Creació d'un BST intersecció de dos BSTs — X14733_ca

Donada la classe *dicc* que permet gestionar diccionaris on només hi guardem claus úniques usant arbres binaris de cerca (BST), cal implementar el mètode

```
void interseccio (const dicc<Clau> &d2, dicc<Clau> &dr) const;
// Pre: El diccionari dr està buit
// Post: dr conté les claus de la intersecció del p.i. amb d2
```

Les claus són del tipus *Clau* que admet una relació d'ordre total, és a dir, tenim una operació de comparació < entre claus.
Cal enviar a jutge.org la següent especificació de la classe *dicc* i la implementació del mètode dins del mateix fitxer. La resta de mètodes públics i privats ja estan implementats. Indica dins d'un comentari a la capçalera del mètode el seu cost en funció del nombre d'elements $n1$ del diccionari del p.i. i nombre d'elements $n2$ del diccionari $d2$.

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

template <typename Clau>
class dicc {

  public:
    // Constructora per defecte. Crea un diccionari buit.
    dicc ();

    // Destructora
    ~dicc ();

    // Insereix la clau k en el diccionari. Si ja hi era, no fa res.
    void insereix (const Clau &k);

    // Imprimeix els elements del diccionari ordenats de menor a major
    void print () const;

    // Retorna la mida del BST que implementa el diccionari
    nat mida() const;

    // Retorna l'altura del BST que implementa el diccionari
    nat altura () const;

    void interseccio (const dicc<Clau> &d2, dicc<Clau> &dr) const;
    // Pre: El diccionari dr està buit
    // Post: dr conté les claus de la intersecció del p.i. amb d2

  private:
    struct node {
      Clau _k;          // Clau
```

```
    node* _esq;     // fill esquerre
    node* _dret;    // fill dret
    node(const Clau &k, node* esq = NULL, node* dret = NULL);
  };
  node *_arrel;

  static void esborra_nodes(node* m);
  static node* insereix_bst (node *n, const Clau &k);
  static void print(node *n);
  static nat mida(node *n);
  static nat altura(node *n);

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

// Aquí va la implementació dels mètodes públics i privats
```

Degut a que jutge.org només permet l'enviament d'un fitxer amb la solució del problema, en el mateix fitxer hi ha d'haver l'especificació de la classe i la implementació del mètode *interseccio* (el que normalment estarien separats en els fitxers *.hpp* i *.cpp*).
Per testejar la classe disposes d'un programa principal que llegeix dos diccionaris d'enters, després crida el mètode *interseccio* i finalment mostra el contingut del diccionari amb la intersecció dels dos diccionaris inicials. També s'indica si el BST del diccionari resultant està suficientment balancejat.

Entrada

L'entrada conté dues línies formades per seqüències d'enters, són els elements que tindran els dos diccionaris inicials.

Sortida

A la sortida apareixen ordenats i separats per espais, els elements del diccionari resultant que conté la intersecció dels dos diccionaris inicials. A la segona línia apareix el text "BST poc balancejat" o "BST suficient balancejat", depenen de si l'altura del BST resultant supera o no $min(mida, log_2(mida) * 10)$, sent *mida* el nombre de nodes de l'arbre resultant.

Observació

Només cal enviar l'especificació de la classe *dicc*, la implementació del mètode *interseccio* i el seu cost en funció del nombre d'elements $n1$ i $n2$ dels dos diccionaris inicials. Pots ampliar la classe amb mètodes privats. Segueix estrictament la definició de la classe de l'enunciat.
Primer cal crear un BST amb el resultat correcte. Posteriorment obtenir un BST més balancejat si no ho està suficientment. Hi ha diferents tècniques per aconseguir-ho, una d'elles és alternar de forma aleatòria l'ordre en que es recorren els BSTs inicials (esquerra-dreta o dreta-esquerra), et pot ser d'utilitat l'expressió *rand*()%2 que simula el cara/creu d'una moneda llançada a l'aire, el seu resultat és 0 o 1.
Els primers tres exemples mostren la intersecció de dos diccionaris buits o un de buit i un que no ho és.

Exemple d'entrada 1

Exemple de sortida 1

```
BST suficient balancejat
```

Exemple d'entrada 2

```
5 -3 8 2 -1 7 -7 -6
```

Exemple de sortida 2

```
BST suficient balancejat
```

Exemple d'entrada 3

```
7 -2 9 5 -3 2 -7
```

Exemple de sortida 3

```
BST suficient balancejat
```

Exemple d'entrada 4

```
5 -3 8 2 -1 7 -7 -6
7 -2 9 5 -3 2 -7
```

Exemple de sortida 4

```
-7 -3 2 5 7
BST suficient balancejat
```

Exemple d'entrada 5

```
5 -5 -3 9 -9 2 -2 1 -1 7 -7 0 4 -4 8 -8 6
2 10 4 12 8 14 0 10 14 6
```

Exemple de sortida 5

```
0 2 4 6 8
BST suficient balancejat
```

Exemple d'entrada 6

```
-191 427 -13 -2 -11 353 10 -333 -85 -4763 400 -4743 -2713 4853 443 483 4313 4383
```

Exemple de sortida 6

```
BST suficient balancejat
```

Informació del problema

Autoria: Jordi Esteve

Generació: 2026-01-25T13:47:07.410Z