

```

list<int> nivel (const bintree<int> &A) {
    typedef pair<const bintree<int>::node, int> minfo;
    queue<minfo> miq;
    miq.emplace (A.root(), 0);
    map<int, list<int>> mimap;
    while (!miq.empty()) {
        minfo aux = miq.front(); miq.pop();
        mimap[aux.second].pushback (aux.first);
        if (!aux.first.left().null())
            miq.emplace (aux.first.left(), aux.second + 1);
        if (!aux.first.right().null())
            miq.emplace (aux.first.right(), aux.second + 1);
    }
    map<int, list<int>::iterator> it_max;
    it_max = mimap.begin();
    for (it = mimap.begin(); it != mimap.end(); ++it) {
        if ((*it).second.size() > (*it).second.size())
            it_max = it;
    }
    // it_max apunta a elemento con lista de mayor tam.
    return it_max->second;
}

```

⑧ list<int> camino_demenores (const bintree<int>& A);

dado un arbol binario devuelva en L el camino del arbol a una hoja de forma
que la suma de sus etiquetas sea la menor posible.

list<int> camino_demenores (const bintree<int> &A) {
 return c.m(A.root());

}

list<int> c-m (bintree<int>::node n) {

if (n.null())
 return list<int>();

else {

list<int> l-i = c-m(n.left());

list<int> l-d = c-m(n.right());

if (suma - l(i) < suma - l(d)) {

~~l-d.push-front(&n);~~

return l-d;

else {

l-i.push-front(&n);

return l-i;

}

Otra forma:

```
bool esABB (ArbolBinario c.ints::nodo n, int min, int max) {
    if (n.nvlc())
        return true;
    else if (! (n>min && n<max))
        return false;
    return esABB (n.hic(), min, n) && esABB (n.hdc(), n, max);
}
```

12) Compruebe si es un ABB

```
bool esABB (ArbolBinario<int> &ab, ArbolBinario<int>::nodo n)
{
    if (n.null())
        return true;
    else
        int mayor = getMayor (n.hi());
        int menor = getMenor (n.hd());
        if (!n < mayor || !n > menor)
            return false;
        return esABB (ab, n.hi()) && esABB (ab, n.hd());
}
```

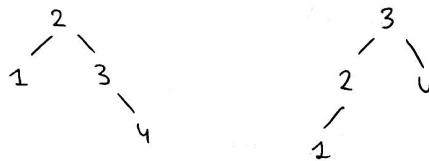
int getMayor (ArbolBinario<int>::nodo n)

```
if (n.null())
    return numeric_limits<int>::max();
else
    int max = n;
    int m_i = getMayor (n.hi());
    int m_d = getMayor (n.hd());
    if (max < m_i)
        max = m_i;
    if (max < m_d)
        max = m_d;
    return max;
```

// Igual getMin

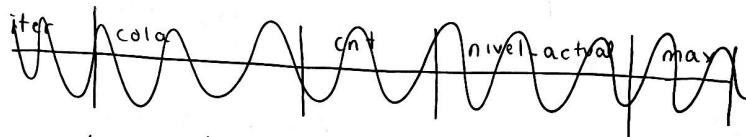
- ④ Dado un árbol binario cuyas etiquetas estan organizados como un AVL, medo reemplazar de forma univoca a partir de su recorrido en inorden. Falso. Hay que dar contraejemplo

1, 2, 3, 4 \rightarrow Inorden

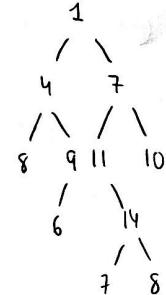


- ⑤ Función `list<int> nivel (const bintree <int> &A);`

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



iter	cola	mimap	nivel actual
0	$\langle 1, 0 \rangle$		
1	$\langle 4, 1 \rangle, \langle 7, 0 \rangle$	$\langle 0, 4 \rangle, \langle 4 \rangle$	
2	$\langle 7, 1 \rangle, \langle 8, 2 \rangle, \langle 9, 2 \rangle$	$\langle 0, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 4 \rangle$	
3	$\langle 8, 2 \rangle, \langle 9, 2 \rangle, \langle 11, 2 \rangle, \langle 10, 2 \rangle$	$\langle 0, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 4 \rangle, \langle 11, 7 \rangle, \langle 1, 4 \rangle, \langle 7 \rangle$	
4	$\langle 9, 2 \rangle, \langle 1, 4 \rangle, \langle 10, 2 \rangle$	$\langle 0, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 7 \rangle, \langle 2, 4 \rangle, \langle 8 \rangle$	
5	$\langle 11, 2 \rangle, \langle 10, 2 \rangle, \langle 6, 3 \rangle$	$\langle 0, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 7 \rangle, \langle 2, 4 \rangle, \langle 8 \rangle, \langle 9 \rangle$	

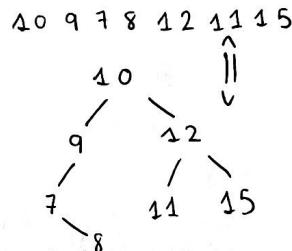
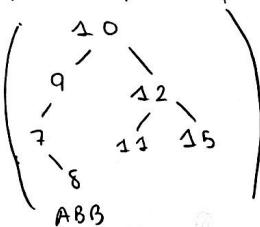


iter	cola	mimap
6	$\langle 10, 2 \rangle, \langle 6, 3 \rangle$	$\langle 0, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 7 \rangle, \langle 2, 4 \rangle, \langle 8 \rangle, \langle 9 \rangle, \langle 11 \rangle$
7	$\langle 6, 3 \rangle$	$\langle 0, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 4 \rangle, \langle 1, 4 \rangle, \langle 7 \rangle, \langle 2, 4 \rangle, \langle 8 \rangle, \langle 9 \rangle, \langle 11 \rangle, \langle 14 \rangle$

Ejercicios Árboles - Exámenes

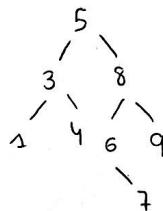
① ¿El orden en que las hojas se listan en los recorridos preorden, inorden y postorden de un árbol binario es el mismo en los tres casos? Verdadero. Siempre de izquierda a derecha (HOJAS) si fueran nodos sería falso.

② Dado un árbol binario cuyas etiquetas están organizadas como un ABB, puedo recuperarlo a partir de su preorden. Verdadero.



③ Dado un ABB, 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

Res: 7, 1, 4, 6, 9, 3, 8, 5



Había falta sacar los resultados.

y sobrecargar > porque sino, la priority queue, no sería priority-queue

```
#include <queue>
void ListaProfundidad (ArbolBinario ab)
{
    typedef pair<int, ArbolBinario*> nodo;
    priority-queue<nodo> pq;
    list<int> salida;
    pq.emplace(0, ab.getRaiz());
    while (!pq.empty())
    {
        nodo aux = pq.top();
        pq.pop();
        salida.push_back(aux.second);
        if (!aux.second.hd().null())
            pq.emplace(aux.first + 1, aux.second.hd());
        if (!aux.second.hd().null())
            pq.emplace(aux.first + 1, aux.second.hd());
    }
}
```

```
return (
```

Numerocaminos (bintree , int k) {

```
typedef bintree<int>::nodo n;
```

```
int contador = 0;
```

```
n = ab.raiz();
```

```
if (n.left() == bintree::null() || n.right() == bintree::null())
```

```
if (*n == k)
```

```
return 1;
```

```
else
```

```
return 0;
```

```
} else {
```

```
int contador = 0;
```

```
if (n.left() != bintree::null() || !n.left().null()) {
```

```
contador += NumeroCaminos (ab, k - *n, n.left());
```

```
if (n.right() != bintree::null())
```

```
contador += NumeroCaminos (ab, k - *n, n.right());
```

```
return contador;
```

```
}
```

Feb 19 - 20

```
bool check_sum (const list<int> &L1, const list<int> &L2) {  
    list<int>::iterator it1, it2;  
    it1 = L1.begin();  
    it2 = L2.begin();  
    list<int> comparar (L2.size(), 0);  
    while (it1 != L1.end() && it2 != L2.end()) {  
        if (*it2 != 0) {  
            *it2 -= *it1;  
            ++it1;  
        }  
        else  
            ++it2;  
    }  
    return (it1 == L1.end()) && (comparar == L2);  
}
```

$L1 = \{1, 2, 3, 4, 1, 3, 2, 5, 6, 8, 3\}$

$L2 = \{6, 10, 5, 6, 11\}$

$$6 - 1 = 5 - 2 = 3 - 3 = 0$$

$$10 - 4 - 1 - 3 - 2 = 0 \dots$$

```

class Freq_char {
private:
    map<char, unsigned int> datos;
public:
    Freq_char (vector<char> &a) {
        vector<char>::iterator it; it = a.begin();
        while (it != a.end()) {
            pair<char, unsigned int> inserta;
            while (char letra = *it; ++it) {
                unsigned int cuenta_letras = 0;
                while (*it == letra) {
                    cuenta_letras++;
                    ++it;
                }
                inserta->first = letra; inserta->second = cuenta_letras;
                datos.insert(inserta);
            }
        }
    }
}

```

```

char operator [] (int) {
    map<char, unsigned int>::iterator it; it = datos.begin();
    unsigned int cnt = 0;
    while (it != datos.end()) {
        if (i >= cnt && i < (cnt + (*it).second)) {
            return it->first;
        } else {
            cnt = *it.second;
            ++it;
        }
    }
    return 0;
}

```

```
class iterator {
```

```
private:
```

```
    map<char, unsigned int>::iterator i; final;
```

~~bool condicion (const iterator& it) const;~~

```
public:
```

```
    bool condicion (unsigned int n) {
```

```
        return n%2 == 0;
```

```
}
```

```
public
```

```
    iterator() {
```

```
        bool operator == (const iterator& it) const {
```

```
            return i == it.i;
```

```
}
```

```
        bool operator != (const iterator& it) const {
```

```
            return i != it.i;
```

```
        }
```

```
        pair<char, unsigned int> operator * (const) {
```

```
            return *i;
```

```
}
```

```
        iterator& operator ++ () {
```

```
            ++i;
```

```
            while (i != final && !condicion(i->second)) {
```

```
                ++i;
```

```
}
```

```
            return *this;
```

```
}
```

```
        friend class Freq_char;
```

```
}
```

```
    iterator begin () {
```

```
        iterator it;
```

```
        i = datos.begin();
```

```
        it = datos.end();
```

```
        if (!it.condicion(i->second)) ++it;
```

```
        return i;
```

```
}
```

```
    iterator end () {
```

```
        iterator it;
```

```
        i = datos.end();
```

```
        it = datos.end();
```

```
        return it;
```

```
}
```

```
int mas_conectado (const vector<set<int>> &VS) {
```

~~data A, factores de VS, const int~~

~~vector de conjuntos~~

```
vector<int> unsigned conexiones (VS.size(), 0);  
for (unsigned int i = 0; i < VS.size(); i++) {
```

```
    for (unsigned int j = 0; j < VS.size(); j++) {
```

if (!son_disjuntos (VS[i], VS[j])) {

conexiones[i]++;

conexiones[j]++;

```
int mas_conectado = 0;  
int pos = -1;
```

```
for (unsigned int i = 0; i < VS.size(); i++) {
```

if (conexiones[i] > mas_conectado) {

pos = i;

mas_conectado = conexiones[i];

return pos;

}

```
bool son_disjuntos (const set<int> &S1, const set<int> &S2) {
```

~~vector<int>::const_iterator it;~~

```
for (it = S1.begin(); it != S1.end(); ++it) {
```

*if (S2.count(*it) > 0)*

return false;

return true;

Y

{104, 514, 624, 40, 1, 24}

0 1 2 ③

vector<int>

conexiones (VS.size(), 0);

si !son_disjuntos i, j

i++

j++;

return j → conjunto más conectado

con otros conjuntos

VS

⑥ class contenedor.h

private:

unordered_map<T, list<int>> datos;

public:

bool todosPares (list<int> &lista) {

auto it; bool todos_pares = true;

for (it = lista.begin(); it != lista.end(); ++it) {

if (*it % 2 != 0) {

todos_pares = false;

}

return todos_pares;

}

class iterator.h

private:

unordered_map<T, list<int>>::iterator it;

unordered_map<T, list<int>>::iterator final;

public:

bool operator == (const iterator &i) {

const

return it == i.it;

}

bool operator != (const iterator &i) const {

return it != i.it;

}

iterator & operator ++ () {

do {

++it;

while (it != final && !TodosPares(*it->second));

return ~~**this~~;

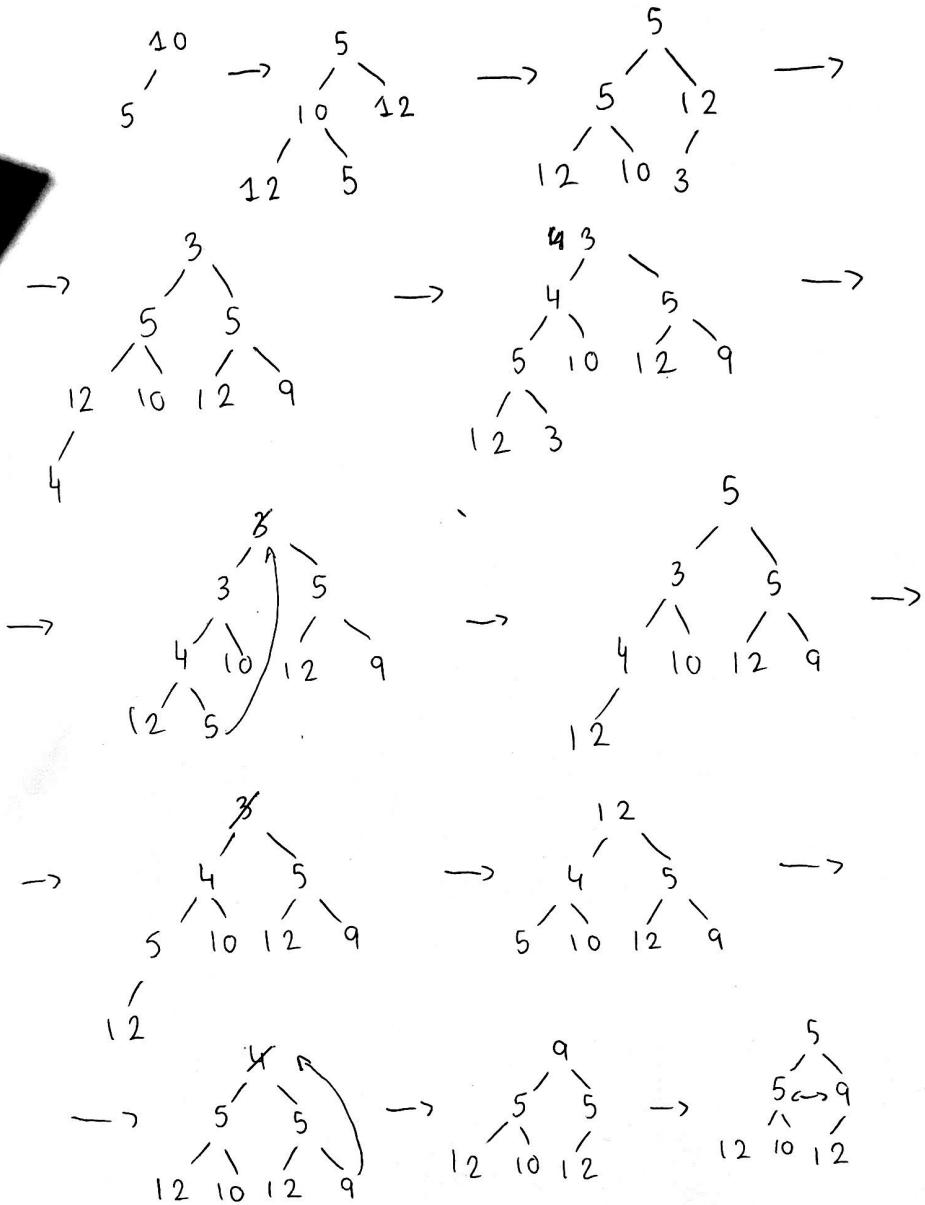
}

friend class contenedora;

const
pair<T, list<int>>
operator * () {
return *it;

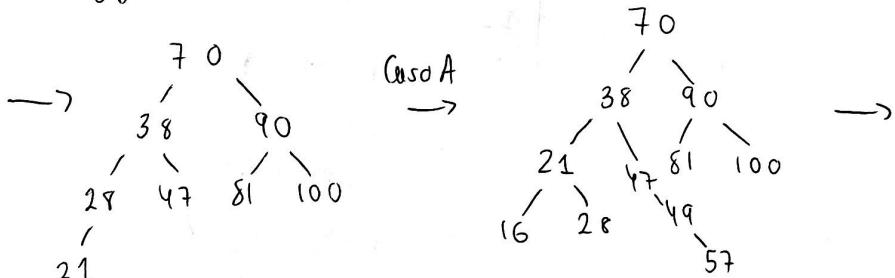
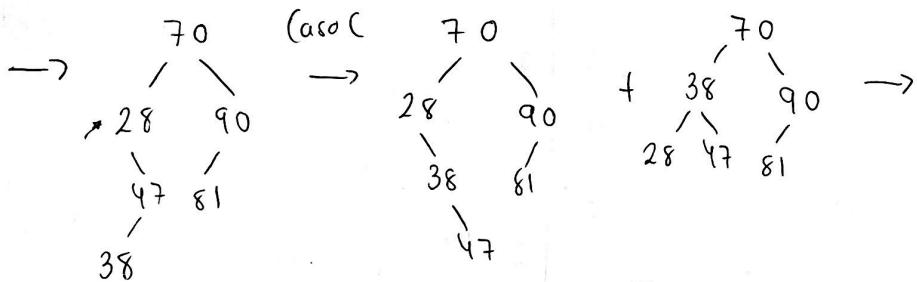
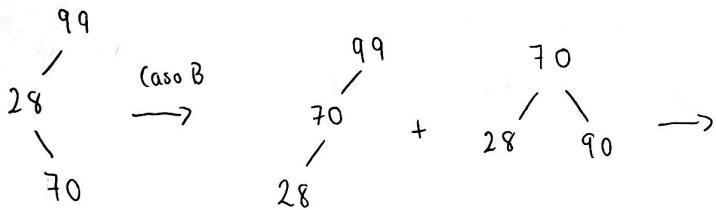
}

c) AP0 $\downarrow [10, 5, 12, 12, 5, 3, 9, 4, 3] \downarrow$

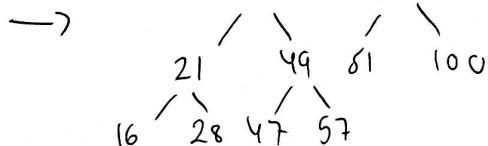


b) AVL

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



Caso D



(5)

$$a) \{4, 12, 16, 37, 6, 58, 23, 61, 9, 10\}$$

$$h_0(x) = x \% 11$$

$$h_i(x) = [h_{i-1}(x) + h_0(x)] \% 13$$

$$h(x) = x \% 13$$

0		7	
1	23	8	30
2	9	9	61
3	16	10	58
4	4	11	37 48
5		12	12
6	6		

$$h_0(58) = 4$$

$$h_2(48) = [6 + 4] \% 13 = 10$$

$$h_0(23) = 2 \quad h(23) = 10$$

$$h_2(23) = [10 + 2] \% 13 = 12$$

$$h_3(23) = 12 + 2 \% 13 = 1$$

$$h_0(9) = 10$$

$$h(9) = 9$$

$$h_2(9) = [10 + 9] \% 13 = 6$$

$$h_3(9) = 6 + 9 \% 13 = 2$$

$$h_0(10) = 11$$

$$h(10) = 10$$

$$h_2(10) = 11 + 10 \% 13 = 8$$

$$h_0(48) = 5 \quad h(48) = 9 \quad h_2(48) = (9 + 5) \% 13 = 1 \quad h_3(48) = 1 + 5 \% 13 = 6$$

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

```
ng palabra (int i) const
Map<string, set<int>> datos;
it = datos.begin();
for( ; it != datos.end(); ++it) {
    set<int>::c_iterator sit = it->second.begin();
    if (sit != it->second.end())
        return it->first;
}
return n";
```

③ L 2 listas

```
void reemplaza (list<int> &l, const list<int> &seq, const list<int> &reemp);
list<int>::const_iterator it;
```

②

class Documento {

private:

map<string, set<int>> datos;

public:

a) Documento (const list<string> & texto) {

list<string>::const_iterator it;
map<string, set<int>>::iterator mit;
it = texto.begin();
int pos = 0;

while (it != texto.end()) {

pair<string, set<int>> insertar;

mit = datos.find (*it);

if (mit == datos.end()) {

set<int> aux;

aux.insert (pos);

pair<string, set<int>> insertar (*it, aux);

datos.insert (insertar);

}

else {

mit -> second.insert (pos);

}

++it;

pos++;

b)

set<int> ~~de~~ obtener_posiciones (string palabra) const {

set<int> s;

map<string, set<int>>::const_iterator it;

if (it = datos.find (palabra));

if (it != datos.end)

return it -> second;

else return set<int> s();

}

$$n_1 \neq n_2$$

$$\checkmark \quad \downarrow$$

$$m_1 \quad m_2 \Rightarrow \text{dist a AMC}$$

$$m_1=0 \quad m_2=0 \Rightarrow n_1=n_2$$

a) V

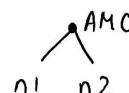
b) V

c) V

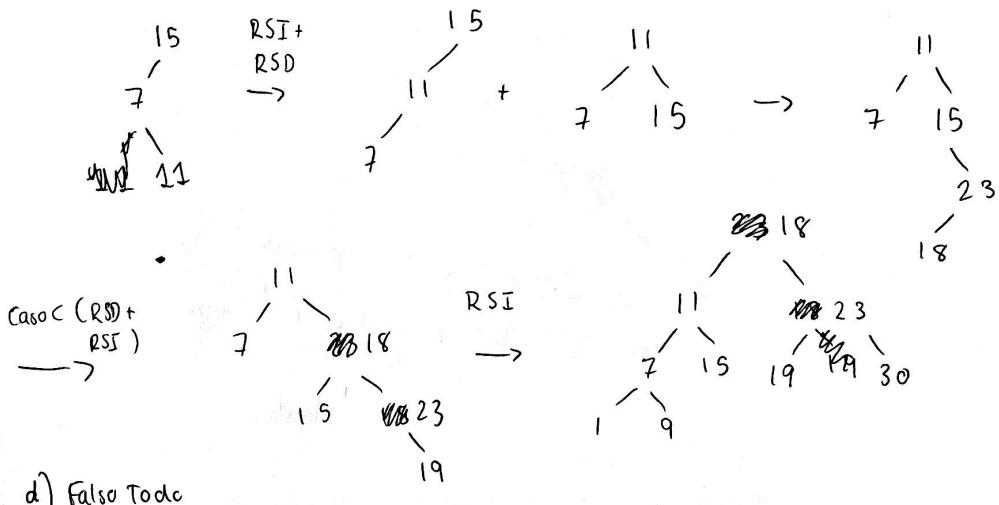
d) Todo aero

$$m_1=0 \quad y \quad m_2>0$$

n2 sucesor n1



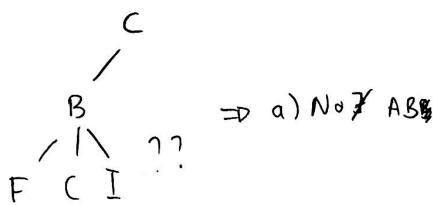
?) $15, 7, 11, 23, 18, 19, 9, 30, 14$ AVL



③

Preorden = {C B F C I H G A J D E}

Postorden = {F I C B H A D J E G C}



④ d

list<int> caminomenores (const bintree<int>& A)

metodo:

pair<int, list<int> caminomenores (nodo) {

```
if (nodo == null)
    return {0, list()};
else if (nodo.left == null && nodo.right == null)
    return {nodo.root, list()};
else {
```

```
    list<int> resultado;
    resultado.push_back(nodo.root);
    resultado += caminomenores(nodo.left);
    resultado += caminomenores(nodo.right);
    return {resultado, resultado};
```

②

- a) T Solo se puede acceder a él con \rightarrow peticiones
- b) F
- c) Falso, mejor no a la función
- d) Falso
- e) Falso

① V/F

a) Verdadero

(Siempre se listan de izquierda a derecha)

b) Verdadero

c) ~~all~~ ✓

d) ~~MF~~ porque en hashing necesitamos acceder a la clave para recuperarla y con $random(M)$ no podríamos

e) F No puede analizarse.

$$\begin{array}{c}
 & & 1 & & 2 & 4 & 5 & 3 \\
 & & / \quad \backslash & & & & & \\
 & & 1 & & & & & \\
 & & / \quad \backslash & & & & & \\
 & & 2 & & 3 & & & \\
 & & / \quad \backslash & & & & & \\
 & & 4 & & 5 & & & \\
 \end{array}
 \quad I_n = 42513
 \quad Post = 4$$

```

graph TD
    5 --- 2
    5 --- 6
    2 --- 1
    2 --- 3
    1 --- 2
    1 --- 3
  
```

5 2 1 3 6

④ ~~Interchangeable~~

(a) \vee

d) \vee

b) ✓

e) F (Numerus)

c) F

1 2 3 4 5
6 7 8

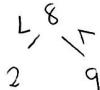
1 2 3 4

7 5 3 2 8 9

```

graph TD
    1_1 --- 5_1
    1_1 --- 2_1
    5_1 --- 8_1
    5_1 --- 7_1
    2_1 --- 9_1
    1_2 --- 5_2
    1_2 --- 2_2
    5_2 --- 8_2
    5_2 --- 7_2
    5_2 --- 9_2
  
```

ABB

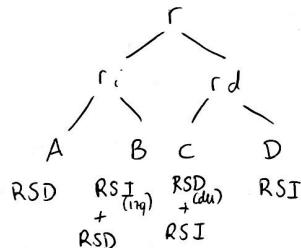


Menor elemento (nodo más izq)

Mayor O($\log_2(n)$) excepto 1 caso O(n)

AVL \rightarrow Un ABB es AVL \Leftrightarrow dif de altura de los subárboles izq y derecho que cuelgan de un nodo es como mucho 1

Insertar $\begin{cases} 1. \text{ Buscar donde insertar} \\ 2. \text{ Insertarlo} \\ 3. \text{ Equilibrar el árbol} \end{cases}$



Hashing cerrado

Rehashing lineal

$$h_i(k) \rightarrow \text{colisión} \rightarrow h_i(k) = (h(k) + (i-1)) \% M \quad i = 2, 3, \dots$$

Casillas $\begin{cases} \text{Ocupada} == \text{Borrada consulta} \\ \text{Vacia} \\ \text{Borrada} \end{cases}$ Y Mismo para inserción

Rehashing doble

(división)

$$h_i(k) = h_1(k) = k \% M \quad \text{y} \quad h_i(k) = (h_{i-1}(k) + h_0(k)) \% M \quad i = 2, 3, \dots$$

$$h_0(k) = 1 + (k \% (M-2)) \quad M \neq M-2 \text{ (primos relativos)}$$

Sondeo Aleatorio

$$h_i(k) = k \% M$$

$$\text{Colisión: } h_i(k) = (h_{i-1}(k) + c) \% M \quad i = 2, 3, \dots$$

M es el tam, $c = \text{cte} > 1$ y primo relativo con M.

Recorrido Por Niveles

queve < infinito > cola;

cola.push(n); //guardar 1er elemento

while (!cola.empty()) {

//ultimo nodo guardado

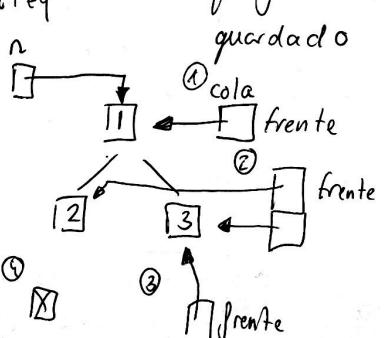
nodo aux = cola.front(); cola.pop(); //borramos el

if (aux->n.left) //tiene hijo izq

cola.push(

if (aux->n.right)

cola.push



APQ \rightarrow et cada nodo \leq hijos

M[0] \rightarrow root M[2] \rightarrow right

M[1] \rightarrow left nodo k \rightarrow M[k]

Eficiencia $O(\log_2(n))$ (Inserción)

y borrado.

Borrado:

//colocar raiz en ultimo nodo

datos[0] = datos[datos.size() - 1]

datos.pop_back() //borramos

ultimo = datos.size() - 1;

int pos = 0;

bool acabar = false;

while (pos <= (ultimo - 1) / 2 && !acabar) {

int pos_min;

if ((pos * 2) + 1 == ultimo) //ver que hijos es

pos_min = ultimo;

else

if (datos[2 * pos + 1] < datos[2 * pos + 2]) else (hacer)

pos_min = (2 * pos) + 1;

Comparar con nodo padre hasta que se cumpla cond APQ

Insertar

Borrar Minimo

M[2k+1] y M[2k+2] (hijos)

USANDO VECTOR \rightarrow

datos.push_back(valor);

int pos = datos.size() - 1;

while (pos > 0 && datos[pos] < datos[(pos - 1) / 2])

swap (datos[pos], datos[(pos - 1) / 2]);

pos = (pos - 1) / 2;

}

if (datos[pos] > datos[pos_min])

swap (datos[pos],

datos[pos_min]);

pos = pos_min;

else acabar = true;

pos_min = (2 * pos) + 2;