Labirintus

Olivér kedvenc számítógépes játékában egy labirintusból kell kijutni, több különböző nehézségi szinten. A labirintus felépítése mindegyik szinten azonos. A labirintus leírható egy R sorból és C oszlopból álló táblázattal. A sorokat 0-tól $R-1$-ig, az oszlopokat 0-tól $C-1$-ig sorszámozzuk. Az u-adik sor v-edik mezőjét az (u, v) számpárral azonosítjuk ($0 \leq u < R$, $0 \leq v < C$). A labirintus minden mezője vagy üres, vagy falat tartalmaz.

A játékban összesen Q szint található. Egy-egy szinten a labirintust N darab őr felügyeli (N lehet szintenként különböző), melyeket 0-tól $N-1$-ig számozunk. Közülük az i-edik őr az (U_i, V_i) mezőn található, ahol (U_i, V_i) mező eredetileg üres. A labirintus egy mezőjét egy adott szinten *szabadon átjárhatónak* nevezzük, ha se falat, se őrt nem tartalmaz.

Olivér minden szinten a $(0, 0)$ mezőről indul és célja elérni az $(R-1, C-1)$ mezőt úgy, hogy minden lépésben a négy szomszédos mező valamelyikére léphet, feltéve, hogy az része labirintusnak és szabadon átjárható. Formálisan megfogalmazva, egy lépésben az (u, v) mezőről ($0 \leq u < R$, $0 \leq v < C$) pontosan akkor léphet át az (s, t) mezőre, ha $0 \leq s < R$ és $0 \leq t < C$ teljesül, továbbá $|s-u| + |t-v| = 1$ és az (s, t) mező szabadon átjárható.

A $(0, 0)$ és $(R-1, C-1)$ mezők nem tartalmaznak falat. Ha létezik lépéseknek olyan sorozata, amivel el lehet jutni a $(0, 0)$ mezőről az $(R-1, C-1)$ mezőre, akkor azt mondjuk, hogy Olivér *ki tud szabadulni* a labirintusból. A labirintus őrök nélkül olyan, hogy ki lehet belőle szabadulni. Egyik szint sem tartalmaz olyan őrt, aki a $(0, 0)$ vagy $(R-1, C-1)$ mezők valamelyikét foglalja el.

Sajnos a játék fejlesztői egyes szinteket hibásan valósítottak meg, így bizonyos szinteken nem lehetséges kiszabadulni. Írj programot, ami egyesével meghatározza az egyes szintekről, hogy ki tud-e szabadulni Olivér!

Implementáció

A következő eljárásokat kell megvalósítanod:

```
void init_labyrinth(int R, int C, std::vector<std::vector<int>> L)
```

- R: a labirintus sorainak a száma.
- C: a labirintus oszlopainak a száma.
- L: egy R elemű tömb, melynek minden eleme egy-egy C elemű tömb. $L_{u,v}$ értéke 0, ha az (u, v) mező üres, illetve 1, ha falat tartalmaz.
- Ez az eljárás a labirintus felépítését adja meg. Az értékelés során pontosan egyszer kerül meghívásra, minden más eljárás hívását megelőzve.

```
bool can_escape(int N, std::vector<int> U, std::vector<int> V)
```

- N: az őrök száma.
- U, V: N elemű tömbök, melyek az őrök által elfoglalt mezőket írják le.
- Ez az eljárás egy szint leírását adja meg. Egy logikai értékkel kell visszatérnie, mely pontosan akkor igaz, ha ki lehet szabadulni a labirintusból.
- Ez az eljárás Q alkalommal kerül meghívásra.

A megoldásodnak tartalmaznia kell az `#include "labirintus.h"` importáló sort!

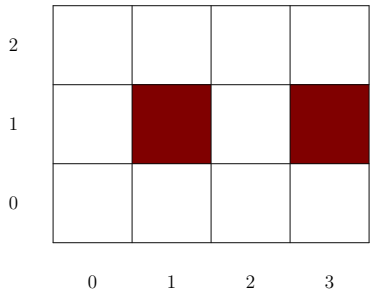
Ha a megoldásod a `standard bemenet`ről olvas, a `standard kimenet`re ír, megszakítja a program végrehajtását, `main` függvényt tartalmaz, vagy más módon eltér a specifikációban foglaltaktól, az rendszerint `Protocol Violation` visszajelzést, vagy fordítási hibát eredményez az értékelő rendszerben.

Példa

Tekintsük a következő függvényhívást:

```
init_labyrinth(3, 4, {{0, 0, 0, 0}, {0, 1, 0, 1}, {0, 0, 0, 0}})
```

Ez az alábbi labirintust írja le:



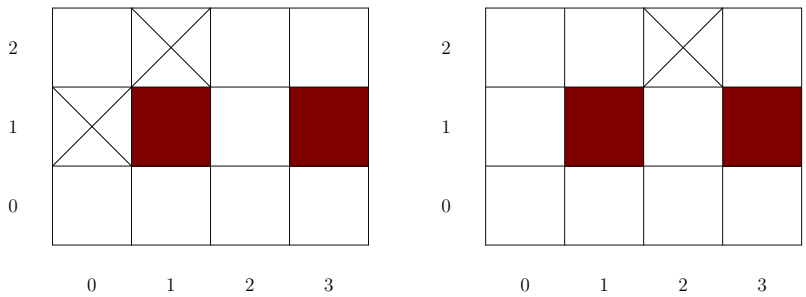
Most tekintsük a következő hívásokat:

```
can_escape(2, {1, 2}, {0, 1})
```

Ekkor az `(1, 0)` és `(2, 1)` mezőkön áll őr, így lehetséges a szabadulás, tehát ennek a hívásnak a `true` igazságértéket kell visszaadnia.

```
can_escape(1, {2}, {2})
```

Ekkor a `(2, 2)` mezőn áll őr, így nem lehetséges a szabadulás, tehát ennek a hívásnak a `false` igazságértéket kell visszaadnia.



Gyakorlás

A letölthető `minta.zip` tartalmazza a `labirintus.cpp` állományt és egy tesztelésre alkalmas `main` programot. A projekt mappába másold be, majd add hozzá a projekthez a fejlesztői környezetedben a `labirintus.h`, `labirintus.cpp` és `main.cpp` állományokat.

A megvalósítandó eljárásokat teljes egészében a `labirintus.cpp` állományban implementáld! A programod által használt saját függvényeket, egyéb programelemeket is ennek kell tartalmaznia! A szükséges standard könyvtárakat is ebben importáld és a használni kívánt direktívákat is itt add meg (pl. `using namespace std;`)! Az értékelő rendszerben csak a `labirintus.cpp` állományt kell beadnod!

Ez a tesztelő program a futása során a `standard bemenetről` olvassa be az adatokat. Az első sor az `R`, `C` és `Q` értékeket tartalmazza. A következő `R` sor tartalmazza a labirintust leíró `L` tömb számjegyeit, elválasztó karakterek nélkül. Ezt `Q` szint leírása követi. Minden szint leírásának első sora az őrök `N` számát tartalmazza. A következő `N` sor soronként két egész értéket tartalmaz, az egyes őrök pozícióját leíró U_i és V_i értékeket.

A program a `standard kimenetre` soronként kiírja a `can_escape` hívások visszatérési értékét: igaz esetén `1`-et, hamis esetén `0`-t. A program a megoldásod helyességét nem ellenőrzi!

A `be1.txt` a feladatleírásban található hívásokat írja le az itt specifikált formában, a `ki1.txt` pedig a helyes visszatérési értékeket tartalmazza.

Korlátok

$3 \leq R, C \leq 1\,000$

$L_{u,v} \in \{0, 1\}$ minden $0 \leq u < R$, $0 \leq v < C$ esetén, továbbá $L_{0,0} = L_{R-1,C-1} = 0$.

$1 \leq Q \leq 100\,000$

$N \geq 1$

$0 \leq U_i < R$, $0 \leq V_i < C$ minden $0 \leq i < N$ esetén.

$(U_i, V_i) \neq (0, 0)$ és $(U_i, V_i) \neq (R-1, C-1)$ minden $0 \leq i < N$ esetén.

Az `N` értékek összege az összes `can_escape` hívás során legfeljebb $500\,000$.

Időlimit: 2.0 s

Memórialimit: 256 MB

Pontozás

Részfeladat	Korlátok	Pontszám
1	a minta	0
2	$R, C \leq 200$ és $Q \leq 100$	15
3	az őrök nélküli labirintusban pontosan egy féle képpen lehet eljutni a $(0, 0)$ mezőről az $(R-1, C-1)$ mezőre úgy, hogy közben minden mezőre legfeljebb egyszer lépünk	18
4	az egymást követő `can_escape` hívások során `N` értéke nem növekedhet és az őrök pozíciói mindig az előző hívásban kapott pozíciók részhalmaza	28
5	nincsenek további megkötések	39