
Arbre binari. Calcula arbre amb factors de pes**X53343_ca**

Donada la classe *Abin* que permet gestionar arbres binaris usant memòria dinàmica, cal implementar el mètode

```
void arbre_factors_pes ();
```

que modifica el contingut de l'arbre per tal de guardar a cada node el factor de pes (diferència entre la quantitat de nodes del fill esquerra i la quantitat de nodes del fill dret).

Cal enviar a jutge.org la següent especificació de la classe *Abin* i la implementació del mètode dins del mateix fitxer.

```
include <cstdlib>
#include <iostream>
using namespace std;
typedef unsigned int nat;

template <typename T>
class Abin {
public:
    Abin(): _arrel (NULL) {};
    // Pre: cert
    // Post: el resultat és un arbre sense cap element
    Abin(Abin<T> &ae, const T &x, Abin<T> &ad);
    // Pre: cert
    // Post: el resultat és un arbre amb un element i dos subarbres

    // Les tres grans
    Abin(const Abin<T> &a);
    ~Abin();
    Abin<T>& operator=(const Abin<T>& a);

    // operador ii d'escriptura
    template <class U> friend std::ostream& operator<<(std::ostream&, const Abin<U> &a);

    // operador jj de lectura
    template <class U> friend std::istream& operator>>(std::istream &, Abin<U> &a);

    // Modifica el contingut de l'arbre per tal de guardar a cada node el factor
    // de pes (diferència entre quantitat nodes fill esquerra i quantitat nodes fill dret).
    void arbre_factors_pes ();
};

private:
    struct node {
        node* f_esq ;
        node* f_dret ;
        T info ;
    };
};
```

```

node* _arrel ;
static node* copia_nodes (node* m);
static void esborra_nodes (node* m);
static void print_nodes (node* m, ostream &os, string d1);

// Aquí va l'especificació dels mètodes privats addicionals
};

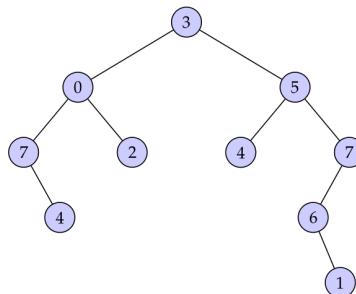
// Aquí va la implementació del mètode arbre_factors_pes

```

Per testejar la solució, jutge.org ja té implementats la resta de mètodes de la classe *Abin* i un programa principal que llegeix un arbre binari i després crida el mètode *arbre_factors_pes*.

Entrada

L'entrada consisteix en la descripció d'un arbre binari d'enters (el seu recorregut en preordre, en el qual inclou les fulles marcades amb un -1). Per exemple, l'arbre (mira el PDF de l'enunciati)



es descriuria amb

3 0 7 -1 4 -1 -1 2 -1 -1 5 4 -1 -1 7 6 -1 1 -1 -1 -1

Sortida

El contingut de l'arbre binari abans i després de cridar el mètode *arbre_factors_pes*.

Observació

Només cal enviar la classe requerida i la implementació del mètode *arbre_factors_pes*. Pots ampliar la classe amb mètodes privats. Segueix estrictament la definició de la classe de l'enunciati.

Exemple d'entrada 1

7 5 -1 -1 8 9 -1 -1 4 6 -1 -1 3 -1 -1

Exemple de sortida 1

```

[7]
 \__[8]
 |  \__[4]
 |  |  \__[3]
 |  |  |  \__.
 |  |  |  \__.
 |  |  \__[6]
 |  |  \__.
  
```

```

|      |
|      \__ [ 9 ]
|          \__ .
|
\__ [ 5 ]
    \__ .
    \__ .

```

[-4]

```

\__ [ -2 ]
|   \__ [ 0 ]

```

Exemple d'entrée 2

3 0 7 -1 4 -1 -1 2 -1 -1 5 4 -1 -1 7 6 -1

```
graph TD; Root[0] --- L1_1[0]; Root --- L1_2[0]; L1_1 --- L2_1[0]; L1_1 --- L2_2[0]; L1_2 --- L2_3[0]; L1_2 --- L2_4[0]; L2_1 --- L3_1[0]; L2_1 --- L3_2[0]; L2_3 --- L3_3[0]; L2_3 --- L3_4[0]; L2_4 --- L3_5[0]; L2_4 --- L3_6[0]; L3_1 --- L4_1[0]; L3_1 --- L4_2[0]; L3_3 --- L4_3[0]; L3_3 --- L4_4[0]; L3_5 --- L4_5[0]; L3_5 --- L4_6[0]; L4_1 --- L5_1[0]; L4_3 --- L5_2[0]; L4_5 --- L5_3[0];
```

Exemple de sortida 2

```
graph TD; 13[13] --> 5[5]; 13 --> 0[0]; 5 --> 7[7]; 5 --> 4[4]; 7 --> 6[6]; 7 --> 1[1]; 4 --> 4_1[4]; 4 --> 2[2]; 4 --> 7_1[7]; 2 --> 1_1[1]; 2 --> 0_1[0]; 1_1 --> 0_1; 1_1 --> 1_2[1]; 1_2 --> 1_3[1]; 1_2 --> 0_2[0]; 7_1 --> 4_2[4]; 7_1 --> 2_1[2]; 4_2 --> 1_4[1]; 4_2 --> 0_3[0]; 2_1 --> 1_5[1]; 2_1 --> 0_4[0]; 1_5 --> 0_4;
```

A binary search tree diagram with root node 13. The tree has the following structure:

- Root: 13
- Left child: 5
- Right child: 0
- Node 5 has left child 7 and right child 4.
- Node 7 has left child 6 and right child 1.
- Node 4 has left child 4 (leaf), right child 2, and right child 7 (leaf).
- Node 2 has left child 1 (leaf) and right child 0 (leaf).
- Node 7 (under 4) has left child 4 (leaf) and right child 2 (leaf).
- Node 2 (under 7) has left child 1 (leaf) and right child 0 (leaf).

```

[-1]
 \__[-2]
 | \__[2]
 | | \__.
 | | \__[-1]
 | | | \__[0]
 | | | | \__.
 | | | | \__.
 | | | | \__.
 | | | | \__[0]
 | | | | \__.
 | | | | \__.
 \__[1]
   \__[0]
   | \__.
   | \__.
 \__[-1]
   \__[0]
   | \__.
   | \__.
   \__.

```

Exemple d'entrada 3

-1

Exemple de sortida 3

.

.

Exemple d'entrada 4

3 -1 -1

Exemple de sortida 4

[3]

__.

__.

[0]

__.

__.

Exemple d'entrada 5

3 2 -1 -1 -1

Exemple de sortida 5

[3]

__.

__[2]

__.

__.

[1]

__.

__[0]

__.

__.

Exemple d'entrada 6

3 -1 2 -1 -1

Exemple de sortida 6

[3]

__[2]

| __.

| __.

__.

[-1]

__[0]

| __.

| __.

__.

Exemple d'entrada 7

-3 -2 -1 -1 -4 -1 -1

Exemple de sortida 7

[-3]

__[-4]

| __.

| __.

__[-2]

__.

__.

[0]

__[0]

| __.

| __.

__[0]

__.

__.

Informació del problema

Autor : Jordi Esteve
Generació : 2021-10-21 01:05:40

© *Jutge.org*, 2006–2021.
<https://jutge.org>