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

Titulació: Grau en Enginyeria Informàtica

Assignatura: Programació 2 (PRO2)

Curs: Q2 2022–2023 (2n Parcial)

Data: 15 de juny 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 la freqüència de la info. Aquest nou camp es diu freq. La llista està ordenada descendentment per freqüència i els elements amb una mateix freqüència estaran ordenats creixentment 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;
    // INVARIANT DE LA REPRESENTACIÓ:
    // No hi ha cap camp info repetit.
    // No hi ha cap node amb freqüència 0 (o negativa)
    // Els nodes estan ordenats decreixentment pel camp freq.
    // El node de cada freqüència més recent és
    // el primer de la seva freqüència.
    ... // operacions privades
public:
    ... // operacions públiques
```

Es demana implementar una operació per a restar la quantitat q de la freqüència d'un valor v. Si la freqüència és igual o més petita que q, s'esborren totes les aparicions de v i també l'element que conté v. En cas contrari s'esborren q aparicions de v. S'indicarà també quantes aparicions s'han pogut esborrar. $Continua\ a\ l'altra\ cara$.

Feu servir la següent especificació:

```
// Pre: q > 0
// Post: S'ha intentat esborrar q aparicions de v. Si q és més gran o
// igual que el nombre d'aparicions de v, s'esborren totes.
// Es torna el nombre d'aparicions esborrades.
// act és nullptr
int esborrar(const T& v, int q);
```

Un exemple del funcionament de l'operació afegir a partir d'una llista 1. El primer element de cada parell correspon al camp info i el segon al camp freq.

```
1 = { [30, 9] [50, 5] [20, 4] [40, 4] [10, 4] }
1.esborrar(30,67);
1 = { [50, 5] [20, 4] [40, 4] [10, 4] } i esborra 9
1.esborrar(10,6);
1 = { [50, 5] [20, 4] [40, 4] } i esborra 4
1.esborrar(20, 4);
1 = { [50, 5] [40, 4] } i esborra 4
```

Un altre exemple del funcionament de l'operació afegir a partir d'una llista 1.

```
1 = { [50, 10] [30, 9] [40, 8] [10, 5] [20, 4] }
1.esborrar(50, 5);
1 = { [30, 9] [40, 8] [50, 5] [10, 5] [20, 4] } i esborra 5
1.esborrar(30, 6);
1 = { [40, 8] [50, 5] [10, 5] [20, 4] [30, 3] } i esborra 6
1.esborrar(30, 2);
1 = { [40, 8] [50, 5] [10, 5] [20, 4] [30, 1] } i esborra 2
1.esborrar(20, 4);
1 = { [40, 8] [50, 5] [10, 5] [30, 1] } i esborra 4
```

Recordeu que el p.i. ha de complir l'invariant de la representació. No feu servir operacions públiques de la classe Llista.

SOLUCIÓ:

Posar el nom i contestar al full de resposta.

```
int esborrar(const T& valor, int quantitat) {
 act = nullptr;
 node_llista* n;
 bool trobat = false;
                                   ){
  while (
    if (
                                  ) trobat = true;
    else
 }
  int esborrats;
  if (n == nullptr)
  else{
    int queden;
    if (n-) freq >= quantitat) esborrats =
    else esborrats =
    queden =
    if (queden == 0) { // Cal esborrar el node
      if (longitud == 1)
      else if (n ==
                                               ) {
      else if (n ==
                                               ) {
      else {
      }
    }
```

Continua a l'altra cara.

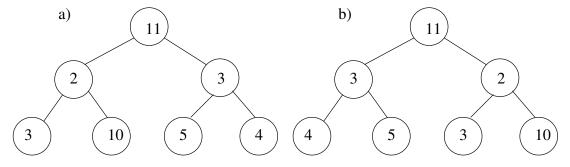
```
else{
      n->freq =
                                                          ; // Cercar on posar el node
      node_llista* aux =
      while(
                                           ) {
      }
      if (aux != n->seg){ // Cal moure el node
         // Desenganxar el node
         if (n ==
                                                 ){
         }
         \verb"else" \{
         }
         // Enganxar el node
                                                   ){
         if (aux ==
         }
         \verb"else" \{
    }
  {\tt return}
}
```

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);
    ...
  public:
    ...
};
```

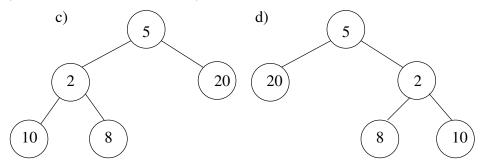
Els valors dels nodes representen pesos. Volem escorar l'arbre de tal manera que tot subarbre esquerre sempre pesi menys que el subarbre dret corresponent, intercanviant-los si cal. En cas que els dos subarbres pesin el mateix, els intercanviarem si el subarbre esquerre té més nodes que el dret. Noteu que a l'arbre hi haurà els mateixos nodes abans i després, però col·locats de manera diferent, i també hi ha els mateixos camins abans i després però disposats de manera diferent. Per exemple:

a) escorat es converteix en b):

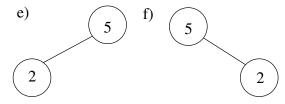


Continua a l'altra cara.

c) escorat es converteix en d):



e) escorat es converteix en f):



Implementeu el mètode demanat i l'auxiliar que es troba al següent full.

No fer servir operacions públiques de la classe Arbre.

SOLUCIÓ:

Posar el nom i contestar al full de resposta.

```
// Pre: cert
// Post: la jerarquia de nodes a partir d'n està escorada
// pes =
// mida =
static void escora_rec(node_arbre* n, int & pes, int & mida){
  if (
                           ) { // Cas base
  }
  else {
    int
    if (
                                                          ){
      node_arbre* aux;
    }
  }
}
// Pre: p.i. = A
// Post: el p.i. és com A escorat
void escora() {
                       ;
}
```