Cerca en un BST (arbre de cerca binària) X75537_ca

Implementeu una funció **RECURSIVA** que, donat un arbre binari de cerca (BST) d'enters `t`, i un valor `x`, retorna un booleà indicant si `x` apareix a l'arbre. Aquesta és la capcelera:

```
// Pre: t és un BST
// Post: Retorna cert si i només si x apareix a t
bool searchInBST(BinTree<int> t, int x);
```

Recordeu que un BST és un arbre a on cada subarbre no buit $r(t_0, t_1)$ cumpleix que l'arrel r és major estricte que tots els valors que apareixen en el seu subarbre esquerre t_0, i r és menor estricte que tots els valors que apareixen en el seu subarbre dret t_1. La gràcia dels BST és que, per a trobar si un cert element hi apareix, ho podem fer més ràpid aprofitant el fet que els seus elements cumpleixen aquesta propietat d'ordenació. Tingueu en compte que els jocs de proves consistiran en arbres bastant equilibrats, així que valdrà la pena que feu això.
Aquí tenim un exemple de paràmetres d'entrada de la funció i la corresponent sortida:

```
t:         3
           |
    ------- -------
   |               |
   1               5
   |               |
    ----         ----
        |       |
        2       4

x: 2

=>

true
```

Fixeu-vos que l'enunciat d'aquest exercici ja ofereix uns fitxers que haureu d'utilitzar per a compilar: `main.cc, BinaryTree.hh, searchInBST.hh`. Us falta crear el fitxer `searchInBST.cc` amb els corresponents `includes` i implementar-hi la funció anterior. Només cal que pugeu `searchInBST.cc` al jutge.

Entrada

La primera linia de l'entrada descriu el format en el que es descriuen els arbres, o bé INLINEFORMAT o bé VISUALFORMAT.
Després ve la descripció d'un únic arbre binari d'enters, que és un BST.
Després segueixen un nombre arbitrari de casos. Cada cas consisteix en una línia amb un enter `x`.
Fixeu-vos en que el programa que us oferim ja s'encarrega de llegir aquesta entrada. Només cal que implementeu la funció abans esmentada.

Sortida

Per a cada cas, la sortida conté la corresponent indicació de si l'element pertany a l'arbre o no. Fixeu-vos en que el programa que us oferim ja s'encarrega d'escriure aquesta sortida. Només cal que implementeu la funció abans esmentada.

Exemple d'entrada 1

```
VISUALFORMAT
                      8
                      |
           ----------- -----------
          |                       |
          7                      24
          |                       |
      ----                ---------- -------
     |                   |                  |
     4                  15                  28
     |                   |                  |
 ---- ----        ------- -------         ----
|          |     |                |       |
0          5     10               17     25
                 |                |
             ---- ----        ---- ----
            |          |     |          |
            9         12    16         18
```

```
0
3
2
4
26
16
11
3
3
7
12
22
15
27
28
24
2
23
4
13
```

Exemple de sortida 1

```
Exists
Does not exist
Does not exist
Exists
Does not exist
Exists
Does not exist
Does not exist
Does not exist
Exists
Exists
Does not exist
Exists
Does not exist
Exists
Exists
Does not exist
Does not exist
Exists
Does not exist
```

Exemple d'entrada 2

```
INLINEFORMAT
-7(-8(-11(-15,-10),),9(0(-5(-6,-3),2(1,3)),13(10,)))
-15
-12
-13
-11
11
1
-4
-12
```

```
-12
-8
-3
7
0
12
13
9
-13
8
-11
-2
```

Exemple de sortida 2

```
Exists
Does not exist
Does not exist
Exists
Does not exist
Exists
Does not exist
Does not exist
Does not exist
```

```
Exists
Exists
Does not exist
Exists
Does not exist
Exists
Exists
Does not exist
Does not exist
Exists
Does not exist
```

Observació

La vostra funció i subfuncions que creeu han de treballar només amb arbres. Heu de trobar una solució **RECURSIVA** del problema.

Informació del problema

Autoria: PRO2

Generació: 2026-01-25T21:25:50.533Z