Adatszerkezetek és algoritmusok 2. kis házi feladat

A következő két feladat csak a kerettörténetében hasonló, egyébként egymástól függetlenek. A házi feladat teljesítéséhez mindkettőt meg kell oldani.

A feladatok során a bemenetek nem léphetnek túl a limiteken, ezt nem kell külön ellenőrizni.

1

A doktoranduszi szobában felállított terepasztalon összekeveredtek az állomásra beérkező vagonok. Az állomáson van egy segédvágány, amely kétféleképp tud funkcionálni: veremként és sorként; de mindig csak az egyik. A feladatod, hogy eldöntsd: adott vagonsorrendet sorba lehet-e rendezni a kimeneti vágányra?

A bemeneten N vagon áll felcímkézve: az i-ik vagon címkéje $L_i \in \{1, \ldots, 4\}$, $i = 1 \ldots N$. Három lehetséges lépést lehet végrehajtani: a bemeneti vágányról át lehet tolni egy vagont a kimeneti vágányra vagy a segédvágányra, valamint a segédvágányról a kimeneti vágányra. A segédvágány lehet verem vagy sor típusú: azaz vagy csak a legutoljára, vagy csak a legrégebben rátolt vagont lehet róla letolni. Az első lépésben az $i = 1$ vagont lehet mozgatni, utolsónak pedig az $i = N$ vagont lehet letolni a bemeneti vágányról (nyilván eközben, illetve ezután a segédvágányról vagy segédvágányra is lehet mozgatni).

Jelölje az összes tologatás utáni állapotot a kimeneti vágányon L'_i, $(i = 1 \ldots N)$, ahol az $i = 1$ vagon az elsőnek, míg az $i = N$ az utolsónak rátolt vagon. A feladat annak eldöntése, hogy létezik-e olyan lépéssorozat, amelynek eredményeképpen a kimeneti vágányon monoton növekvő sorrendben állnak a vagonok: $i < j \implies L'_i \leq L'_j$?

Példa

Bemenet: `{1, 1, 4, 2, 3, 2}`. Ez sorbarendezhető veremmel (például áttolva a 4 és 3 címkéjű vagonokat a segédvágányra, a többit rögtön a kimeneti vágányra tolva), de nem rendezhető sorral.

Limitek

- $1 \leq N \leq 100$
- Időlimit: az összes tesztesetre tesztesetenként 0.01 másodperc
- Memórialimit: 100 MiB

API

A feladat megoldásához implementáld a következő függvényt:

```
bool is_orderable(const std::vector<short> &cars, bool stack, bool queue);
```

2

A feladat egy ternális fa felépítése. A fának minden node-ja vagy egy számértéket tartalmazó levél, vagy pontosan három gyereke van (és ekkor nem tartalmaz értéket). (Nem feltétlen kiegyensúlyozott és nem feltétlen keresőfa.)

A bemenet egy szöveg a következő formátumban: a levelek egy-egy egyjegyű szám, a belső node-ok pedig a részfák pontosvesszővel elválasztva: `(x;y;z)`, ahol `x`, `y` és `z` rendre a bal, középső, illetve jobb gyerek.

A kész fán a `get_value` metódus segítségével le lehet kérni az egyes levelek értékeit. A metódus egyetlen paramétere egy útvonal a gyökérelemtől indulva a levélig: az útvonal i-ik eleme meghatározza, hogy az i-ik magasságban levő node melyik gyereke felé kell továbbhaladni. Ha az útvonal nem levelet határoz meg (túl hosszú vagy túl rövid), akkor a metódus `TreeException` kivételt dob. A megadott útvonal minden eleme 0, 1 vagy 2, ezt nem kell külön ellenőrizni.

Példa

Fa bemenet: `((1;(2;3;4);5);(6;7;8);(9;0;1))`; ez az 1. ábrán látható fát eredményezi.

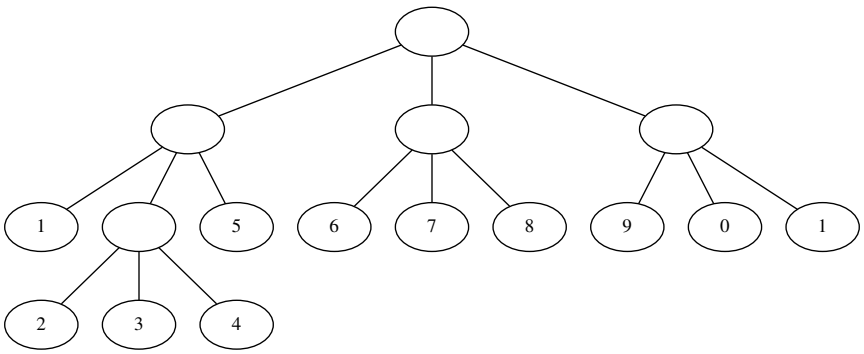

Figure 1: Példa fa a második feladathoz.

Példa lekérdezés: a `{0, 2}` útvonal az 5 értékű levelet kérdezi le, míg a `{0, 1, 1}` a 3 értékűt. A `{2}` és az `{1, 2, 1}` érvénytelen útvonalak.

Limitek

- A fa csúcsainak száma legfeljebb 100000
- Időlimit: az összes tesztesetre
  - a fa felépítésére 0.1 másodperc
  - a kérésekre kérésenként 0.001 másodperc, kivéve:
  - az utolsó teszteseten kérésenként 0.1 másodperc
- Memórialimit:
  - Összes memória: 100 MiB
  - Stack méret: 2 MiB

API

A feladat megoldásához implementáld a következő osztályt:

```
class TernaryTree {
public:
  TernaryTree(const std::string &input);
```

```
  int get_value(const std::vector<short> &path) const;
};
```

A feladatban megadott kivétel származzon a `std::exception` osztályból:

```
struct TreeException : public std::exception {};
```