, welche dein Programm im Speicher gespeichert hat, verloren.

Alles, was bleibt, ist dass die Hints in jedem Knoten "00" sind, bis auf Knoten 2, welcher den Hinweis "10" hat.

In der zweiten Ausführung (Speedrun-Phase), wird deine Funktion wie folgt aufgerufen:

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

Darauf beginnt die Funktion den Speedrun im Baum.

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

Zu diesem Zeitpunkt warst du ursprünglich in Knoten 1, hast du herausgefunden, dass $\ell = 2$ gilt, dass der Hinweis von Knoten 1 "00" ist, und du hast dich erfolgreich von Knoten 1 nach Knoten 2 bewegt und herausgefunden, dass der Hinweis von Knoten 2 der Bitstring "10" ist. Die Ausführung könnte sich wie folgt fortsetzen.

```
goTo(2);      // = false (du kannst nicht von Knoten 2 zurück zu Knoten 2 gehen)  
goTo(5);      // = false (es gibt keine Kante von Knoten 2 zu Knoten 5)  
goTo(3);      // = true  
goTo(4);      // = true  
goTo(3);      // = true  
goTo(5);      // = true
```

Nun wurden alle Knoten einmal besucht, und die Funktion `speedrun` darf sich beenden. Der Wert von Q ist 2, weil 2 Aufrufe von `goTo` den Wert `false` zurückgegeben haben.