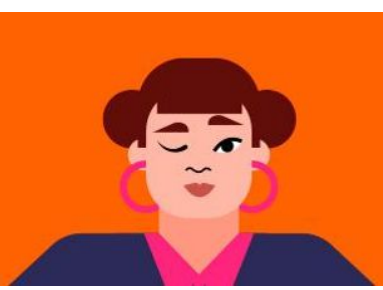
Esto no son apuntes pero **tiene un 10 asegurado** (y lo vas a disfrutar igual).

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Me interesa

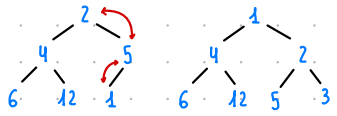


1/6

Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

1. (1 punto) **(a)** Si inserto las claves {2, 4, 5, 6, 12, 1, 3} en un **APO** de enteros, **(a1):** Hay que un solo intercambio padre-hijo **(a2):** Hay que hacer dos intercambios padre-hijo, **(a3):** Hay que hacer tres intercambios padre-hijo **(a4):** Todo lo anterior es falso. **Mostrar el árbol final**



R: a2, dos intercambios

(b) Dadas las siguientes 3 afirmaciones:

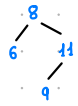
- Es correcto en un esquema de **hashing cerrado** el uso como función hash de:

  h(k)= [k + 2*k] % M, M primo

- La declaración **map<list<int>, string> m;** es una declaración válida.
- El elemento de valor máximo en un **ABB<int>** se encuentra en el nodo de más profundidad.

(b1) Todas son falsas **(b2)** Hay 2 ciertas y 1 falsa **(b3)** Hay 1 cierta y dos falsas **(b4)** Todas son ciertas

1. Verdadera, una función Hash tiene como requisito distribuir uniformemente las claves la función indicada equivale a 3k % M que al ser M un número primo cumple los requisitos.
2. Verdadero
3. Falso, ejemplo

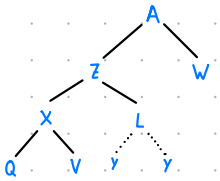


R: b2

(c) Dados los siguientes recorridos Preorden y Postorden:

Pre = {A,Z,X,Q,V,Y,L,W,T,R} **Post ={Q,V,X,Y,L,Z,T,R,W,A}**

(c1) Hay exactamente 2 árboles binarios con esos recorridos
(c2) No hay ningún árbol binario con esos recorridos
(c3) Hay exactamente 1 árbol binario con esos recorridos
(c4) Hay más de 2 árboles binarios con esos recorridos
Razona la respuesta



1.- como A está al principio del pre y al final del post → Raíz
2.- vemos h.izq y dercha de A → Z y W
3.- descendientes de Z y W
4.- h.izq y dercha de Z → X y L
5.- descendientes de X y L
6.- INCOPATIBILIDAD en el preorden nos indica que L es hijo de Y, al contrario que hemos visto en el postorden

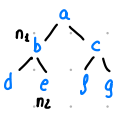
R: c2, no árbol posible

(d) Dados dos nodos n1 y n2 en un árbol binario T y dadas las distancias (longitudes de los caminos) m1 y m2 de ambos nodos a su antecesor común más cercano (nodo más profundo que tiene tanto a n1 como a n2 como descendientes) : (d1) Si m1=m2=1 los nodos son el mismo nodo (d2) Si m1=0 y m2>0: n2 es sucesor de n1 (d3) Si m1=m2=2 los nodos no son hermanos; (4) Todo lo anterior es cierto

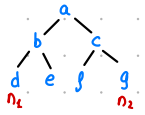
d1. falsa

para ser el mismo nodo → $m_1 \cdot m_2 = 0$

d2. verdadero



d3. Verdadero



Consulta condiciones aquí

do your thing

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

2. (1 punto) Supongamos que representamos una lista usando un vector de la siguiente forma:

```
class listacursores {
private:
 struct dato {  char elem;
                int siguiente; };
vector<dato> elementos;
int primero;
int nelems;
public:
 .......
};
```

donde el campo *elem* es el elemento de cada posición de la lista, y siguiente indica la posición dentro del vector en que está el siguiente elemento de la lista. Ejemplo: El vector:

0	1	2	3	4
a	4	d	5	b
3	e	1	c	2

con: **nelems=5**; **primero=0** representa la lista (en orden) **L: <a, c, b, e, d>**

Dada dicha representación de listas donde la posición de cada elemento viene determinada por un número entero, **construir una clase iteradora**, de forma que los elementos listados por el iterador deben aparecer en el orden en que están en la lista (independientemente de como estén almacenados en el vector). Para hacerlo correctamente, deben implementarse (constructor, *, ==, !=, ++) junto con las funciones **begin()** y **end()** de la clase listacursores.

```
class listacursores {
    private:
        :
        :
    public :
        class iterator {
            private :
                dato * it;
                dato * comienzo;

            public:
                iterator () { }
                bool operator == (const iterator &i) {
                    return i.it == it;
                }
                bool operator != (const iterator &i) {
                    return !(*this == i);
                }
                char &operator * () {
                    return it -> elem;
                }
                iterator & operator ++() {
                    int next = it -> siguiente;
                    it = comienzo + next;
                    return *this;
                }
                friend class listacursores;
        };
        iterator begin() {
            iterator i;
            i.it = &(datos[primero]);
            i.comienzo = &(datos[0]);
            return i;
        }
        iterator end () {
            iterator i;
            i.it = &(datos[nelems]);
            i.comienzo = &(datos[0]);
            return i;
        }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Abre tu Cuenta NoCuenta con el código WUOLAH10 y llévate 10 € al hacer tu primer pago

3. (1 punto) Implementar una función:
void divide_por_signo(list<int> &L,vector< list<int> > &VL);

que dada una lista L, devuelva en el vector de listas VL las sublistas contiguas del mismo signo (el 0 se considera junto con los positivos). El algoritmo puede modificar a L

Ejemplos:

L: {4,-3,-5,-4,-5,-1,4,-1,-5,-5} => **VL: [{4}, {-3,-5,-4,-5,-1}, {4}, {-1,-5,-5}]**

L: {0,4,-2,4,1,-1,-4,-4,-3,-1,-4,4,1} => **VL: [{0,4}, {-2}, {4,1}, {-1,-4,-4,-3,-1,-4}, {4,1}]**

L: {2,-1,3,-3,3,-3,0,-1,0} => **VL: [{2}, {-1}, {3}, {-3}, {3}, {-3}, {0}, {-1}, {0}]**

```
void divide_por_signo(list<int> &L, vector<list<int>> &VL) {
    int signo;
    auto it = L.begin();
    while (it != L.end()) {
        list<int> lout;
        if (*it == 0) signo = 1
        else
            signo = abs(*it)/*it ;   { 1 positivos y 0
                                     { -1 negativos
        auto it2 = it;
        bool seguir = true;
        while (it2 != L.end() && seguir) {
            if (*it2 == 0 && signo == 1) {
                lout.push_back(*it2);
                ++it2;
            }
            else {
                if (*it2 == 0 && signo == -1) {
                    seguir = false;
                }
                else
                    if (signo == abs(*it2)/*it2) {
                        lout.push_back(*it2);
                        ++it2;
                    }
                    else
                        seguir = false;
            }
        it = it2;
        VL.push_back(lout);
    }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

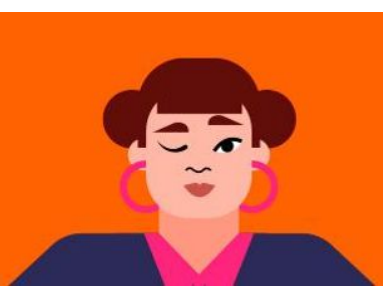
WUOLAH

Abre tu Cuenta NoCuenta con el código WUOLAH10 y llévate 10 € al hacer tu primer pago

Esto no son apuntes pero **tiene un 10 asegurado** (y lo vas a disfrutar igual).

Abre la **Cuenta NoCuenta** con el código **WUOLAH10**, haz tu primer pago y llévate 10 €.

Me interesa



1/6
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.

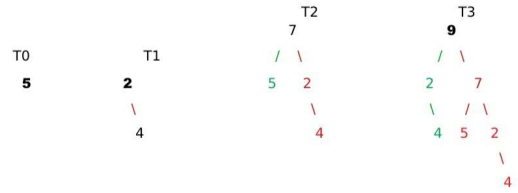
ING BANK NV se encuentra adherido al Sistema de Garantía de Depósitos Holandés con una garantía de hasta 100.000 euros por depositante. Consulta más información en ing.es

4. (1 punto) Implementar la función:

void Fibonacci_Trees(vector<bintree<int>> & v, int n);

que construye la sucesión de árboles binarios de Fibonacci y los almacena en un vector de árboles. La sucesión comienza con un árbol con 1 solo nodo (T0) y un árbol con solo un hijo a la derecha (T1). A partir de ellos, se construye la sucesión construyendo cada árbol binario Ti insertando Ti-1 a la derecha y Ti-2 a la izquierda (i=2,..,n). La etiqueta de la raíz del nuevo árbol se obtiene como la suma de las etiquetas de las raices de los árboles izquierdo y derecho

Ejemplo:




```
void Fibonacci_Trees (vector < bintree <int>> & v, int n){
    for (int i=2; i<n; i++){
        int etraiz = *(v[i-2].root());
        etraiz += *(v[i-1].root());
        bintree<int> nuevo(etraiz);
        bintree<int> ti(v[i-2]);
        bintree<int> td(v[i-1];
        nuevo.insert_left(nuevo.root(), ti);
        nuevo.insert_right(nuevo.root(), td);
        v.push_back(nuevo);
    }
}
```

5. (1 punto) Dados dos map, M1 y M2, definidos como:
map<string,int> M1,M2;
con el primer campo representando el nombre de una **persona** (string) y el segundo campo su **numero de seguidores** (int) en una red social, **implementar una función:**

map<string,int> Union (const map<string,int> &M1, const map<string,int> &M2);

que obtenga el map correspondiente a la unión de los dos map de entrada, en el que el numero de seguidores será la suma de los seguidores en M1 y los seguidores en M2 para la misma persona que aparece en M1 y M2. En el caso que solamente aparezca en uno de los dos se queda tal cual en el map resultado

```
map<string, int> Union (const map<string, int> &M1, const map<string, int> &M2){
    map<string, int> Mout(M1);
    map<string, int>::iterator it1;
    map<string, int>::const_iterator it2;

    for(it2 = M2.cbegin(); it2 != M2.cend(); ++it2){
        it1 = Mout.find((*it2).first);
        if(it1 != Mout.end())
            it1->second += it2->second;   // Mout[it1->first] += it2->second;
        else
            Mout.insert(*it2);
    }
    return Mout;
}
```

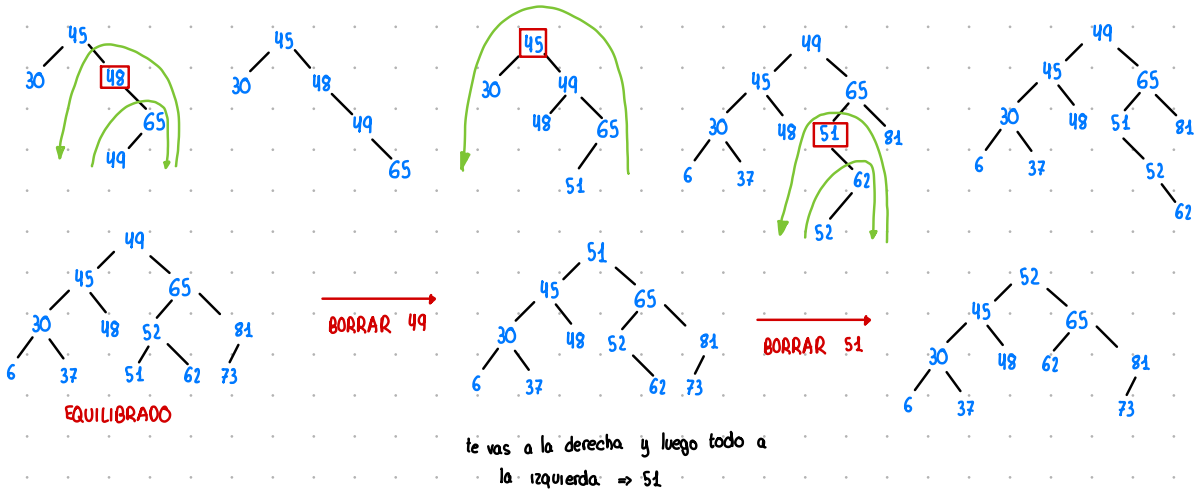
Consulta condiciones aquí

do your thing

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

6. (1 punto) (a) Insertar en el orden indicado (detallando los pasos) las siguientes claves en un **AVL**: {45, 30, 48, 65 ,49 ,51, 81, 37, 6, 62, 52, 73 }. Borrar el elemento 49 del árbol



te vas a la derecha y luego todo a la izquierda ⇒ 51

(b) Insertar (detallando los pasos) las claves {8, 16, 12, 41, 10, 62, 27, 65, 13} en una **Tabla Hash cerrada** de tamaño 13. A continuación borrar el 10 y finalmente insertar el valor 51. Resolver las colisiones usando hashing doble.

$M = 13$

$h_i(k) = h(k) = K \% 13$

- $h_0(k) = 1 + (k \% 11)$
- $h_i(k) = (h_{i-1}(k) + h_0(k)) \% M \quad i = 2,3\ldots$

	8	16	12	41	10	62	27	65	13	51
$h(k)$	8	3	12	2	10	10	1	0	0	12
$h_0(k)$	9	6	2	9	11	8	6	11	3	8

	K	dir	estado
0	65		x
1	27		x
2	41		x
3	16		x
4			
5	62		x
6	13		x
7	51		x
8	8		x
9			
10	10		B
11			
12	12		x

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Abre tu Cuenta NoCuenta con el código WUOLAH10 y llévate 10 € al hacer tu primer pago