UNIVERSIDAD DE GRANADA
Departamento de Ciencias de la Computación e Inteligencia Artificial

Estructuras de datos. Curso 2020-2021
Convocatoria extraordinaria de Febrero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. (1 punto) Se desea construir un **buscador** de productos dados por un código y el nombre del comercio en que se encuentran. Un producto puede estar en más de un comercio.
   • Dar una **representación** para el **TDA buscador** usando el tipo **map<int, set<string> >**
   • Implementar la función **insertar** que añade un producto dando el código junto con los comercios en que se puede encontrar.
   • Implementar una **función que obtenga** los comercios en que se encuentra un determinado producto.
   • Implementar la **clase iteradora** dentro de la clase buscador para poder iterar sobre todos los productos. Han de implementarse (aparte de las de la clase iteradora) las funciones **begin()** y **end()**.

2. (1.5 puntos) Implementar una función:
   bool nullsum(list<int> &L, list<int> &L2);

   que dada una lista L, devuelve true si hay un **rango de iteradores [p,q)**) tal que su suma de los elementos en el rango sea 0. En caso afirmativo debe retornar además el rango (uno de ellos, porque puede no ser único) a través de L2. En caso negativo L2 debe quedar vacía.

   Por ejemplo: si L=(-10, **14, -2, 7**, **-19**, -3, 2, 17, **-8, 8**) entonces debe retornar true y

   L2=(14, -2, 7, -19), ó bien L2=(-8, 8). En cambio:
   Si L=(1,3,-5) entonces debe retornar false y L2=().

3. (1 punto) Dado un bintree<int> T, implementar una función
   void prom_nivel(bintree<int> &T, list<float> &P);

   que genere una lista de reales P, donde el primer elemento de la lista sea el promedio de los nodos del árbol de nivel 0, el segundo sea el promedio de los de nivel 1, el tercero el promedio de los de nivel 2, y así sucesivamente. Es decir, que si el árbol tiene profundidad N, la lista tendrá N+1 elementos de tipo float.

4. (1.5 puntos) Implementar una función
   bool parejasinset(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 conjuntos 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}
   parejasinset(sw,8) debe devolver **true** porque todas las parejas (j,k) 0<= j < k < 8 están en alguno de los conjuntos de sw

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 parejasinset(sw,8) debe devolver **false.**

5. (1 punto) Determinar paso a paso las estructuras resultantes tras:

1-a **Insertar** las claves {31, 39, 43, 64, 33, 85, 50, 88, 36, 37} en una **Tabla Hash cerrada** de tamaño 13. A continuación **borrar** el 36 y el 64 y finalmente insertar el valor 74. Resolver las colisiones usando **rehashing doble**.
1-b **Insertar** las claves {24, 62, 75, 63, 85, 33, 50, 88, 36, 37, 46} en una **tabla hash abierta de tamaño 13,** resolviendo las colisiones usando árboles **AVL** en lugar de listas.

Tiempo: 2.30 horas

```
#include <map>
#include <iostream>
#include <string>
#include <set>
using namespace std;
class buscador{
  private:
    map<int,set<string> > productos;

  public:
      buscador(): {}
      void Insertar(const int & codigo,const set<string> &comercios){
            if (productos.find(codigo)==productos.end())
                productos[codigo]=comercios;
      }
      set<string> operator[](const int & codigo){
          auto it=productos.find(codigo);
          if (it!=productos.end())
              return it->second;
          else
             return set<string>();
      }
      class iterator {
       private:
         map<int,set<string> >::iterator it;
       public:
         iterator(){}
         bool operator==(const iterator &i)const{
            return i.it==it;
         }
         bool operator!=(const iterator &i)const{
            return i.it!=it;
         }
         pair<const int,set<string> > & operator*(){
             return *it;
         }
         iterator & operator++(){
            ++it;
            return *this;
         }
         friend class buscador;
      };//end class iterator
      iterator begin(){
        iterator i;
        i.it=productos.begin();
        return i;
      }
      iterator end(){
         iterator i;
         i.it=productos.end();
         return i;
      }
};
```

```
#include <list>
#include <iostream>
#include <vector>
#include <iterator> //distance
using namespace std;

template <typename T>
void Imprimir(T comienzo,T fin){
   for (auto it =comienzo; it!=fin; ++it){
      cout<<*it<<" ";
   }
}
vector<list<int>::const_iterator> search_anclas(const list<int> &L){
  vector<list<int>::const_iterator> positivos,negativos;
  vector<list<int>::const_iterator> zero;
  for (auto it =L.cbegin(); it!=L.cend(); ++it){
        if (*it<0) negativos.push_back(it);
        else if (*it>0) positivos.push_back(it);
        else{
            zero.push_back(it);
            return zero;
        }
  }
  if (negativos.size()>positivos.size())
     return positivos;
  else return negativos;
}
bool nullsum(const list<int> &L1, list<int> &L2){
    //buscamos si existe un 0;
    vector<list<int>::const_iterator> anclas=search_anclas(L1);
    if (anclas.size()==0) return false;
    //si tiene un cero
    if ((*(anclas[0]))==0){
        L2.push_back(0);
        return true;
    }
    //si todos son positivos o negativos
    if (anclas.size()==L1.size())
        return false;
    bool enc=false;
    //vamos recorriendo solamente o los positivos o negativos
    //dependiendo si el numero de negativos es menor que
    //el de positivos o viceversa
    for (auto it =anclas.begin(); !enc && it!=anclas.end();++it){
        //desde la ancla ira hacia la izquierda
        list<int>::const_reverse_iterator it2=L1.crend();
        //convertir iterator to reverse_iterator
        int p2=distance(*it,L1.begin())-1;
        advance(it2,p2);
        while (it2!=L1.rend() && !enc){
            list<int>::const_iterator it3=L1.cbegin();
            int d=distance(L1.crend(),it2);
            //convertir a const_iterator
            advance(it3,-d-1);
            int sum=0;
            list<int>aux;
            while (it3!=L1.end()){
                sum+=*it3;
                aux.push_back(*it3);
                if (sum==0){
                    enc=true;
                    L2=aux;
                }
                else ++it3;
```

```
            }
            ++it2;
        }
    }
    return enc;
}
int main(){
  list<int>L={-10, 14, -2, 7, -19, -3, 2, 17, -8, 8};
  //list<int>L={1,-3,5,0};
  list<int>L2;

  cout<<"L1:";
  Imprimir(L.begin(),L.end());
  cout<<endl;
  if (nullsum(L, L2)){
    cout<<"La secuencia que suma cero es:";
    Imprimir(L2.begin(),L2.end());
  }
  else cout<<"No existe secuencia que sume cero";
}
```

==========versión simplificada==========

```
bool nullsum(list<int> &L, list<int> &L2){
  list<int>::iterator it, it2;
  int suma = 0;
  bool encontrado = false;

  for (it=L.begin(); it!=L.end() && !encontrado; it++){
    suma = *it;
    it2=it;
    it2++;
    for ( ; it2!=L.end() && suma!=0; it2++)
      suma+=*it2;

    if (suma==0){
      encontrado = true;
      list<int> L3(it, it2);
      L2=L3;
    }
  }
  return encontrado;
}
```

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

template <typename T>
void Imprimir(T comienzo,T fin){
   for (auto it =comienzo; it!=fin; ++it){
      cout<<*it<<" ";
   }
}
void prom_nivel(bintree<int> &T, list<float> &P){
   typedef pair<bintree<int>::node, int> info;
   queue<info> micola;
   int level=0;
   info a(T.root(),0);
   micola.push(a);
   float suma=0;
   int cnt=0;
   while (!micola.empty()){
     a=micola.front();
     micola.pop();
     if (a.second==level){
        suma+=*(a.first);
        cnt++;
     }
     else{//pasamos a otro nivel
        P.push_back(suma/cnt);
        suma=*(a.first);
        cnt=1;
        level++;
     }
     bintree<int>::node n = a.first;
     info h;
     if (!n.left().null()){
        h.first = n.left();
        h.second=a.second+1;
        micola.push(h);
     }
     if (!n.right().null()){
        h.first = n.right();
        h.second=a.second+1;
        micola.push(h);
     }
   }
   P.push_back(suma/cnt);
}
int main(){

  // Creamos el árbol Arb1:
  //        1
  //     /    \
  //    4      7
  //  /  \    /  \
  // 8    9  11   10
//       /     \
        6       14
//             /  \
//            12  13          P={1, 5.5, 9.5, 10, 12.5}

bintree<int> Arb1(1);
Arb1.insert_left(Arb1.root(), 4);
```

```
Arb1.insert_right(Arb1.root(), 7);
Arb1.insert_left(Arb1.root().left(), 8);
Arb1.insert_right(Arb1.root().left(), 9);
Arb1.insert_left(Arb1.root().right(), 11);
Arb1.insert_right(Arb1.root().right(),10);
Arb1.insert_left(Arb1.root().left().right(), 6);
Arb1.insert_right(Arb1.root().right().left(), 14);
Arb1.insert_left(Arb1.root().right().left().right(), 12);
Arb1.insert_right(Arb1.root().right().left().right(), 13);

  list<float>out;
  prom_nivel(Arb1,out);
  cout<<"La lista con los promedios por nivel..."<<endl;
  Imprimir(out.begin(),out.end());
}
```

```
#include <iostream>
#include <iostream>
#include <set>
#include <vector>
using namespace std;


bool parejasinset(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 (parejasinset(sw,8)){
    cout<<endl<<"Si cumple la condicion "<<endl;
  }
  else cout<<endl<<"No cumple la condicion"<<endl;
}
```

(a)

k	31	39	43	64	33	85	50	88	36	37
h(k)	5	0	4	12	7	7	11	10	10	11
h_o(k)	10	7	11	10	1	9	7	1	4	5

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

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

Se insertan a la primera 31, 39,43,64,33

Clave 85: colisión:

h_2(85)=3

50 y 88 se insertan a la primera:

clave 36: colisión

h_2(36)= 1

Clave 37: colisión

h_2(24)= 3
h_3(24)= 8

Tabla resultante

0	1	2	3	4	5	6	7	8	9	10	11	12
39	36		85	43	31		33	37		88	50	64

Eliminamos 36 y 64, e insertamos 74 que tiene un valor hash: h(74) =74%13=9

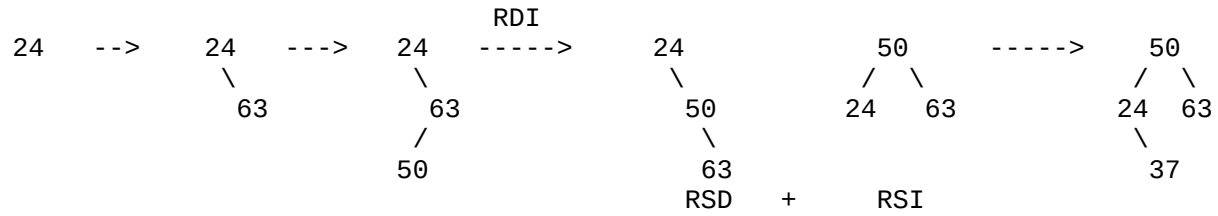
Tabla resultante

0	1	2	3	4	5	6	7	8	9	10	11	12
39			85	43	31		33	37	**74**	88	50	

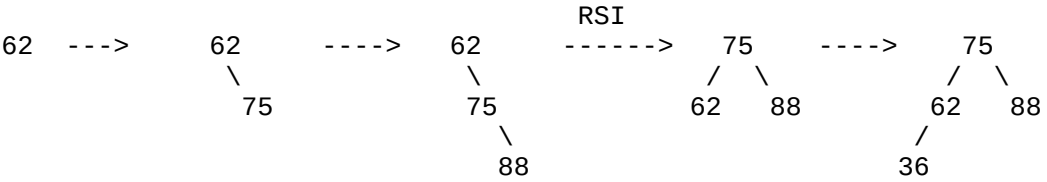
(b)

k	24	62	75	63	85	33	50	88	36	37	46
h(k)	11	10	10	11	7	7	11	10	10	11	7

24, 63, 50 y 37 van a la cubeta 11



62, 75, 88 y 36 van a la cubeta 10



85, 33 y 46 van a la cubeta 7