UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Estructuras de datos. Curso 2021-2022
Convocatoria extraordinaria de Febrero.
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. (1 punto) **(1)** Supongamos que hacemos las 3 siguientes afirmaciones:

   (a) En un esquema de hashing puede usarse como función hash **h(x)=M - (x% M)** con M primo.

   (b) Si insertamos un conjunto de **enteros ordenados** en un **AVL** y en un **APO**, la altura de ambos es la misma.

   (c) El orden en que las hojas se listan en los recorridos preorden, inorden y postorden de un **ABB** cambia dependiendo del recorrido

   1-a: Las tres son ciertas **1-b:** Dos son ciertas y una falsa **1-c:** Dos son falsas y una cierta; **1-d:** Las tres son falsas. **Razonar la respuesta.**

   (2) Si inserto las claves {17, 9, 13, 25, 11, 20, 32, 3} en un AVL de enteros, **2-a:** Hay que hacer dos rotaciones simples y una rotación doble **2-b:** Hay que hacer una rotación simple y una rotación doble, **2-c:** Hay que hacer dos rotaciones dobles y una rotación simple **2-d:** Todo lo anterior es falso. **Mostrar el árbol final**

   (3) Dados los siguientes recorridos en **preorden** = (Z,M,P,S,W,Q,L,R), y **postorden** = (S,W,P,Q,M,R,L,Z) **3-a:** No hay ningún árbol binario con esos recorridos asociados; **3-b:** Hay 1 solo árbol binario con esos recorridos asociados: **3-c:** Hay dos árboles binarios con esos recorridos asociados ; **3-d:** Todo lo anterior es falso. **Razonar la respuesta.**

2. (1 punto) Tenemos un contenedor de pares de elementos, **{string, queue<int>}** definido como:

```
class contenedor {
    private:
       map<string, queue<int> > datos;
    ……..
    ……..
}
```

   Implementar una clase **iterator** (constructor, *, ==, !=, ++) que itere sobre los string de longitud 5 y para los que la queue<int> tenga todos sus enteros pares. Además de los métodos **begin**() y **end**() de la clase contenedor.

3. (1 punto) Implementar una función

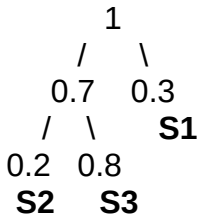
   vector<list<int> > secuencia_ascendente(const list<int> &L)

   que dada una lista de enteros L, encuentre (y devuelva en un vector de listas) todas las secuencias ascendentes que contenga. Una secuencia ascendente es una sublista de enteros consecutivos **ai, ai+1, ..., ai+k** de modo tal que **aj <= aj+1** para **j = i, ..., i+k −1**, y sea de la mayor longitud posible.
   Por ejemplo:
   si **n = 10** y la lista es **L =[0,5,5,9,4,3,9,6,5,5,2]**, entonces hay **6** secuencias ascendentes, a saber, **[0,5,5,9], [4], [3,9], [6], [5,5] y [2]**.

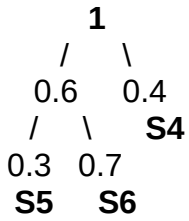
4. (1 punto) Un **árbol de probabilidades** es un árbol binario donde cada nodo tiene asociada una etiqueta con un valor real en el intervalo [0,1]. Cada nodo que no es hoja cumple la propiedad de que la suma de los valores de las etiquetas de sus hijos es 1. Un **suceso** es una hoja y la probabilidad de que ocurra viene determinada por el producto de los valores de las etiquetas de los nodos que se encuentran en el camino que parte de la raíz y acaba en dicha hoja. Se dice que **un suceso es probable si la probabilidad de que ocurra es mayor que 0.5**. Implementar (sin usar iteradores) una función que devuelva true si existe algún suceso probable en el árbol de probabilidades. Su prototipo será :

bool probable (const bintree<float> & A)

Ejemplo:



True: Suceso probable: S3



False

5. (1 punto) Implementar una función

bool contieneparejas(vector< set<int> > &sw, int n);

que devuelva true si cada par de enteros (j; k) con 0 <= j < k; k < n está contenido en al menos uno de los conjunto en sw[].

P.ej si: sw[0] = {0; 1; 2; 3; 4}; sw[1] = {0; 1; 5; 6; 7}; sw[2] = {2; 3; 4; 5; 6; 7} **contieneparejas(sw,8) debe devolver true.**

Por otra parte si tenemos sw[0] = {0; 2; 3; 4}; sw[1] = {0; 1; 5; 7}; sw[2] = {2; 3; 5; 6; 7} entonces los pares (0; 6), (1; 2), (1; 3), (1; 4), (1; 6), (4; 5), (4; 6) y (4; 7) no están en ningún conjunto y **contieneparejas(sw,8) debe devolver false.**

6. (1 punto) Insertar (detallando los pasos) las claves

{8, 16, 12, 41, 10, 62, 27, 65, 13}

en una **Tabla Hash cerrada** de tamaño 13. Resolver las colisiones usando **hashing doble**.

Construir (detallando los pasos) un **APO** con las claves anteriores. Borrar dos elementos en el APO resultante.

Tiempo: 2.30 horas

Soluciones al examen de Estructuras de Datos de 16 de Febrero de 2022

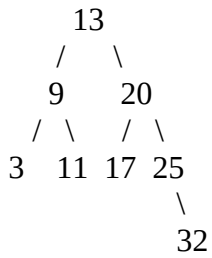
Pregunta 1

1.- **1.a es falsa**. Nunca se accede a la posición 0 y se sale del rango [0,M-1] si x%M=0

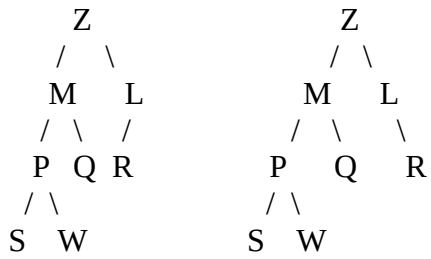
1.b es verdadera por la propia construcción de ambos cuando se insertan elementos ordenados

1.c es falsa. las hojas en los 3 recorridos se listan de izquierda a derecha por lo que su orden relativo coincide.
La respuesta correcta es por tanto 1.c

2.- **La respuesta correcta es la 2.d. Hay que hacer dos rotaciones dobles**. El árbol final es:



3.- **Hay 2 árboles binarios con esos recorridos**



No hay forma de distinguir con pre/post si R es un hijo a la derecha o a la izquierda de L. Hay por tanto 2 árboles binarios con esos recorridos. **La respuesta correcta es 3.c**

Pregunta 2

```
#include <iostream>
#include <map>
#include <vector>
#include <queue>
#include <string>

using namespace std;

template <class T>
class contenedor {
private:
   //Una pequeña modificacion para que se inicialice la cola con otro contenedor.
   // Esto es para implementar el constructor mas facilmente y no altera el ejercicio
   map<string,queue<int, T > >datos;
public:
   //Construyo
   contenedor(){
       vector<int>  l1={1,3,5};
       queue<int,vector<int> > q1(l1);
       vector<int>  l2={6,7,8};
       queue<int,vector<int> > q2(l2);
       vector<int>  l3={9,11,13,15};
       queue<int,vector<int> > q3(l3);

       datos.emplace("pera",q1);
       datos.emplace(string("naranja"),q2);
       datos.emplace(string("coco"),q3);
   }

   class iterator{
   private:
       typename map<string,queue<int, T > >::iterator it,final;

       bool condicion(const pair<const string,queue<int, T> > &d){
          if (d.first.size()!=5) return false;
          queue<int,T> aux(d.second);
          while (!aux.empty()){
            if (aux.front()%2!=0 ) return false;
            aux.pop();
          }
          return true;
       }
   public:
      iterator(){}
      bool operator==(const iterator &i)const{
         return (i.it==it );
      }
      bool operator!=(const iterator &i)const{
         return !(*this==i);
      }
```

```
        pair<const string,queue<int, vector<int> > > & operator*(){
            return *it;
        }
        iterator & operator ++(){
            do{
                ++it;
            }while (it!=final && !condicion(*it));
            return *this;

        }
        friend class contenedor;
    };

    iterator begin(){
     iterator i;
     i.it=datos.begin();
     i.final=datos.end();
     if (i.it!=datos.end()){
        if (!i.condicion(*i)) ++i;
     }
     return i;
    }
    iterator end(){
     iterator i;
     i.it=datos.end();
     i.final=datos.end();
     return i;
    }
};


int main(){

 contenedor<vector<int> > micontainer;
 contenedor<vector<int> >::iterator i;

}
```

Pregunta 3

```
#include <list>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

template <typename T>
void Imprimir(T comienzo,T fin){
   for (auto it =comienzo; it!=fin; ++it){
      cout<<*it<<" ";
   }
}
/**
 @brief Obtiene las secuencias ascendentes de una secuencia de elementos
*/
vector<list<int> > secuencia_ascendente (const list<int> &L){
    vector<list<int>>v;
    auto it=L.begin();
    while( it!=L.end()){
        auto it2=it;
        auto it3=it2;
        ++it3;
        bool seguir=true;
        while ( it3!=L.end()&& seguir){
            if ((*it2)<=(*it3)){
                ++it2;
                ++it3;
            }
            else seguir=false;
        }
        list<int>aux(it,it3);
        v.push_back(aux);
        it=it3;
    }
    return v;
}
int main(){
  list<int> L={0,5,5,9,4,3,9,6,5,5,2};
  vector<list<int> > v=secuencia_ascendente (L);
  cout<<endl;  cout<<"L:";
  Imprimir(L.begin(),L.end());
  cout<<endl;
  cout<<"Secuencias ascendentes encontradas :";
  auto im_vl=[](list<int> l)->void{
      cout<<"(";
      Imprimir(l.begin(),l.end());
      cout<<") ";
  };
  for_each(v.begin(),v.end(),im_vl);
}
```

Pregunta 4

```
#include <iostream>
#include "bintree.h"
using namespace std;

// para comprobar si tenemos un arbol de probabilidades

bool check_rep_node (const bintree<float>::node &n) {
   if (!n.left().null() && !n.right().null()) {
      if (((*n.left()) + (*n.right())) == 1.0) {
         return check_rep_node(n.left());
         return check_rep_node(n.right());
      }
      else
         return false;
   }
   else // el nodo es una hoja
      return true;
}

bool check_rep (const bintree<float> &A) {
   return check_rep_node(A.root());
}


// para comprobar si tenemos sucesos probables en el arbol

float probable_nodo (const bintree<float>::node &n) {
   if (n.left().null() && n.right().null())
      return *n;

   else {
      float v_i = probable_nodo(n.left());
      if ((*n * v_i) > 0.5)
         return *n * v_i;

      float v_d = probable_nodo(n.right());
      if ((*n * v_d) > 0.5)
         return *n * v_d;

      return 0;
   }
}

bool probable (const bintree<float> &A) {
   return probable_nodo (A.root()) >= 0.5;
}
```

```
int main(){
  // Creamos el árbol:
  //          1.0
  //         /    \
  //        0.7   0.3
  //        /  \
  //      0.2  0.8


  bintree<float> Arb(1.0);
  Arb.insert_left(Arb.root(), 0.7);
  Arb.insert_right(Arb.root(), 0.3);
  Arb.insert_right(Arb.root().left(), 0.8);
  Arb.insert_left(Arb.root().left(), 0.2);

  if (probable(Arb))
    cout<<"Hay suceso probable "<<endl;
  else
    cout<<"No hay suceso probable "<<endl;
}
```

Pregunta 5

```
#include <iostream>
#include <set>
#include <vector>

using namespace std;

bool contieneparejas(vector< set<int> > &sw, int n){
    for (int i=0;i<n-1;i++)
     for (int j=i+1;j<n;j++){
      bool enc=false;
      for (auto it=sw.begin();it!=sw.end() && !enc; ++it){
        if (it->find(i)!=it->end() && it->find(j)!=it->end())
            enc=true;
      }
      if (!enc) return false;
    }
    return true;
}


int main(){
  vector<set<int>> sw(3,set<int>());
  sw[0] = {0, 1, 2, 3, 4}; sw[1] = {0, 1, 5, 6, 7};
  sw[2] = {2, 3, 4, 5, 6, 7};
  if (contieneparejas(sw,8)){
    cout<<endl<<"Si cumple la condicion "<<endl;
  }
  else cout<<endl<<"No cumple la condicion"<<endl;
}
```

Pregunta 6

6.a

k	8	16	12	41	10	62	27	65	13
h(k)	8	3	12	2	10	10	1	0	0
h_o(k)	9	6	2	9	11	8	6	11	3

h(k) =k%13
h_o(k)=1+k%11

h_i(k)=(h_i-1(k)+h_o(k))%13

Todas se insertan a la primera excepto:

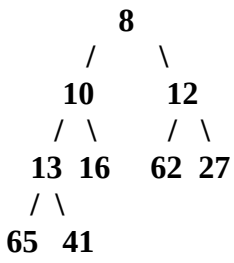
h_2(62) = (10 + 8) % 13 = 5

h_2(13)= (0 + 3) % 13 = 3 Colisión
h_3(13)= (3 + 3) % 13 = 6

Tabla resultante

0	1	2	3	4	5	6	7	8	9	10	11	12
65	27	41	16		62	13		8		10		12

6.b El APO que resulta es:



Tras hacer 2 borrados queda el APO: