UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Estructuras de datos. Curso 2018-2019
Convocatoria de Enero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. (0.5 puntos) Razonar la verdad o falsedad de las siguientes afirmaciones:

   (a) La definición **priority_queue<int>**::iterator p; es correcta.

   (b) Dado un árbol binario cuyas etiquetas están organizadas como un **AVL**, puedo recuperarlo de forma unívoca a partir de su recorrido en inorden.

   (c) El elemento de valor máximo en un **ABB<int>** se encuentra en el nodo más profundo.

   (d) Considerar un **map <int, int> M** en el que hacemos **M[3]=7;** Supongamos ahora que hacemos

   map <int, int> :: iterator p = M. find(3);

   p → second =9;

   Tras hacer eso, el valor de M[3] sigue siendo 7

   (e) Si A es una **tabla hash cerrada** con un 50% de elementos vacíos y un 40% de elementos borrados y B una tabla hash cerrada con un 50% de elementos vacíos y sin elementos borrados, A y B son igual de eficientes cara a la búsqueda de un elemento.

2. (1.5 puntos) Implementa una **clase documento** a partir de una lista de palabras que componen un texto. Esta clase guardaría de forma eficiente las posiciones en las que se encuentran cada una de las palabras que forman dicho texto.

   a) Indica una representación e implementa el constructor

      documento::documento(const list<string> &texto)

   b) Implementa una función que devuelva de forma eficiente las posiciones en las que aparece una determinada palabra.

      set<int> documento::posiciones(string palabra) const;

   c) Implementa una función que calcule la palabra que se encuentra en la posición iésima

      string documento::palabra(int i) const;

3. (1 punto) Dada una lista de enteros L y 2 listas seq y reemp, implementa una función:

   void reemplaza (list<int> & l, const list<int> &seq, const list<int> &reemp);

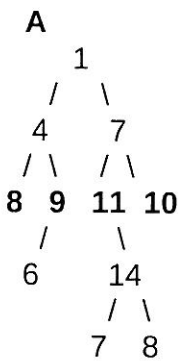
   que busque todas las secuencias seq en l y las reemplace por reemp.
   P.ej si L={1,2,3,**4,5,1**,2,3,**4,5,1**,2,3,4,5} , seq= {4,5,1} y reemp ={9,7,3, 5}
   L quedaría como L={1,2,3,**9,7,3, 5**,2,3,**9,7,3, 5**,2,3,4,5}

4. (1 punto) Implementa una función

   list<int> nivel (const bintree<int> & A);

   que dado un árbol binario A, devuelva una lista con las etiquetas del nivel que tenga un mayor número de nodos

Ejemplo:



L={8,9,11,10}

En caso de que haya más de una lista solución, basta con devolver una de ellas

5. (1 punto) Detalla cada una de las operaciones siguientes:
   a) Insertar las claves {4, 12, 16, 37, 6, 58, 23, 61, 9, 10} en una **Tabla Hash cerrada** de tamaño 13. A continuación borrar el 10 y el 37 y finalmente insertar el valor 48. Resolver las colisiones usando **rehashing doble.**
   b) Construir un **AVL** insertando, en ese orden, las siguientes claves {99, 28, 70, 81, 47, 38, 100, 21, 16, 49, 57}, especificando los pasos seguidos e indicando cuando sea necesario el tipo de rotación que se usa para equilibrar.
   c) Construir un **árbol parcialmente ordenado** realizando las siguientes tareas:
      - Inserta, en ese orden, las siguientes claves {10, 5, 12, 12, 5, 3, 9, 4, 3}
      - Borrar tres elementos.

6. (1 punto) Tenemos un contenedor de pares de elementos, {clave, list<int>} definida como:

```
template <typename T>
class contenedor {
    private:
      unordered_map<T, list<int> > datos;
    ..............
    ..............
}
```

Implementa un **iterador** que itere sobre los elementos que cumplan la propiedad de que la lista asociada a una clave solo contenga números pares. Se debe implementar (aparte de las de la clase iteradora) las funciones begin() y end().

Tiempo: 3 horas

2. (1.5 puntos) Implementa una **clase documento** a partir de una lista de palabras que componen un texto. Esta clase guardaría de forma eficiente las posiciones en las que se encuentran cada una de las palabras que forman dicho texto.

   a) Indica una representación e implementa el constructor

      documento::documento(const list<string> &texto)

   b) Implementa una función que devuelva de forma eficiente las posiciones en las que aparece una determinada palabra.

      set<int> documento::posiciones(string palabra) const;

   c) Implementa una función que calcule la palabra que se encuentra en la posición iésima

      string documento::palabra(int i) const;

```
nuevo.second.in{
 private:
   map<string, set<int>> palabras;

   documento::documento(const list<string> & texto){
        list<string>::const_iterator it;
        map<string,set<int>>::iterator mit;
        int contador = 0;
        for(it = texto.cbegin(); it != texto.cend(); ++it){
          mit = palabras.find(*it);
         if(mit == palabras.end()){
           pair <string,set<int>> nuevo;
           nuevo.first = *it;
           nuevo.second.insert(contador);
           palabras.insert(nuevo);
         }
         else{
           (*mit).second.insert(contador);
         }
         contador++;
        }
   }
 public:
}
```

3. (1 punto) Dada una lista de enteros L y 2 listas seq y reemp, implementa una función:

void reemplaza (list<int> & l, const list<int> &seq, const list<int> &reemp);

que busque todas las secuencias seq en l y las reemplace por reemp.

P.ej si L={1,2,3,**4,5,1**,2,3,**4,5,1**,2,3,4,5} , seq= {4,5,1} y reemp ={9,7,3, 5}

L quedaría como L={1,2,3,**9,7,3, 5**,2,3,**9,7,3, 5**,2,3,4,5}

```
reemp.beza (list<int> & l, const list<int> &seq, const
list<int> &reemp){

list<int>::iterator inicio, fin;
const list <int>::const_iterator aux = seq.cbegin();
inicio = l.begin();
fin = l.begin();
while (fin != l.end()){
  if(aux != seq.end() && *fin == *aux){
     ++fin;
     ++aux;
  }
  else if(aux == seq.end()){
     aux = seq.cbegin();
     erase(inicio,fin);
     insert(reemp.cbegin(),reemp.cend());
     inicio = fin;
  }
  else if(*fin != *aux){
     aux = seq.cbegin();
     fin++;
     inicio = fin;
  }
}
}
```

Ejercicio de Test
a)False. En una cola por prioridad se puede acceder solamente al elemento mas prioritario
b)False. Ej:

c)False. Se encuentra en la hoja mas a la derecha
d) False. El valor de M se modifica a 9
e) Fase. Ya que si se una casilla esta borrada debemos seguir buscando en la tabla hash por si hubiese existido
colision

```cpp
//Ejercicio 2
#include <iostream>
#include <map>
#include <set>
#include <list>
#include <vector>
using namespace std;

class Documento{
  private:
       map <string,set<int> > palabras;
  public:
       Documento(const list<string> & texto){
            list<string>::const_iterator it;
            map <string,set<int> >::iterator mit;
            int pos=0;
            for (it=texto.cbegin(); it!=texto.cend();++it,++pos){
                    mit = palabras.find(*it);
                    if (mit==palabras.end()){
                         set<int>aux;
                         aux.insert(pos);
                         pair<string,set<int>> p(*it,aux);
                         palabras.insert(p);
                    }
                    else {
                         mit->second.insert(pos);
                    }
            }
       }
       set<int> posiciones(string & palabra){
          map<string,set<int> >::iterator mit;
          mit= palabras.find(palabra);
          if (mit!=palabras.end()){
            return mit->second;
          }
       }
       string palabra(int i)const {
             map<string,set<int> >::const_iterator mit;
             mit = palabras.cbegin();
             for (;mit!=palabras.cend();++mit){
                set<int>::const_iterator sit=(*mit).second.find(i);
                if (sit!=(*mit).second.cend()){
                   return (*mit).first;
                }

             }
             return "";
       }
};


int main(){
   string v[]={"El", "dia","de","mañana","sera", "un" ,
   "buen","momento-","Todo","el","dia","fue","soleado"};
   list<string>milist;
   milist.assign(v,v+13);
   Documento D(milist);
   string ss="dia";
   cout<<"Posiciones de "<<ss<<":";

   set<int>aux=D.posiciones(ss);
   set<int>::iterator pit;
   for (pit=aux.begin();pit!=aux.end();++pit){
      cout<<*pit<<endl;
   }

   cout<<"La palabra en la posicion 5 es "<<D.palabra(5)<<endl;
}
```

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

void Imprimir (list<int> & l){
    list<int>::iterator it;
   for (it=l.begin();it!=l.end();++it){
       cout<<*it<<" ";
   }
   cout<<endl;
}

void reemplaza(list<int> &l, const list<int> &seq,const list<int> &reemp){
  list<int>::iterator lit,it_find;
  list<int>::const_iterator seq_it,reem_it;
   lit=l.begin();
   while (lit!=l.end()){
      it_find=lit;
      seq_it=seq.cbegin();

      while (it_find !=l.end() && seq_it!=seq.cend() &&  *it_find == *seq_it){
         ++it_find;
         ++seq_it;
      }
      if (seq_it==seq.cend()){
         lit= l.erase(lit,it_find);
         reem_it=reemp.cbegin();
         while (reem_it!=reemp.cend()){
            lit=l.insert(lit,*reem_it);
            ++lit;
            ++reem_it;
         }
      }
      else ++lit;
   }
}




int main(){
   int v1[]={1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
   int v2[]={4,5,1};
   int v3[]={9,7,3,5};
   list<int>l,l2,l3;
   l.assign(v1,v1+15);
   l2.assign(v2,v2+3);
   l3.assign(v3,v3+4);
   cout<<"Reemplazando ...."<<endl;
   reemplaza(l,l2,l3);
   Imprimir(l);
}
```

```
#include "arbolbinario.h"
#include <queue>
#include <list>
#include<iostream>
using namespace std;
//sin usar iteradores
void Imprimir (list<int> & l){
    list<int>::iterator it;
   for (it=l.begin();it!=l.end();++it){
       cout<<*it<<" ";
   }
   cout<<endl;
}

list<int> Maximo(ArbolBinario<int> &ab){
    queue<pair<ArbolBinario<int>::nodo,int> > q;
    ArbolBinario<int>::nodo n = ab.getRaiz();
    pair<ArbolBinario<int>::nodo,int> p(n,0);

    int level =0;
    list<int>laux;
    list<int>lout;

    q.push(p);

    while (!q.empty()){
      p= q.front();

      if (p.second==level){
         laux.push_back(*(p.first));

      }
      else{


         if (laux.size()>lout.size()){
             lout=laux;
             level++;
         }
         laux.clear();
         laux.push_back(*(p.first));
      }
      q.pop();
      n=p.first;
      int aux_level =p.second-1;
      if (!n.hi().nulo()){
       p.first=n.hi();
       //cout<<"Insertado "<<*(n.hi())<<endl;
       p.second=aux_level;
       q.push(p);
      }
      if (!n.hd().nulo()){
         p.first=n.hd();

         p.second=aux_level;
         q.push(p);
      }

    }
    if (laux.size()>lout.size()){
             lout=laux;


   }

    return lout;

}
int main(){
      ArbolBinario<int> a;

 // ej:n1n4n8xxn9n6xxxn7n11xn14n7xxn8xxn10xx


 cout<<"Introduce un arbol:";

 cin>>a;
 cout<<endl<<"El arbol insertado: "<<endl;
 cout<<a<<endl;
 list<int> l=Maximo(a);
 cout<<"El level de mayor longitud:";
 Imprimir(l);
}
```

5. a)

{4, 12, 16, 37, 6, 58, 23, 61, 9, 10}

$h(k) = k \% 13$

$h_i(k) = [h_{i-1}(k) + h_0(k)] \% 13$ $\quad h_0(k) = 1 + k \% (M-2)$

$h_0(k) = 1 + k \% 11$

k	4	12	16	37	6	58	23	61	9	10	48
h(k)	4	12	3	11	6	6	10	9	9	10	9
$h_0(k)$	5	2	6	5	7	4	2	7	10	11	5

$h(4) = 4 \% 13 = 4$; $h(12) = 12 \% 13 = 12$; $h(16) = 16 \% 13 = 3$

$h(37) = 37 \% 13 = 11$ $\quad h(6) = 6 \% 13 = 6$

$h(58) = 58 \% 13 = 6$ coliziune

$h_2(58) = [h(58) + h_0(58)] \% 13 = 10 \% 13 = 10$ $\quad h_2(58) = 10$

($h(58)$: 6, $h_0(58)$: 4)

$h(23) = 23 \% 13 = 10$ coliziune

$h_2(23) = [h(23) + h_0(23)] \% 13 = 12 \% 13 = 12$ coliziune

($h(23)$: 10, $h_0(23)$: 2)

$h_3(23) = [h_2(23) + h_0(23)] \% 13 = 1 \rightarrow h_3(23) = 1$

($h_2(23)$: 12, $h_0(23)$: 2)

$h(61) = 61 \% 13 = 9$

$h(9) = 9 \% 13 = 9$ coliziune

$h_2(9) = [h(9) + h_0(9)] \% 13 = 6$ coliziune

($h(9)$: 9, $h_0(9)$: 10)

$h_3(9) = [h_2(9) + h_0(9)] \% 13 = 3$ coliziune

($h_2(9)$: 6, $h_0(9)$: 10)

$h_4(9) = [h_3(9) + h_0(9)] \% 13 = 0$

($h_3(9)$: 3, $h_0(9)$: 10)

$h_4(9) = 0$

$h(10) = 10 \% 13 = 10$ coliziune

$h_2(10) = [h_0(10) + h_0(10)] \% 13 = 8$

($h_0(10)$: 10, $h_0(10)$: 11)

$h_2(10) = 8$

$h(48) = 48 \% 13 = 9$ coliziune

$h_2(48) = [h(48) + h_0(48)] \% 13 = 1$ coliziune

($h(48)$: 9, $h_0(48)$: 5)

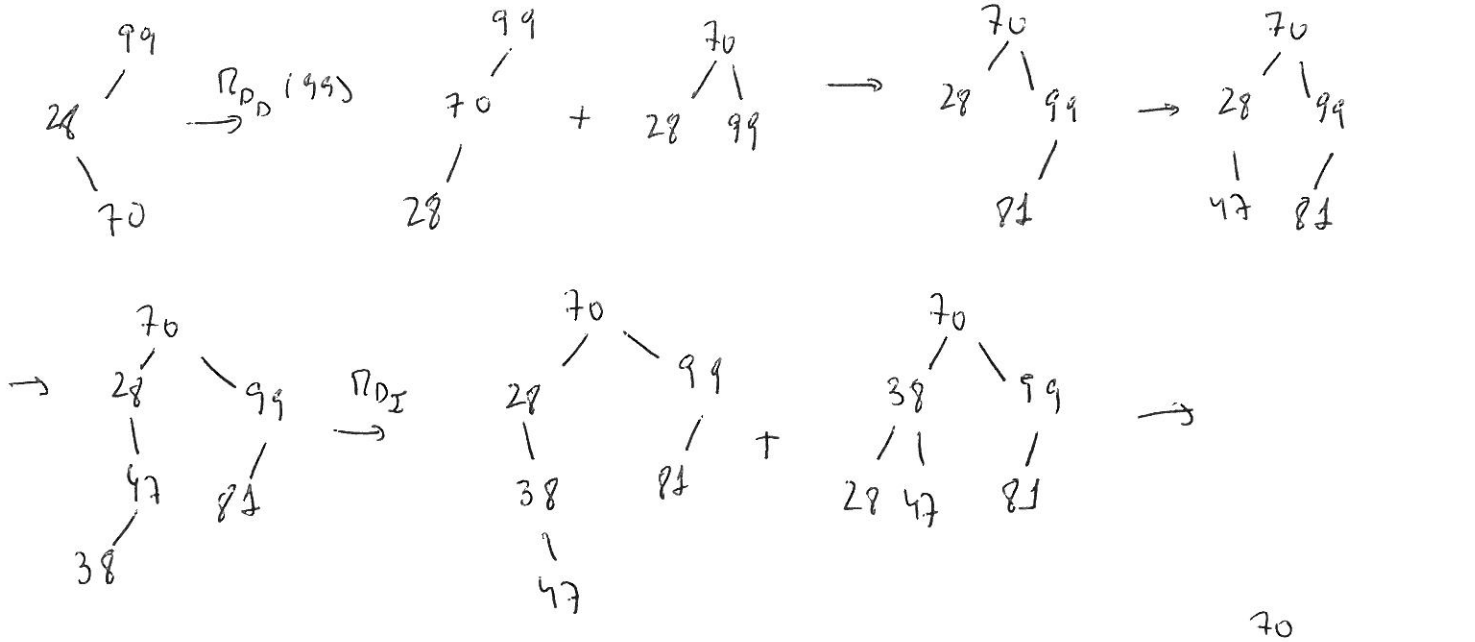
$h_3(48) = [1 + 5] \% 13 = 6$ coliziune

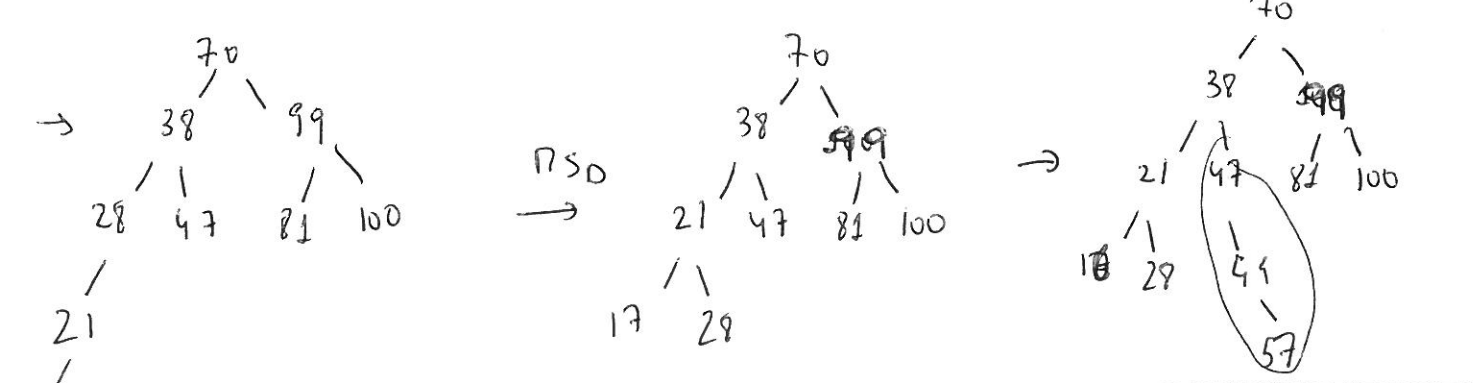
$h_4(48) = [6 + 5] \% 13 = 11$

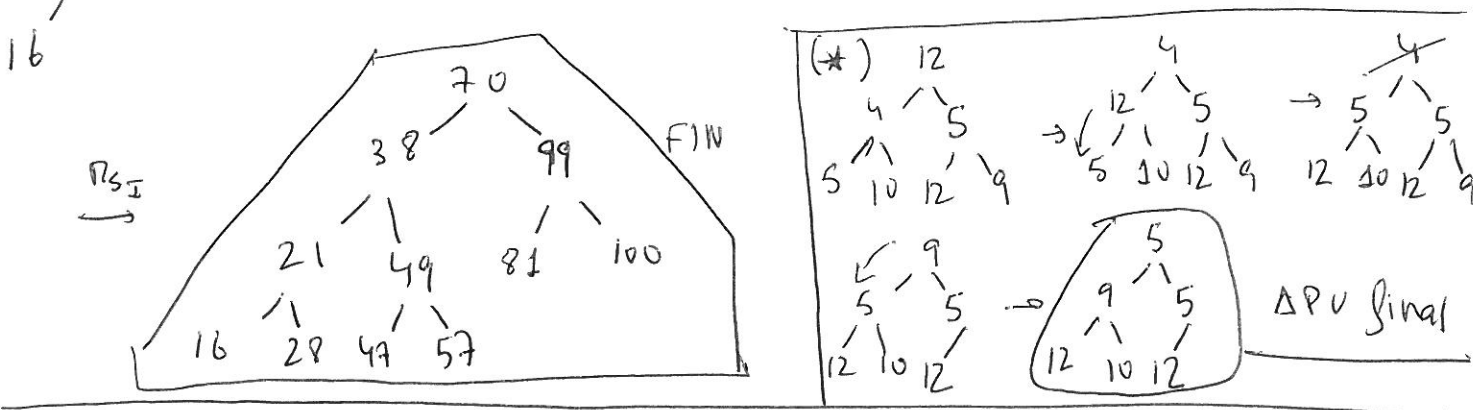
$h_4(48) = 11$

0	9
1	23
2	
3	16
4	4
5	
6	6
7	
8	~~10~~
9	61
10	58
11	~~37~~ (48)
12	12

5.5 99, 28, 70, 81, 47, 38, 100, 21, 16, 49, 57

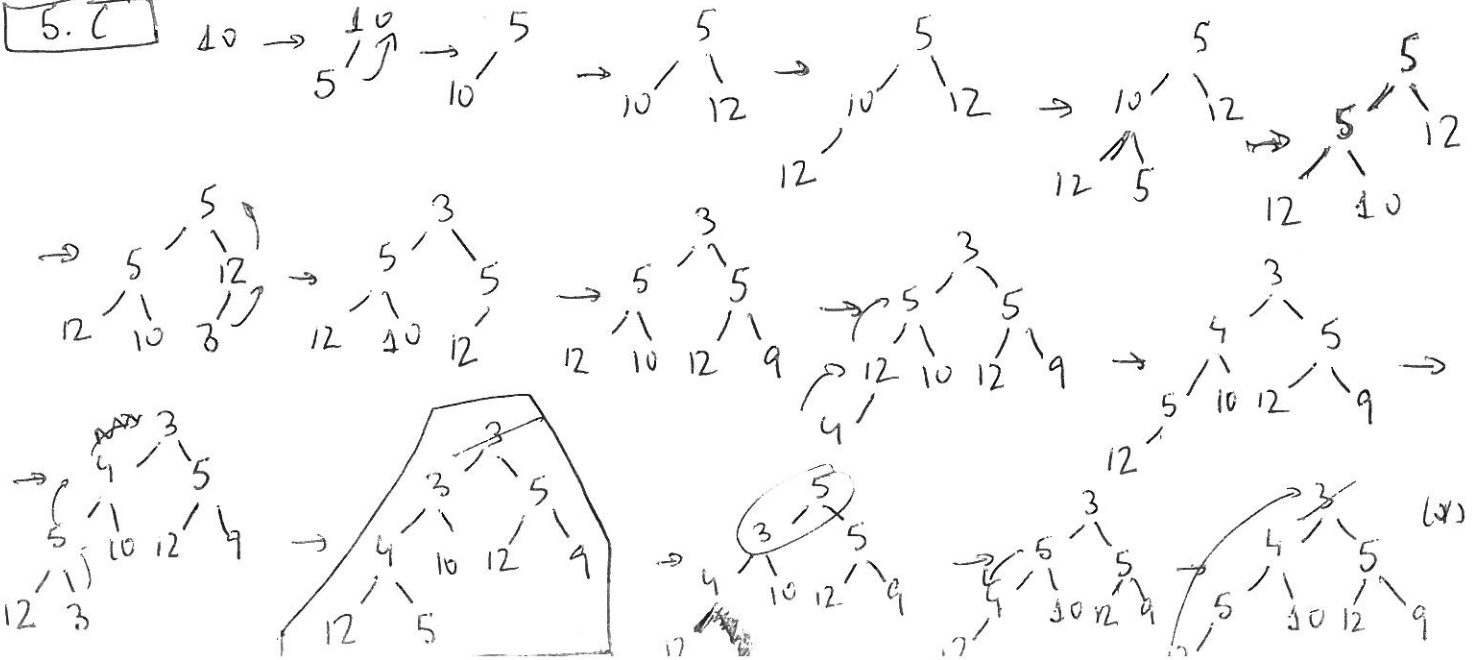






(*)

5.6



```
#include <unordered_map>
#include <list>
#include<iostream>
using namespace std;
//sin usar iteradores
void Imprimir (list<int> & l){
    list<int>::iterator it;
   for (it=l.begin();it!=l.end();++it){
       cout<<*it<<" ";
   }
   cout<<endl;
}
bool AllPares(const list<int> & l){
   list<int>::const_iterator it;
   for (it=l.cbegin(); it!=l.cend();++it)
      if (*it%2==1) return false;
   return true;
}


template <typename T>
class contenedora{
  private:
   unordered_map <T, list<int> > datos;
  public:


    class iterator{
     private:
      typename unordered_map <T, list<int> >::iterator it;
      typename unordered_map <T, list<int> >::iterator f;;
     public:
      iterator (){}

      bool operator==(const iterator &i)const{
         return i.it==it;
      }
      bool operator!=(const iterator &i)const{
         return i.i!=it;
      }
      pair<T,list<int> > & operator*(){
         return *it;
      }
      iterator &operator++(){

         do{
            ++it;

         }while (it!=f && !AllPares(*it.second));
         return *this;
      }
      friend class contenedora;
   };

   iterator begin(){
      iterator i;
      i.it=datos.begin();
      i.f = datos.end();
      if (!AllPares((*(i.it)).second))
         ++i;
      return i;
   }
   iterator end(){
      iterator i;
      i.it=datos.end();
      i.f = datos.end();
      return i;
   }
};


int main(){
       contenedora<string> c;
       contenedora<string>::iterator i;
}
```