Cognoms, Nom (A	Amb lletres majúscules)	D.N.I.

Titulació: Grau en Enginyeria Informàtica

Assignatura: Programació 2 (PRO2)

Curs: Q1 2022–2023 (2n Parcial)

Data: 12 de gener de 2023

Duració: 2h 30m

1. (5 punts) Multiconjunt ordenat per freqüència. Farem servir les llistes habituals excepte pel fet que a cada node se li ha afegit un camp int que serveix per a comptar quants cops s'ha afegit la info. Aquest nou camp es diu freq. La llista està ordenada descendentment per freqüència i els elements amb una mateix freqüència estaran ordenats decreixentment pel temps que fa que tenen la freqüència.

Ens donen la següent representació amb apuntadors i memòria dinàmica per a una classe llista amb punt d'interès (act) i encadenament doble.

```
template <class T> class Llista {
private:
    struct node_llista{
        T info;
        int freq; // camp afegit
        node_llista* seg;
        node_llista* ant;
    };
    int longitud;
    node_llista* primer_node;
    node_llista* ultim_node;
    node_llista* act;
    // No hi ha cap camp info repetit.
    // Els nodes estan ordenats decreixentment pel camp freq.
    // El node de cada freqüència afegit més recentment és
    // el darrer de la seva freqüència.
    ... // operacions privades
public:
    ... // operacions públiques
}
```

Es demana implementar una operació per afegir un element a la llista. Si no hi és, s'afegeix amb freqüència 1, si hi és, se li suma 1 a la freqüència.

Continua a l'altra cara.

Un exemple del funcionament de l'operació afegir a partir d'una llista buida 1:

```
l.afegir(10); l = [10, 1]
l.afegir(20); l = [10, 1] [20, 1]
l.afegir(10); l = [10, 2] [20, 1]
l.afegir(20); l = [10, 2] [20, 2]
l.afegir(20); l = [20, 3] [10, 2]
l.afegir(10); l = [20, 3] [10, 3]
l.afegir(30); l = [20, 3] [10, 3] [30, 1]
l.afegir(30); l = [20, 3] [10, 3] [30, 2]
l.afegir(30); l = [20, 3] [10, 3] [30, 3]
l.afegir(30); l = [30, 4] [20, 3] [10, 3]
Feu servir la següent especificació:

// Pre: cert
// Post: Si valor no hi és, s'afegeix amb freqüència 1, // si hi és, se li suma 1 a la freqüència.
void afegir(const T& valor);
```

No fer servir operacions públiques de la classe Llista.

SOLUCIÓ:

Posar el nom i contestar al full de resposta.

```
void afegir(const T& valor) {
  node_llista* n;
  n =
  bool trobat = false;
                                                                  ){
  while(
    if (
                                 ) trobat = true;
    else
  }
                               ){ // valor no hi era
  if (
    n =
                             ;
    if (
                                 ){ // és l'únic node
    }
    \verb"else" \{ \ // \ \ hi \ \ havia \ \ altres \ \ nodes
```

Continua a l'altra cara.

longitud++;

}

}

```
else{ // valor sí hi era
     node_llista* aux = n->ant; // cercar on posar el node
                                                             ){
     while(
       aux = aux->ant;
     if (aux != n->ant){ // cal fer canvis
       if (n == ultim_node){
       }
       else{
       }
       if (aux == nullptr){
       }
       \verb"else" \{
}
```

2. (5 punts) Tenim un arbre binari genèric amb la seva representació habitual que recordem aquí:

```
template <typename T>
class Arbre{
  private:
    struct node_arbre {
        T info;
        node_arbre* segE;
        node_arbre* segD;
    };
    node_arbre* primer_node;
    static void esborra_node_arbre(node_arbre* m); // Es pot usar
    ...
  public:
    ...
};
```

Sigui $k \ge 0$ un enter i v un vector amb 2^k elements de tipus T. Tenim un arbre binari que compleix:

- (a) Tots els seus camins tenen longitud al menys k+1.
- (b) Tots els nodes tenen o zero o dos successors.

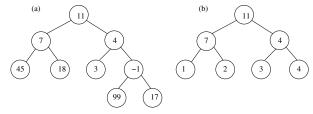
Direm que un arbre està normalitzat amb un enter k i un vector v si compleix:

- (a) Els k primers nivells no canvien.
- (b) Tots els seus camins tenen exactament longitud k + 1, per tant els elements de nivell k + 1 són fulles.
- (c) L'ordre del valor de les fulles quan l'arbre es recorre en preordre correspon amb l'ordre dels elements de v.

Es pot extendre el concepte de normalització a jerarquies de nodes.

Per acabar d'aclarir el concepte, oferim el següent exemple:

L'arbre (a) normalitzat amb k = 2 i $v = \{1, 2, 3, 4\}$ és l'arbre (b).



Continua a l'altra cara.

Implementeu els següents mètodes:

```
// Pre: k \geq 0, la mida de v[i..j] és 2^k, 0 \leq i \leq j < v.size(), // tots els camins de la jerarquia de nodes a partir d'n // tenen al menys longitud k+1, // tots els nodes de la jerarquia de nodes a partir d'n // tenen o zero o dos successors // Post: la jerarquia de nodes a partir d'n // està normalitzada amb k i v[i..j] static void normalitza_rec(node_arbre* n, int k, const vector<T>& v, int i, int j); // Pre: k \geq 0, la mida de v és 2^k, // tots els camins del p.i. tenen al menys longitud k+1, // tots els nodes del p.i. tenen o zero o dos successors // Post: el p.i. està normalitzat amb k i v void normalitza(int k, const vector<T>& v);
```

No fer servir operacions públiques de la classe Arbre.

SOLUCIÓ:

Posar el nom i contestar al full de resposta.

```
// Pre: k \geq 0 \,,\,la mida de v[i..j] és 2^k \,,\, 0 \leq i \leq j < v.size() \,,\,
// tots els camins de la jerarquia de nodes a partir d'n
// tenen al menys longitud k+1,
// tots els nodes de la jerarquia de nodes a partir d'n
// tenen o zero o dos successors
// Post: la jerarquia de nodes a partir d'n
// està normalitzada amb k i v[i..j]
static void normalitza_rec(node_arbre* n, int k,
                             const vector <T>& v, int i, int j){
  if (
                            ) { // Cas base
  }
  else {
    int aux =
  }
}
// Pre: k \ge 0, la mida de v és 2^k,
// tots els camins del p.i. tenen al menys longitud k+1,
// tots els nodes del p.i. tenen o zero o dos successors
// Post: el p.i. està normalitzat amb k i v
void normalitza(int k, const vector<T>& v){
}
```