# Task: TRE
# Kincskereső


**CEOI 2011, Nap 1. Forrás fájl `tre.*` Memórialimit: 256 MB.** *9.07.2011*

Bytie egy parkban keres kincset. A park egyes tereit utak kötik össze, úgy, hogy bármely két tér között pontosan 1 útvonal van. A kincskeresés közben lépésről lépésre fedezi fel a park tereit: egy már ismert térről indulva adott számú, eddig ismeretlen téren halad keresztül (nem érint már ismert teret). Valamely két tér között félúton van a kincs.

## Párbeszéd

Három műveletet tartalmazó modult kell írnod, amelyekkel megvalósítható a keresés (saját függvényeket is írhatsz). Ezeket fogja hívni az értékelő program.

- `init` — ezt a függvényt egyszer hívja az értékelő, a futás elején. A másik két eljáráshoz szükséges kezdőértékek, adatszerkezetek beállítására használhatod.
  - C/C++: `void init();`
  - Pascal: `procedure init();`

  Az init meghívásakor egyetlen teret ismersz, az 1-es sorszámút.
- `path` — egy ismert térről induló útvonal mentén jut el új terekre.
  - C/C++: `void path(int a, int s);`
  - Pascal: `procedure path(a, s: longint);`

  Az útvonal az $a$. térről indul (ami már ismert volt) és $s$ számú új teret érint ($s > 0$). Az eljárásod tároljon olyan adatokat, ami alapján válaszolhatsz a harmadik eljárásban feltett kérdésekre! Az új terek sorszáma a bejárás sorrendjében a következő legkisebb pozitív egész lesz, amit eddig még nem használtunk. Ezt az eljárást az értékelő sokszor is meghívhatja.
- `dig` — kérdés a kincs helyére.
  - C/C++: `int dig(int a, int b);`
  - Pascal: `function dig(a, b: longint) : longint;`

  Meg kell adnia a paraméterként megadott $a$ és $b$ sorszámú tér közötti útvonalon levő középső tér sorszámát! Feltehető, hogy $a$ és $b$ már ismert és $a \neq b$. Ha páros számú tér van az összekötő útvonalon, akkor az $a$-hoz közelebbit kell megadni a két középsőből (ahogyan a példában is látszik)! Ezt az eljárást az értékelő sokszor is meghívhatja.

A modulodban **nem** olvashatsz semmit, sem a standard bemenetről, sem fájlból és **nem** írhatsz sem fájlba, sem standard kimenetre! the standard output nor to any file.

C/C++ modul **nem** tartalmazhat `main` függvényt! Pascal esetén unit-ot kell írnod (lásd a gépeden található mintát)!

## Fordítás

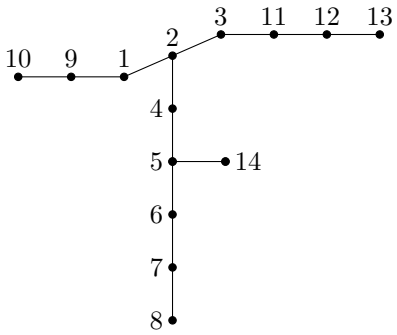
Az általad írt — `tre.c`, `tre.cpp`, vagy `tre.pas` — modul a következőképpen lesz fordítva::

- C: `gcc -O2 -static -lm tre.c tregrader.c -o tre`
- C++: `g++ -O2 -static -lm tre.cpp tregrader.cpp -o tre`
- Pascal:
```
ppc386 -O2 -XS -Xt tre.pas
ppc386 -O2 -XS -Xt tregrader.pas
mv tregrader tre
```

## Példa

A táblázat az értékelő által hívott eljárásokat és azok eredményét tartalmazza, beleértve a `dig` hívások elvárt eredményét is. Az ábra mutatja a park szerkezetét.

| Függvény hívás | Eredmény | Új terek sorszámai |
|---|---|---|
| `init();` | | 1 |
| `path(1, 2);` | | 2, 3 |
| `dig(1, 3);` | 2 | |
| `path(2, 5);` | | 4, 5, 6, 7, 8 |
| `dig(7, 3);` | 5 | |
| `dig(3, 7);` | 4 | |
| `path(1, 2);` | | 9, 10 |
| `path(3, 3);` | | 11, 12, 13 |
| `dig(10, 11);` | 1 | |
| `path(5, 1);` | | 14 |
| `dig(14, 8);` | 6 | |
| `dig(2, 4);` | 2 | |




# Korlátok

- AZ (`init`, `path`, és `dig`) függvényt összesen legfeljebb 400 000 -szer hívja az értékelő, és a parkban legfeljebb 1 000 000 000 tér van.
- 50 pontot szerezhet olyan tesztesetekre, ahol legfeljebb 400 000 tér van.
- 20 pontot szerezhet olyan tesztesetekre, ahol legleljebb 5 000 tér van és legfeljebb 5 000-szer hívódik összesen a `init`, `path`, és `dig` függvény.

# Gyakorlás

Egy egyszerű értékelő programot kapsz a kipróbáláshoz:

`tregrader.c`, `tregrader.cpp`, and `tregrader.pas`

a gépeden a **/home/zawodnik/tre/** könyvtárban. A megoldásodnak a

`tre.c`, `tre.cpp`, or `tre.pas`

fájlokban kell lennie a megfelelő könyvtárban (`c`, `cpp`, vagy `pas`). Itt találsz egy mintamegoldást, ami nem igazi megoldás. A megoldásodat így fordítsd:

```
make tre
```

C/C++ esetén kell a `treinc.h`, amelyet megtalálsz a megfelelő könyvtárban. Az így elkészült értékelő a standard bemenetről olvassa a meghívandó függvények nevét és paramétereit, és a `dig` függvényhívások eredményét a standard kimenetre írja. A bemenet leírása a következő formájú: Az első sor tartalmazza a műveletek $q$ számát. A következő $q$ sor mindegyikének első karaktere az `i`, `p`, vagy `d` betű, ezt követi két nemnegatív egész szám, egy-egy szóközzel elválasztva. A karakter a meghívandó függvényt azonosítja: `i` : `init`, `p` : `path`, and `d` : `dig`. Az egész számoka függvény arumentumai: $a$ és $s$ : `path`, $a$ és $b$ : `dig`. Az `i` esetén mindkét egész értéke 0. Az egyszerű értékelő **nem** ellenőrzi, hogy formailag helyes-e a bemenet, vagy hogy megfelel-e a feltételeknek. A példának megfelelő bemeneti állományt a `tre0.in` fájl tartalmazza.

```
12
i 0 0
p 1 2
d 1 3
p 2 5
d 7 3
d 3 7
p 1 2
p 3 3
d 10 11
p 5 1
d 14 8
d 2 4
```

Végrehajtáskor a bemenet átirányítást használd:

```
./tre < tre0.in
```

A `dig` hívások eredményét a standard kimeneten láthatod, a helyes eredményt pedig a `tre0.out` fájlban:

```
2
5
4
1
6
2
```

Az értékelő redszer (SIO) ellenőrzi, hogy a megoldásod a példára helyes-e.