Final Programació 2

Enginyeria Informàtica
Juny 2015
Temps estimat: 2h 45min

- Feu bona lletra.
- Lliureu els dos problemes (en full separats, encara que siguin en blanc).
- En tots els problemes està prohibit usar mètodes públics de la classe; s'han de resoldre accedint directament a la implementació.
- Els codis no s'han de justificar, però poden incloure comentaris aclaridors si ho creieu convenient. Si feu servir operacions auxiliars, és necessari que les especifiqueu.
- Es valorarà l'eficiència de les solucions.

Problema 1 (4 punts)

Considerem la representació habitual amb nodes de la classe `Cua` per manegar cues genèriques d'elements de tipus `T`

```
template <class T> class Cua {
  private:
   struct node_cua {
     T info;
     node_cua* seg;
   };
   int longitud;
   node_cua* prim;
   node_cua* ult;
   ... // especificació i implementació d'operacions privades
  public:
   ... // especificació i implementació d'operacions públiques
};
```

Els nodes són simplement encadenats amb punters al següent (`seg`). Una cua té tres atributs: la longitud i dos punters a nodes, un al primer element (`prim`, el més antic) i un per l'últim (`ult`, el més recent).

Volem implementar dins de la classe `Cua` una operació que extregui la subcua més llarga entre dues aparicions d'un element donat (sense cap altra aparició entre elles, i en cas d'empat, la més recent) amb la següent especificació:

```
void subcua_mes_llarga(const T& elem, Cua& sub)
/* Pre: el p.i. conté una cua C, sub és buida */
/* Post: sub és una cua amb els elements de la subcua més llarga entre dues
         aparicions d'elem a C; el p.i. és com C però sense aquesta subcua
*/
```

Exemples:

1. Si el p.i. és originalment:

   `2 1 2 3 4 1 2 3 4 5 1 2 1 2 3 4`

   i elem és `1`, el p.i. quedaria:

`2 1 2 3 4 1 1 2 1 2 3 4`

i sub contindria la cua:

`2 3 4 5`

2. Si el p.i. és originalment:

   `2 3 4 1 5 6`

   i elem és `1`, el p.i. quedaria igual i sub seria la cua buida.

3. Si el p.i. és originalment:

   `1 1 1`

   i elem és `1`, el p.i. quedaria igual i sub seria la cua buida.

Es demana una implementació d'aquesta operació que accedeixi directament als atributs de la classe `Cua`.

Problema 2 (7 punts)

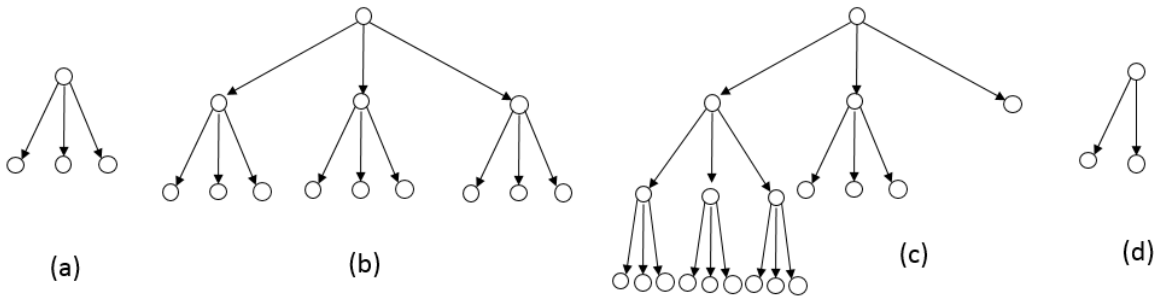
Considerem la classe que representa arbres N-aris, implementada com s'ha explicat a classe:

```
template <class T> class ArbreNari {
  private:
   struct node {
     T info;
     vector<node*> seg;
   };
   int N;   // nombre de fills de cada subarbre
   node* primer;
   ... // especificació i implementació d'operacions privades
  public:
   ... // especificació i implementació d'operacions públiques
};
```

Recordeu que en un arbre N-ari tots els nodes tenen exactament N fills, que poden ser buits, cosa que es tradueix en la representació en què tots els vectors seg tenen com a mida l'atribut N de l'arbre i que algunes de les seves components poden ser `NULL`.

Diem que un arbre N-ari d'altura h és *complet* si conté tots els nodes que pot contenir un arbre N-ari d'altura h. Fixeu-vos que, recursivament, tots els fills d'un arbre complet d'altura h són arbres complets d'altura $h-1$. L'arbre buit també és complet.

Així, dels quatre arbres 3-aris següents, els arbres (a) i (b) són complets; (c) no és complet, malgrat que els seus tres fills ho són, i (d) tampoc no és complet perquè el seu tercer fill és buit. (En aquest dibuix s'ometen els arbres buits).



Atenció: els dos apartats 2.1 i 2.2 són independents. No suggerim usar la funció feta a 2.1 per solucionar 2.2, i no es pot usar a 2.1 la funció `altura` que s'esmenta a 2.2.

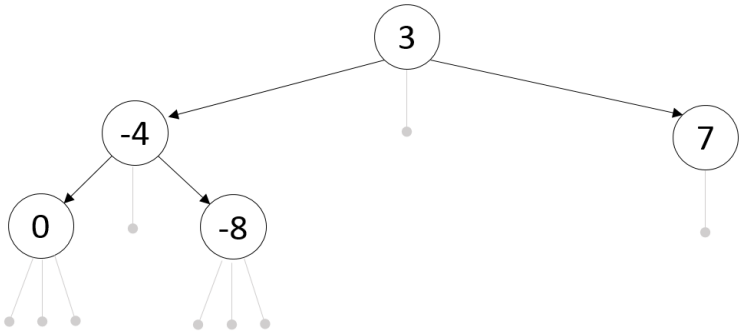
Apartat 2.1) [4 punts] Feu una funció pròpia de la classe `ArbreNari<T>`

```
bool es_complet() const;
```

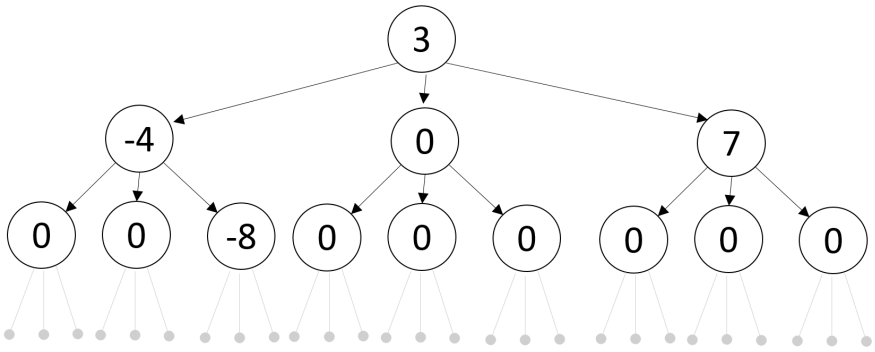
que digui si el seu paràmetre implícit és complet o no.

Apartat 2.2) [3 punts] Volem fer un mètode `completa(const T& v)` de la classe `ArbreNari<T>` que afegeixi el mínim de nodes al seu paràmetre implícit per fer-lo complet. En particular, si ja és complet el deixa intacte. Altrament, tots els nodes que calgui afegir han de contenir el valor `v`.

Per exemple, si `T` és `int`, la crida `completa(0)` aplicada a l'arbre



hauria de transformar-lo en l'arbre



S'ha decidit implementar aquesta funció així:

```
// Pre: el p.i. és un arbre N-ari A
// Post: el p.i. és l'arbre complet que completa A amb el mínim nombre de nodes;
//       els nodes que són al p.i. però no eren a A contenen el valor v
void completa(const T& v) {
    int h = altura(primer);
    i_completa(primer,N,h,v);
}
```

on `static int altura(node* q)` és una funció privada que retorna l'altura de la jerarquia de nodes que penja de `q` i que se suposa ja implementada. `i_completa` té l'especificació següent:

```
// Pre: h >= altura(p), n es l'aritat de l'ArbreNari<T> al que pertany
// la jerarquia apuntada per p
// Post: la jerarquia apuntada per p ha estat completada a un arbre
// complet d'altura h, afegint els nodes necessaris amb v al camp info
static void i_completa(node*& p, int n, int h, const T& v);
```

Doneu una implementació eficient d'`i_completa`.

Fixeu-vos en el `&` del paràmetre `p`, i en què `p` bé podria ser `NULL` inicialment.