



PruebaAlien

www.wuolah.com/student/PruebaAlien



sol1718.pdf

Examenes



2º Estructuras de Datos



Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



Exámenes, preguntas, apuntes.

12:48

WUOLAH

Join the student revolution.

MULTI

Conéctate dónde y cómo prefieras.

Guarda tus apuntes en un lugar seguro y ordenado, y accede a ellos desde tu pc, móvil o tablet.

Acceder

Registrarse

GET IT ON
Google Play

Download on the
App Store

Estructuras de Datos
Curso 2017-2018. Convocatoria de Enero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas

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

- (a) El orden en que las hojas se listan en los recorridos preorden, inorden y postorden de un árbol binario es el mismo en los tres casos. V *Siempre & listan de izda a dcha*
- (b) Dado un árbol binario cuyas etiquetas están organizadas como un árbol binario de búsqueda, puedo recuperarlo a partir de su preorden. V *pueder ser DBB*
- (c) Dado un árbol binario cuyas etiquetas están organizadas como un árbol parcialmente ordenado, puedo recuperarlo a partir de su postorden. V *pueder ser DPO*
- (d) Es correcto en un esquema de hashing cerrado el uso como función hash de la función $h(k) = [k + \text{random}(M)] \% M$, M primo y con $\text{random}(M)$ una función que devuelve un número entero aleatorio entre 0 y $M-1$. F *con random(M) no podríamos recuperar la clave*
- (e) Es correcto en un esquema de hashing doble el uso como función hash secundaria de la función $h_0(x) = [(B-1)-(x \% B)] \% B$ con B primo. F *h0 no puede anularse*

2. (1.5 puntos) Supongamos que tenemos una clase **Liga** que almacena los resultados de enfrentamientos en una liga de baloncesto:

```

struct enfrentamiento{
    unsigned char eq1,eq2; // códigos de los equipos enfrentados
    unsigned int puntos_eq1, puntos_eq2; //puntos por cada equipo
};

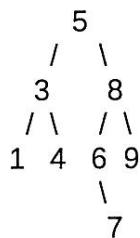
class liga{
    private:
        list<enfrentamiento> res;
        ...
};


```

- Implementa un método que dado un código de equipo obtenga el número de enfrentamientos que ha ganado.
- Implementa la clase iterator dentro de la clase **Liga** que permita recorrer los enfrentamientos en los que el resultado ha sido el empate. Implementar los métodos **begin()** y **end()**.

3. (1 punto) Implementa una función **int orden (const list<int> &L)**; que devuelva 1 si L está ordenada de forma ascendente de principio a fin, 2 si lo está de forma descendente y 0 si no está ordenada de ninguna forma.

4. (1 punto) Dado un árbol binario de búsqueda, implementa una función para imprimir las etiquetas de los nodos **en orden de mayor a menor profundidad**. Si tienen la misma profundidad pueden aparecer en cualquier orden. Ejemplo:



El resultado sería **7,1,4,6,9,3,8,5**.

5. (1 punto) Tenemos un contenedor de pares de elementos, `{clave, bintree<int>}` definida como:

```

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

Implementa un **iterador** que itere sobre los elementos que cumplan la propiedad de que la suma de los elementos del `bintree<int>` sea un número par. Debes implementar (aparte de las de la clase iterator) las funciones `begin()` y `end()`.

6. (1 punto) Un "heap-doble" es una estructura jerárquica que tiene como propiedad fundamental que para cualquier nodo Z a profundidad **par** la clave almacenada en Z es **menor** que la del padre pero **mayor** que la del abuelo (cuando existen), y para cualquier nodo Z a profundidad **impar**, la clave almacenada en Z es **mayor** que la del padre pero **menor** que la del abuelo (cuando existen), siendo el árbol binario y estando las hojas empujadas a la izquierda. Diseña una función para insertar un nuevo nodo en la estructura y aplicarla a la construcción de un heap-doble con las claves {30, 25, 12, 16, 10, 15, 5, 18, 23, 32, 4, 17}.

Tiempo: 3 horas

```

#include <iostream>
#include <list>
using namespace std;
struct enfrentamiento{
    unsigned char eq1,eq2;
    unsigned int puntos_eq1,puntos_eq2;
};
class liga{
private:
    list<enfrentamiento>res;
public:
    //...
    int Ganados(unsigned char code){
        list<enfrentamiento>::iterator i;
        int n=0;
        for (i=res.begin();i!=res.end();++i){
            if ((*i).eq1==code && (*i).puntos_eq1>(*i).puntos_eq2) n++;
            else
                if ((*i).eq2==code && (*i).puntos_eq2>(*i).puntos_eq1) n++;
        }
        return n;
    }
    class iterator {
private:
    list<enfrentamiento>::iterator it,final;
public:
    iterator(){}
    bool operator==(const iterator &i)const{
        return i.it==it;
    }
    bool operator!=(const iterator &i)const{
        return i.it!=it;
    }
    enfrentamiento& operator* (){
        return (*it);
    }
    iterator & operator++(){
        ++it;
        bool salir =false;
        while (it!=final && !salir){
            if ((*it).puntos_eq1==(*it).puntos_eq2) salir=true;
            else ++it;
        }
        return *this;
    }
    friend class liga;
};
iterator begin(){
    iterator i;
    i.it=res.begin();
    i.final = res.end();
    if (!(*i.it)).puntos_eq1==(*i.it)).puntos_eq2) ++i;
    return i;
}
iterator end(){
    iterator i;
    i.it=res.end();
    i.final = res.end();
    return i;
}
;

```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

18 | 16/1/18 | ej3.cpp | 1

```

    #include <iostream>
    #include <list>
    using namespace std;
    int orden(const list<int> &L){
        list<int>::const_iterator it1, it2;
        it1=L.cbegin();
        if (it1==L.cend()){
            it2=it1;
            ++it2;
            unsigned int sentido;
            if (*it1<=*it2){
                sentido =1;
            }
            else
                sentido =2;
            ++it1; ++it2;
            while (it2!=L.cend()){
                if (sentido==1 && !(*it1<=*it2))
                    return 0;
                else if (sentido ==2 && !(*it2<=*it1))
                    return 0;
                ++it1; ++it2;
            }
            return sentido;
        }
        return 0;
    }
    int orden (list<int> L)
    {
        bool es_ascendente=true;
        es_descendente=false;
        list<int>:: iterator it1 = L.begin();
        list<int>:: iterator it2 = L.begin();
        if (L.empty())
            return 1;
        else
            ++it2;
        while ((es_ascendente || es_descendente) && it2!=L.end())
        {
            if ((*it1) < (*it2))
                es_descendente=false;
            else if ((*it1) > (*it2))
                es_ascendente=false;
            ++it1;
            ++it2;
        }
        if (es_ascendente)
            return 1;
        else if (es_descendente)
            return 2;
        else
            return 0;
    }
    }
    
```

```
#include "arbolbinario.h"
#include <queue>
#include <stack>
#include<iostream>
using namespace std;
//sin usar iteradores

void ImprimeProfundidad(ArbolBinario<int> &ab){
    queue<ArbolBinario<int>::nodo> q;
    ArbolBinario<int>::nodo n = ab.getRaiz();
    q.push(n);
    stack<ArbolBinario<int>::nodo> p;
    while (!q.empty()){
        n= q.front();
        q.pop();
        p.push(n); //ponemos en la pila
        //ponemos los hijos en la cola
        q.push(n.hd());
        q.push(n.hi());
    }
    while (!p.empty()){
        n=p.top();
        cout<<*n<<" ";
        p.pop();
    }
}
```

```

#include <unordered_map>
#include <arbolbinario.h>
using namespace std;
int Suma (ArbolBinario<int>::nodo n){
    if (n.nulo())
        return 0;
    else
        return (*n)+Suma(n.hi())+Suma(n.hd());
}

template <typename T>
class contenedor{
private:
    unordered_map<T,ArbolBinario<int> > datos;

public:
    //...
    class iterator{
private:
    typename unordered_map<T,ArbolBinario<int> >::iterator it,final;
public:
    iterator(){}
    bool operator==(const iterator &i) const{
        return i.it==it;
    }
    bool operator!=(const iterator &i) const{
        return i.it!=it;
    }
    pair<const T, ArbolBinario<int> > & operator* (){
        return (*it);
    }
    iterator & operator++(){
        ++it;
        bool salir =false;
        while (it!=final && !salir){
            if (suma((*it).second.getRaiz())%2==0)
                salir =true;
            else ++it;
        }
        return *this;
    }
    friend class contenedor;
};

iterator begin(){
    iterator i;
    i.it=datos.begin();
    i.final = datos.end();
    if (suma(*(i.it)).second.getRaiz()%2==1)
        ++i;
    return i;
}
iterator end(){
    iterator i;
    i.it=datos.end();
    i.final = datos.end();
    return i;
}
;

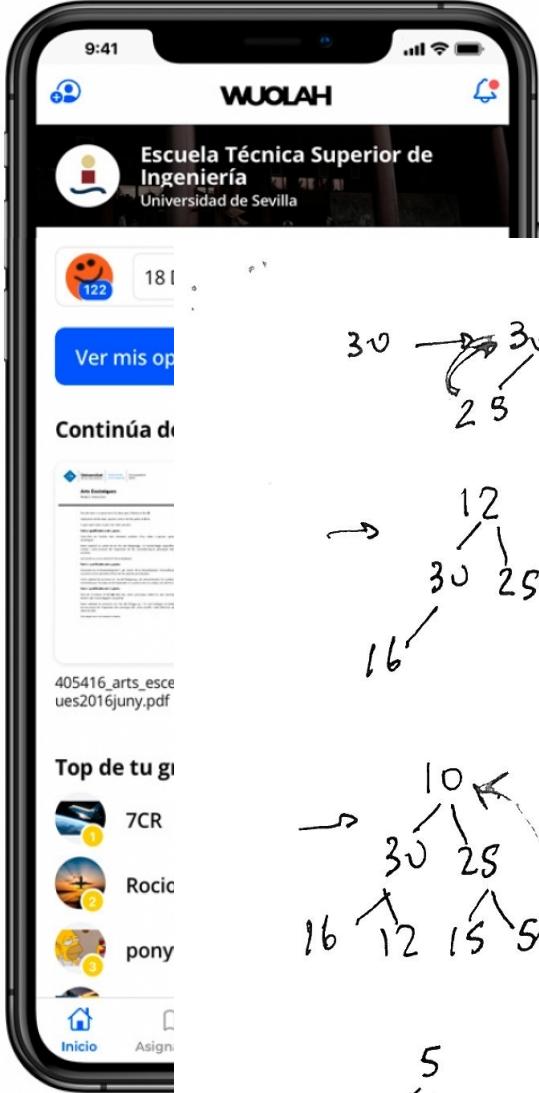
```

```

#include <vector>
#include <iostream>
#include <cmath>
using namespace std;
class heap_doble{
private:
    vector <int> datos;

public:
    void insertar(int clave){
        int pos = datos.size();
        datos.push_back(clave);
        int padre = (pos-1)/2;
        int abuelo = (padre-1)/2;
        int profundidad =(log(pos+1)/log(2)); anotación de la s)
        bool colocado =false;
        while (pos>0 && !colocado){
            if (padre>0){ //dos condiciones tiene parente y abuelo
                if (profundidad%2==0){
                    if ( datos[padre]>datos[pos] && datos[abuelo]<datos[pos])
                        colocado=true;
                    else{
                        if (datos[padre]<datos[pos]){
                            swap(datos[padre],datos[pos]);
                            pos = padre;
                            padre =abuelo;
                            abuelo = (padre-1)/2;
                        }
                        else if (datos[abuelo]>datos[pos]){
                            swap(datos[abuelo],datos[pos]);
                            pos = abuelo;
                            padre =(pos-1)/2;
                            abuelo = (padre-1)/2;
                        }
                    }
                }
                else{ //profundidad impar
                    if ( datos[padre]<datos[pos] && datos[abuelo]>datos[pos])
                        colocado=true;
                    else{
                        if (datos[padre]>datos[pos]){
                            swap(datos[padre],datos[pos]);
                            pos = padre;
                            padre =abuelo;
                            abuelo = (padre-1)/2;
                        }
                        else if (datos[abuelo]<datos[pos]){
                            swap(datos[abuelo],datos[pos]);
                            pos = abuelo;
                            padre =(pos-1)/2;
                            abuelo = (padre-1)/2;
                        }
                    }
                }
            }
            else{ //solamente tiene parente
                //la profundidad debería ser 1
                if ( datos[padre]<datos[pos] )
                    colocado=true;
                else{
                    swap(datos[padre],datos[pos]);
                    pos = padre; colocado =true;
                    //ya se sale porque pos es 0
                }
            }
            profundidad--;
        }
    }
}

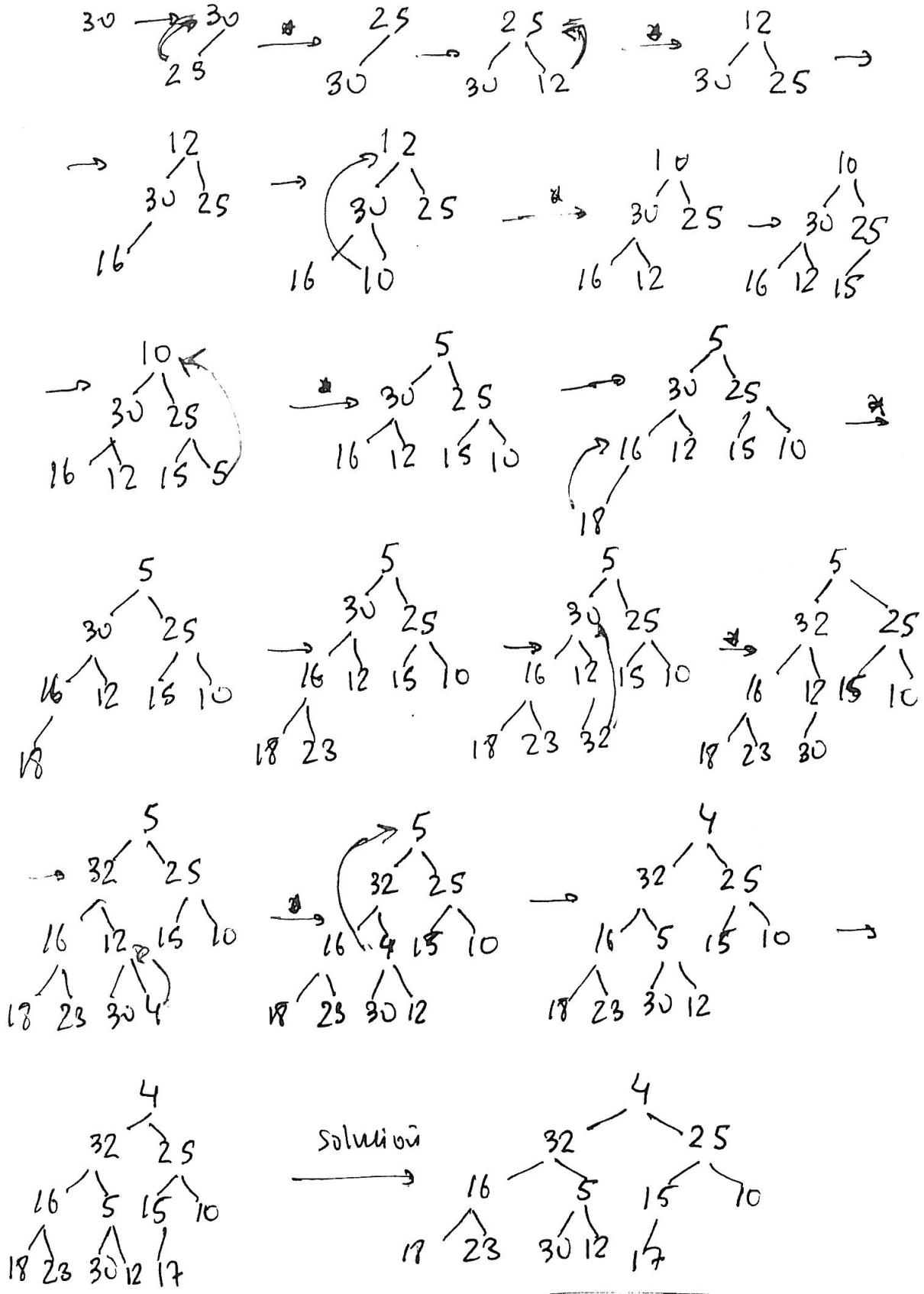
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play





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

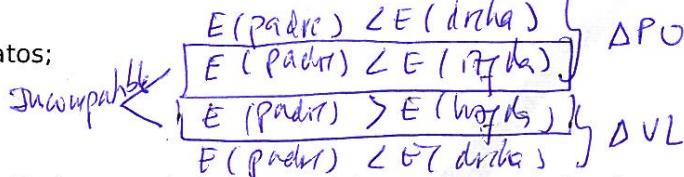
- (a) El TDA **cola con prioridad** se puede implementar de forma óptima usando un heap. **V**
- (b) La declaración **map<list<int>, string> m;** es una declaración válida. **V**
- (c) Un **AVL** puede reconstruirse de forma única dado su recorrido en **inorden** **F**
- (d) Es imposible que un árbol binario (con más de dos nodos) sea **AVL** y **APÓ** a la vez. **V**
- (e) En un esquema de **hashing doble** nunca puede ocurrir que para dos claves k_1 y k_2 distintas coincidan simultáneamente sus valores h (función hash primaria) y h_0 (función hash secundaria). Es decir, con $k_1 \neq k_2$, $h(k_1) = h(k_2)$ y $h_0(k_1) = h_0(k_2)$ **F**

2. (1.5 puntos) Dada una clase **libro** que almacena las palabras que contiene y la posición (1..n) en que está cada palabra en el libro:

```
class libro{
    private:
        struct palabra {
            string pal; // palabra
            unsigned int posición; //posición en la que está la palabra
        };
        list datos;
        ...
};
```

$$h(K) = K \% 7 \quad h_0(K) = 1 + K \% 5$$

K	1	70	35
$h(K)$	0	0	
$h_0(K)$	1	4	



- Implementar un método que dada una palabra obtenga todas las posiciones en las que aparece. **list<int> libro::posiciones(const string &pal)**
- Implementar una clase iterator dentro de la clase **libro** que permita recorrer las palabras que comiencen por **z** y estén en una posición par. Implementar los métodos **begin_z_par()** y **end_z_par()**.

3. (1 punto) Supongamos que disponemos de un **map<string, list<pair<int,int> > >** que contiene un conjunto de palabras de un libro y asociada a cada palabra una lista **list<pair<int,int> >** dónde cada par contiene un número de capítulo y una posición dentro del mismo dónde aparece dicha palabra.

Se pide construir un vector **vector<list<string> >** dónde **v[i-1]** contenga todas las palabras del capítulo **i** ordenadas alfabéticamente y sin repeticiones.

P.ej. si tenemos el map:

```

<casa, {<1,10>,<2,10>,<3,40>}>
<ventana, {<1,2>,<1,20>,<3,30>}>
el vector contendrá:
v[0]={casa,ventana,ventana} -----> v[0]={casa,ventana}
v[1]={casa}
v[2]={casa,ventana}
  
```



4. (1 punto) Implementa una función para determinar si un árbol binario tiene **más de un camino** desde una hoja a la raíz cuya suma de etiquetas sea igual a k.

```
bool suma_k(const bintree<int> &arb, int k)
```

5. (1 punto) Tenemos un contenedor de pares de elementos, `{clave, bintree<int>}` definido como:

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

Implementar un iterador que itere sobre los string de longitud 4 y para los que el `bintree<int>` tenga una estructura de árbol binario de búsqueda.

6. (1 punto) Un **APO sesgado** es un árbol binario que tiene como propiedad fundamental el que para cualquier nodo Z la clave almacenada en Z es **menor** que la del hijo izquierda de Z y esta a su vez **menor** que la del hijo derecha (cuando existen), y estando las hojas empujadas a la izquierda. Implementar una función para insertar un nuevo nodo en la estructura y aplicarlo a la construcción de un APO sesgado con las claves {29, 24, 11, 15, 9, 14, 4, 17, 22, 31, 3, 16}.

Tiempo: 3 horas

```

#include <iostream>
#include <list>
#include <string>
using namespace std;
class libro{
private:
    struct palabra{
        string pal;
        unsigned int posicion;
    };
    listdatos;
public:
    //...
    list<int> posiciones(const string &pal){
        list<int>out;
        list::iterator i;
        for (i=datos.begin();i!=datos.end();++i){
            if ((*i).pal==pal) out.push_back((*i).posicion);
        }
        return out;
    }
    class iterator {
private:
    list::iterator it,final;
public:
    iterator(){}
    bool operator==(const iterator &i) const{
        return i.it==it;
    }
    bool operator!=(const iterator &i) const{
        return i.it!=it;
    }
    pair<string,int> operator* (){
        return pair<string,int>((*it).pal,(*it).posicion);
    }
    iterator & operator++(){
        ++it;
        bool salir =false;
        while (it!=final && !salir){
            if (((*it).pal[0]=='z' || (*it).pal[0]=='Z') && (*it).posicion&2==0)
                salir=true;
            else ++it;
        }
        return *this;
    }
    friend class libro;
};
iterator begin_z_par(){
    iterator i;
    i.it=datos.begin();
    i.final = datos.end();
    if (!((i.it)).pal[0]=='z' || ((i.it)).pal[0]=='Z' && ((i.it)).posicion&2==0)
        ++i;
    return i;
}
iterator end_z_par(){
    iterator i;
    i.it=datos.end();
    i.final = datos.end();
    return i;
};
}

```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



```

#include <iostream>
#include <vector>
#include <map>
#include <list>
using namespace std;
unsigned int Dime_Max_Capitulo(const map<string,list<pair<int,int> > > & m){
    map<string,list<pair<int,int> > >::const_iterator mit;
    list<pair<int,int> >::const_iterator it;
    unsigned int maxcap=0;
    for (mit=m.cbegin();mit!=m.cend(); ++mit){
        for (it=mit->second.begin(); it!=mit->second.end();++it){
            if (maxcap<(*it).first)
                maxcap = (*it).first;
        }
    }
    return maxcap;
}

vector<list<string> > ObtenPalabrasCap(const map<string,list<pair<int,int> > > & m){
    unsigned int maxchap=Dime_Max_Capitulo(m);
    vector<list<string> > v(maxchap);
    map<string,list<pair<int,int> > >::const_iterator mit;
    list<pair<int,int> >::const_iterator it;

    for (mit=m.cbegin();mit!=m.cend(); ++mit){
        for (it=mit->second.cbegin(); it!=mit->second.cend();++it){
            v[(*it).first-1].push_back((*mit).first);
        }
    }
    //Ahora eliminamos los repetidos del vector
    for (unsigned int i=0;i<v.size();i++)
        v[i].unique();
    return v;
}

int main(){
    map<string,list<pair<int,int> > > mimap;
    list<pair<int,int> > laux;
    laux.push_back(pair<int,int>(1,10));
    laux.push_back(pair<int,int>(2,10));
    laux.push_back(pair<int,int>(3,40));
    mimap.insert(pair<string,list<pair<int,int> >>("casa",laux));
    laux.clear();

    laux.push_back(pair<int,int>(1,2));
    laux.push_back(pair<int,int>(1,20));
    laux.push_back(pair<int,int>(3,30));
    mimap.insert(pair<string,list<pair<int,int> >>("ventana",laux));
    laux.clear();
    vector<list<string> >vaux=ObtenPalabrasCap(mimap);
    for (int i=0;i<vaux.size();i++){
        list<string>::iterator it;
        cout<<"v["<<i<<"]=";
        for (it=vaux[i].begin();it!=vaux[i].end();++it){
            cout<<(*it)<< ",";
        }
        cout<<endl;
    }
}

```

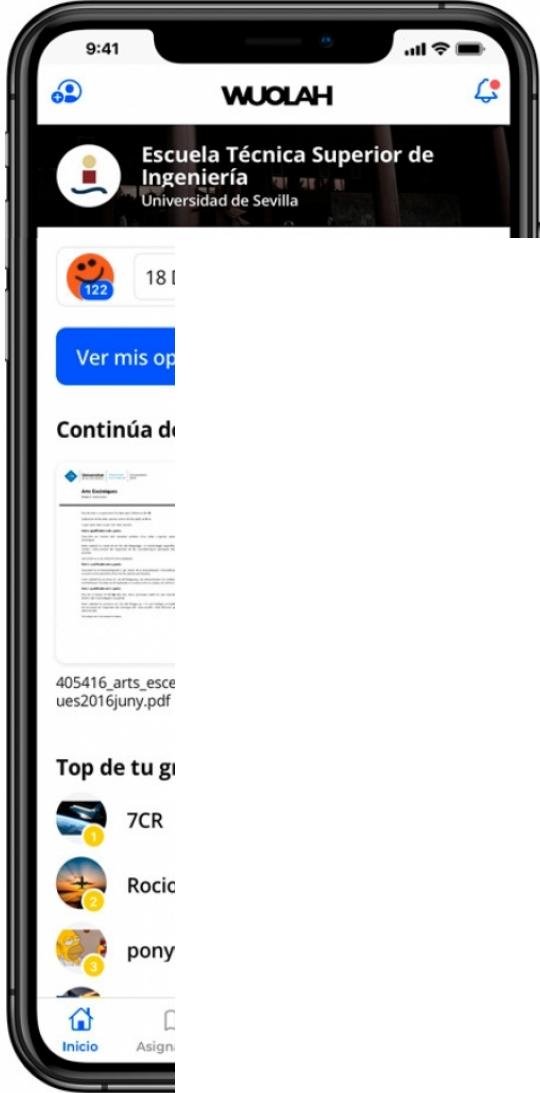
WUOLAH

Después de los finales, ¡RELÁJATE CON GUATAFAC!

```
#include "arbolbinario.h"
#include <queue>
#include <stack>
#include<iostream>
using namespace std;
//sin usar iteradores
int caminos_sumak(ArbolBinario<int>::nodo n,int s,int k){
    if (!n.nulo() && s<=k){
        if (n.hi().nulo() && n.hd().nulo()){ //es hoja
            if (s+(*n)==k){
                return 1;
            }
            else{
                return 0;
            }
        }
        else{
            if (s<k){
                return caminos_sumak(n.hi(),s+(*n),k)+caminos_sumak(n.hd(),s+(*n),k);
            }
            else {
                return 0;
            }
        }
    }
    else return 0;
}

bool suma_k(ArbolBinario<int> &arb,int k){
    if (caminos_sumak(arb.getRaiz(),0,k)>1) return true;
    else return false;
}

int main(){
    ArbolBinario<int> a;
    // ej:n1n2n4xxn8xxn3n6xxn7xx se corresponde con el arbol
    // 1
    //   | -2
    //   |   | -4
    //   |   | -8
    //   |   | -3
    //   |   | -6
    //   |   | -7
    cout<<"Introduce un arbol:";
    cin>>a;
    cout<<endl<<"El arbol insertado: "<<endl;
    cout<<a<<endl;
    if (suma_k(a,11))
        cout<<"Si tiene varios caminos que suman 11"<<endl;
    else
        cout<<"No tiene varios caminos"<<endl;
}
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

```

#include <map>
#include <arbolbinario.h>
#include <limits>
using namespace std;
bool esABB (ArbolBinario<int>::nodo n,int min,int max){
    if (n.nulo())
        return true;
    else{
        if ((*n<min || *n>max)) return false;
        if (!n.hi().nulo())
            if (*n<*(n.hi())) return false;
        if (!n.hd().nulo())
            if (*n>*(n.hd())) return false;
        return esABB(n.hi(),min,*n) && esABB(n.hd(),*n,max);
    }
}

class contenedor{
private:
    map<string,ArbolBinario<int> > datos;

public:
    //...
    class iterator{
private:
    map<string,ArbolBinario<int> >::iterator it,final;
public:
    iterator(){}
    bool operator==(const iterator &i)const{
        return i.it==it;
    }
    bool operator!=(const iterator &i)const{
        return i.it!=it;
    }
    pair<const string, ArbolBinario<int> > & operator* (){
        return (*it);
    }
    iterator & operator++(){
        ++it;
        bool salir =false;
        while (it!=final && !salir){
            if ((*it).first.size()==4 &&
                esABB((*it).second.getRaiz(),numeric_limits<int>::min(),numeric_limits<int>::max()))
                salir =true;
            else ++it;
        }
        return *this;
    }
    friend class contenedor;
};

iterator begin(){
    iterator i;
    i.it=datos.begin();
    i.final = datos.end();
    if (!((i.it)).first.size()==4 &&
        esABB((*(i.it)).second.getRaiz(),numeric_limits<int>::min(),numeric_limits<int>::max()) )
        ++i;
    return i;
}
iterator end(){
    iterator i;
    i.it=datos.end();
    i.final = datos.end();
    return i;
}

```

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
#include <vector>
#include <iostream>
#include <cmath>
using namespace std;
class APO_sesgado{
private:
    vector <int> datos;
public:
    void insertar(int clave){
        int pos = datos.size();
        datos.push_back(clave);
        int padre = (pos-1)/2;
        bool colocado =false;
        int herm_drcha;
        while (pos>0 && !colocado){

            bool izquierda= (padre*2+1==pos);
            if (izquierda){ //si es el hijo a la izquierda
                if (pos<datos.size()-1)
                    herm_drcha=padre*2+2;
                if (datos[pos]>datos[herm_drcha])
                    swap(datos[herm_drcha],datos[pos]);
            }
            else{//es el hijo a la derecha
                int herm_izq=padre*2+1;
                if (datos[pos]<datos[herm_izq]){
                    swap(datos[herm_izq],datos[pos]);
                    pos = herm_izq;
                }
            }
            if (datos[pos]<datos[padre]){
                swap(datos[padre],datos[pos]);
                pos =padre;
                padre=(pos-1)/2;
            }
            else
                colocado=true;
        }
    }
};
```




Preguntas específicas de prácticas (4 puntos). Tiempo: 1 hora

1. (2 puntos) Supongamos que dos personas (usuarios) entran en un **laberinto** de habitaciones con las siguientes características:

- Cada habitación del laberinto tiene una puerta por donde se entra y dos puertas posibles por la que se sale.
- Hay habitaciones que conducen a la calle y por lo tanto finaliza el recorrido por el laberinto.
- Para pasar por una puerta se tira un dado y si sale un número par se sale por la puerta izquierda y si sale impar se sale por la derecha.
- A cada habitación solamente se accede por una única habitación.

Se pide:

- Establecer la representación para el laberinto e implementar el constructor de la clase laberinto. El laberinto se debe generar de forma aleatoria pasándole como parámetro el número de habitaciones.
- Implementar una función **int laberinto::quien_gana()** que determine si llega primero a la salida el usuario 1, el usuario 2 o si no llega ninguno de los dos (en cuyo caso devuelve 0).

2.- (2 puntos) Implementar una función que obtenga la **codificación Huffman** de un conjunto de caracteres. Esta codificación codifica cada carácter en función del número de veces que aparece, asignando códigos más cortos a caracteres que aparecen más veces. En la siguiente imagen se puede ver un ejemplo de funcionamiento. Los pasos para obtener el código Huffman de un conjunto de caracteres, con las frecuencias de aparición de cada carácter son:

Se definen las parejas: {frecuencia, carácter asociado}. Dichas parejas frecuencia-carácter se insertan en una cola con prioridad. La más prioritaria debe ser la pareja con menor frecuencia.

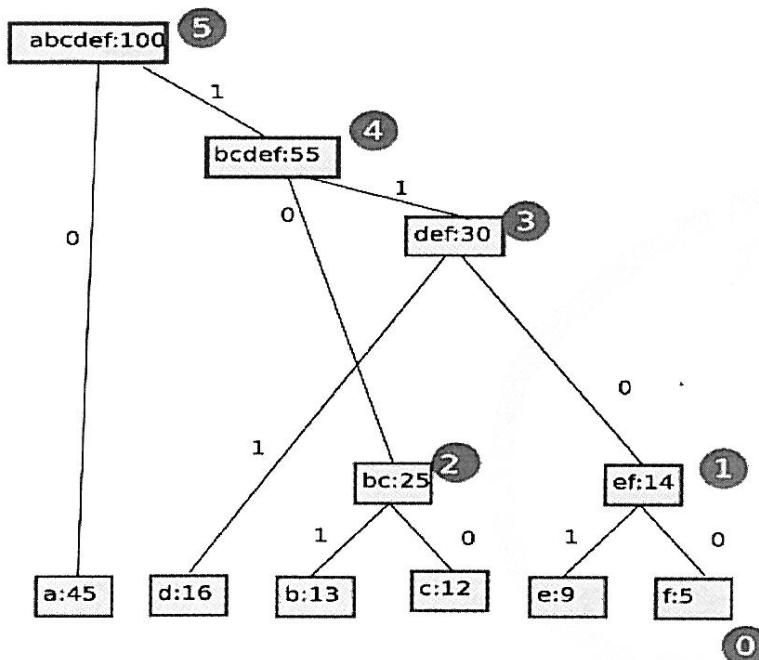
Se sacan los dos elementos con menor frecuencia, se suman sus frecuencias y se añaden a su codificación los valores 0 y 1. A continuación, se inserta un nuevo elemento en la cola con prioridad con frecuencia la suma de las frecuencias de los elementos, y con caracteres asociados la concatenación de los que sean sacado.

Si la cola sólo tiene un elemento, terminar, en otro caso volver a sacar las dos elementos con mayor prioridad y repetir los pasos anteriores.

Construir el árbol (o la estructura que se decida) que subyace a la codificación y realizar las siguientes tareas:

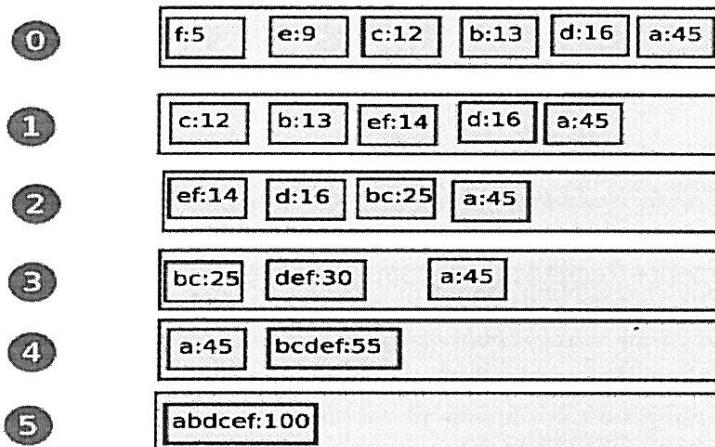
- Implementar una función que dada una cadena de caracteres devuelva el texto codificado.
- Implementar una función que decodifique una cadena (formada por 0s y 1s)

En la siguiente imagen se puede ver un ejemplo de codificación Huffman para los caracteres a, b, c, d, e, f con frecuencias 45,13,12,16,9,5 respectivamente. En el primer paso se introducen los caracteres en la cola con prioridad con sus frecuencias y con una codificación: "". A continuación, se realiza el proceso descrito anteriormente.



Paso

Cola de Prioridad



Códigos

a=""	c=""
d=""	e=""
b=""	f=""
a=""	c=""
d=""	e="1"
b=""	f="0"
a=""	c="0"
d=""	e="1"
b="1"	f="0"
a=""	c="0"
d="1"	e="01"
b="1"	f="00"
a=""	c="00"
d="11"	e="101"
b="01"	f="100"
a="0"	c="100"
d="111"	e="1101"
b="101"	f="1100"