Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona

Cognoms, Nom | D.N.I.

Titulació: Grau en Enginyeria Informàtica
Assignatura: Programació 2 (PRO2)
Duració:

Curs: Q1 2018–2019 (2n Parcial)
Data: 14 de Gener de 2019

1. **(50%)** Considerem la següent classe `Llista`:

```
template <class T>
class Llista {
public:
  ...
  void splice(const T& x, Llista& l2);
  ...
private:
  struct node_llista {
    T info;
    node_llista* ant;
    node_llista* seg;
  };
  int longitud;
  node_llista* primer_node;
  node_llista* ultim_node;
  node_llista* act;

  void nullify();
  static node_llista* search(node_llista* p, const T& x);
};
```

que representa una llista d'`T`s mitjançant una cadena doblement enllaçada de nodes. Els apuntadors `primer_node` i `ultim_node` apunten al primer i últim nodes de la llista, respectivament. Si la llista és buida llavors `primer_node == ultim_node == act == nullptr`. Per a una llista no buida, el predecessor del primer element és nul (`primer_node -> ant == nullptr`) i el sucessor de l'últim element és nul (`ultim_node -> seg == nullptr`). L'apuntador `act` apunta al punt d'interès. L'atribut `longitud` és el nombre d'elements de la llista.

(a) (10%) Implementa el mètode privat

```
// Pre: si no està buida, la cadena de nodes del
// p.i. està compartida amb una altra llista
// Post: el p.i. representa a una llista buida
  void nullify();
```

Com que la cadena de nodes de la llista implícita està compartida (si no és buida) i no volem efectes secundaris sobre altres llistes, aquesta operació **no** ha de destruir la cadena de nodes.

(b) (15%) Implementa el mètode privat

```
// Pre: cert
// Post: retorna un apuntador al primer node que conté x
// en la cadena de nodes que comença a p, o nullptr
// si no hi ha cap node a partir de p que contingui x
  static node_llista* search(node_llista* p, const T& x);
```

(c) (25%) Implementa el mètode públic

```
// Pre: cert
// Post: insereix la llista l2 just davant de la primera
// aparició de l'element x en el p.i. si x està present,
// o al final del p.i. si x no està present; en qualsevol cas
// la llista l2 queda buida, i el punt d'interès del p.i.
// queda inalterat
  void splice(const T& x, Llista& l2);
```

Per exemple, si tenim les llistes $l1 = [3, 7, -2, 5, 0, 7]$ i $l2 = [3, 4, 3, 4]$ llavors després de la crida `l1.splice(7, l2)` tindrem $l1 = [3, 3, 4, 3, 4, 7, -2, 5, 0, 7]$ i $l2 = []$. Si la crida hagués estat `l1.splice(8,l2)` llavors, un cop acabada la crida, tindríem $l1 = [3, 7, -2, 5, 0, 7, 3, 4, 3, 4]$ i $l2 = []$. Si $l1 = []$ i $l2 = L2$ aleshores després de la crida `l1.splice(x, l2)` tindrem $l1 = L2$ i $l2 = []$, siguin quins siguin el valor de x i els continguts de $l2$.

Es valorarà l'eficiència de la solució proposada. La teva solució no pot utilitzar memòria extra (a part de variables auxiliars senzilles com ara booleans o apuntadors) i no ha de crear ni destruir nodes. Pots suposar que els operadors Booleans `==` i `!=` entre objectes de la classe `T` estan definits.

Vigila que la teva solució tracti adequadament, a més dels que ja apareixen en els exemples, la resta dels casos extrems, com ara que la llista $l2$ sigui una llista buida, o que la llista $l2$ s'hagi de transferir just davant del primer element de la llista implícita.

SOLUCIÓ:

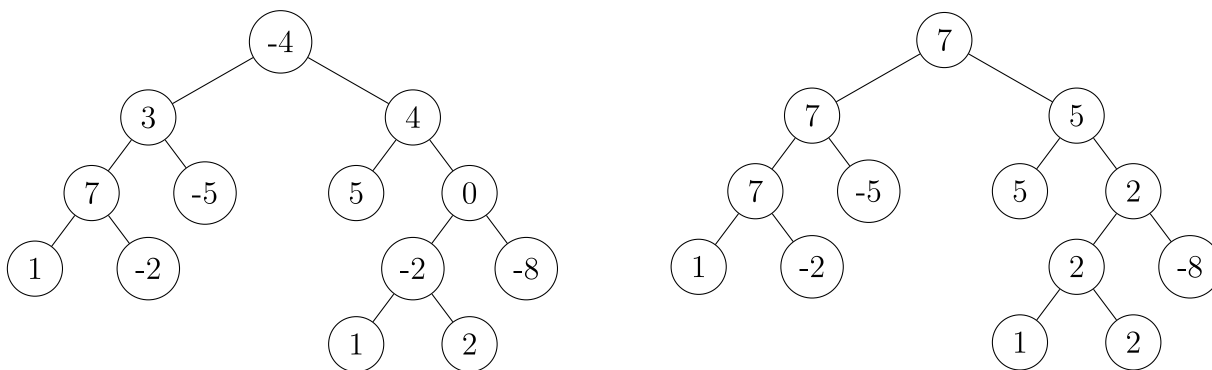
Cognoms, Nom D.N.I.

2. **(50%)** Considerem la següent classe `Arbre`:

```
template <class T>
class Arbre {
public:
  ...
// Pre: el p.i. no és buit i tot node té exactament zero
//      o 2 fills no buits.
// Post: retorna l'arbre de màxims corresponent al p.i.
//       (s'explica a continuació)
Arbre arbre_maxims();
  ...
private:
   struct node_arbre {
     T info;
     node_arbre* segE;
     node_arbre* segD;
   };
   node_arbre* primer_node;
   // altres operacions que pots afegir com a auxiliars
   // d'arbre_maxims
};
```

que representa a un arbre binari d'`T`s, mitjançant un únic apuntador (`primer_node`) a l'arrel de l'arbre.

Implementa el mètode `arbre_maxims` que torna un `Arbre` amb idèntica estructura que el paràmetre ímplicit i ón cada node conté el màxim dels nodes del subarbre corresponent en l'arbre original. Per exemple si `a` és l'arbre a la part esquerra de la figura, aleshores `a.arbre_maxims()` retorna l'arbre de la part dreta.



Es valorarà l'eficiència de la teva solució. El mètode `arbre_maxims` ha de tenir cost lineal respecte al nombre de nodes de l'arbre implícit. Pots assumir que tots els operadors Booleans de comparació (`<`,`<=`, `>`, ...) entre objectes de la classe `T` estan definits.

SOLUCIÓ: