Adaptar BinaryTree per a mantenir informació sobre l'alçadaX38946_ca

L'objectiu d'aquest exercici és afegir un nou mètode `getHeight` a la classe Genèrica `BinaryTree` que retorni l'alçada de l'arbre. Definim que un arbre buit té alçada 0 i un arbre amb un sol node té alçada 1. Una opció seria que aquest mètode calculés aquesta alçada, per exemple recursivament, i la retornés, però aquest enfoc seria massa lent per a poder superar els jocs de proves privats. Aquesta operació hauria de tenir cost constant, i per això, convindrà afegir informació adicional a la classe que permeti mantenir actualitzada informació sobre l'alçada. A continuació donem una guia de com fer això.
D'entre els fitxers que s'adjunten en aquest exercici, trobareu `BinaryTree.old.hpp`, a on hi ha una implementació de la classe genèrica `BinaryTree`. En primer lloc, haureu de fer:

```
cp BinaryTree.old.hpp BinaryTree.hpp
```

A continuació, heu de fer tot un seguit de canvis sobre la classe `BinaryTree` definida a `BinaryTree.hpp`:

- Heu d'afegir un nou atribut `int height`.
- Heu d'afegir un nou mètode privat per a actualitzar l'alçada de l'arbre i l'alçada dels seus antecessors (els arbres que tenen a l'arbre actual com a subarbre). Això es pot fer de forma recursiva o iterativa. Una possible manera iterativa és:

  ```
  void updateHeight()
  {
  BinaryTree<T> *pt = this;
  while (pt != NULL) {
  if (pt->isEmpty()) pt->height = ...;
  else pt->height = ...;
  pt = pt->parent;
      }
  }
  ```

  Una possible manera recursiva és:

  ```
  void updateHeight()
  {
  if (isEmpty()) height = ...;
  else height = ...;
  if (parent != NULL) parent->updateHeight();
  }
  ```

- A les constructores i a l'operació d'assignació heu d'afegir crides a `updateHeight`.
- Heu d'implementar el mètode `getHeight`, simplement retornant el valor del nou atribut.

D'entre els fitxers que s'adjunten a l'exercici també hi ha `program.cpp` (programa principal) i `Makefile` per a compilar. Per a pujar la vostra solució, heu de crear el fitxer `solution.tar` així:

```
tar cf solution.tar BinaryTree.hpp
```

Entrada

El programa principal té una variable d'arbre d'enters `t`, inicialment buida, i llegeix instruccions que, o bé mostren com és `t`, o bé modifiquen algun subarbre de `t` o mostren l'alçada d'algun subarbre de `t`. Les instruccions que mostren `t` són simplement de la forma `<< t`. Les altres instruccions comencen per `t`, seguit d'una seqüència de `.left` o `.right`. Finalment, o bé la instrucció acaba amb `.height`, cas en el qual s'escriurà l'alçada del corresponent subarbre, o ve seguida de `= t'`, on `t'` és un string que representa un arbre, cas en el qual `t'` (com a arbre) serà assignat al corresponent subarbre de `t`. Per exemple:

```
t = 3(4,5(1,2))
<< t
t.height
t.left.height
t.right.height
t.right.left = 8(9,10)
<< t
t.right.height
```

La sortida de la seqüència anterior és:

```
3(4,5(1,2))
3
1
2
3(4,5(8(9,10),2))
3
```

Com podeu observar, el `height` d'un arbre que està per sobre del que hem assignat també ha estat actualitzat.
Se suposa que la seqüència d'entrada serà correcta (sense accessos fora de l'arbre, tot i que sí que es pot accedir a subarbres buits de l'arbre).
El programa principal que us oferim ja s'encarrega de llegir aquestes entrades i fer les crides als corresponents mètodes de la classe BinaryTree. Només cal que feu les modificacions abans esmentades dins el fitxer `BinaryTree.hpp`.

Sortida

Per a cada instrucció `<< t`, s'escriurà el contingut actual de l'arbre. Per a cada instrucció acabada en `height`, s'escriurà l'alçada del subarbre indicat. El programa que us oferim ja fa això. Només cal que feu les modificacions abans esmentades dins el fitxer `BinaryTree.hpp`.

Exemple d'entrada 1

```
t = 7(2,5)
```

```
t.height
<< t
t = 5(,1)
```

```
t.height
<< t
t.left = 4(2(,3),2)
t.left.height
<< t
t.left.left = 7(3,)
t.left.height
<< t
t.right = 5(6(1,),2)
t.height
<< t
t.left.right = 5(,8(,3))
t.height
<< t
t.left.right.left = 1(3,4(3,2))
t.height
<< t
t.right.right = 2(5(2,2),)
t.right.right.height
<< t
t.left.left = 6
t.left.left.height
<< t
t.right.right.left = 1
t.right.height
<< t
```

Exemple de sortida 1

```
2
7(2,5)
2
5(,1)
3
5(4(2(,3),2),1)
3
5(4(7(3,),2),1)
4
5(4(7(3,),2),5(6(1,),2))
5
5(4(7(3,),5(,8(,3))),5(6(1,),2))
6
5(4(7(3,),5(1(3,4(3,2)),8(,3))),5(6(1,),2))
3
5(4(7(3,),5(1(3,4(3,2)),8(,3))),5(6(1,),2(5(2,2),)))
1
5(4(6,5(1(3,4(3,2)),8(,3))),5(6(1,),2(5(2,2),)))
3
5(4(6,5(1(3,4(3,2)),8(,3))),5(6(1,),2(1,)))
```

Informació del problema

Autoria: PRO1

Generació: 2026-01-25T21:10:50.557Z