

¿ $\Delta B B$ diferentes con las claves $1, 2, \dots, n-1, n$?

$n=1$

(1)

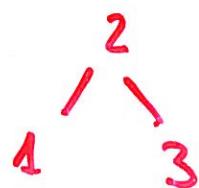
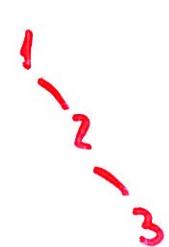
(1)

$n=2$



(2)

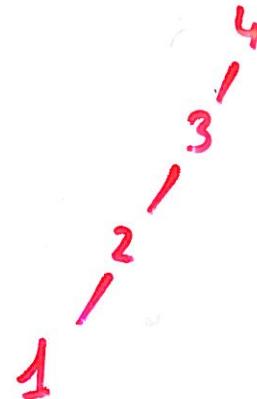
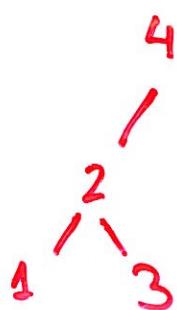
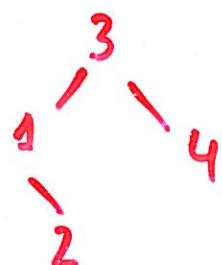
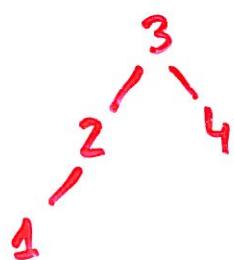
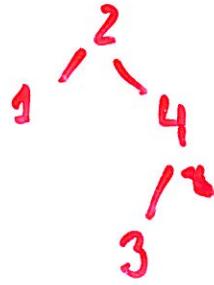
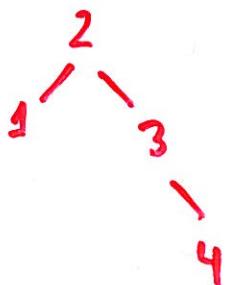
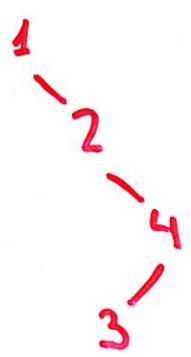
$n=3$



(5)



$n=4$

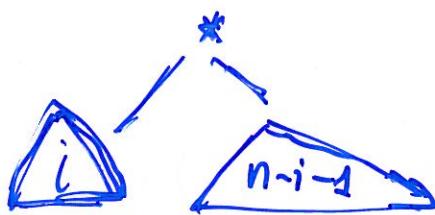


(14)

¿Cuántos para un n arbitrario?

n : ΔBB diferentes con las claves $\{1, 2, \dots, n\}$

$$\Delta(n) = \sum_{i=0}^{n-1} A(i) * A(n-i-1)$$



$$A(0) = 1, \quad A(1) = 1, \quad A(2) \approx 2$$

$$A(3) = A(0) * \Delta(2) + \Delta(1) * \Delta(1) + A(2) * \Delta(0) \approx 5$$

$$A(4) = A(0) * \Delta(3) + \Delta(1) * \Delta(2) + \Delta(2) * \Delta(1) + \Delta(3) * \Delta(0) \approx 14$$

$$\Delta(5) = A(0) * \Delta(4) + \Delta(1) * \Delta(3) + \Delta(2) * \Delta(2) + \Delta(3) * \Delta(1) + \Delta(4) * \Delta(0) \approx 42$$

$$\begin{aligned} \Delta(6) = & \Delta(0) * \Delta(5) + \Delta(1) * \Delta(4) + \Delta(2) * \Delta(3) + \Delta(3) * \Delta(2) + \\ & + \Delta(4) * (\Delta(1) + \Delta(5)) * \Delta(0) = 132 \end{aligned}$$

$$\Delta(3) = \sum_{i=0}^2 \Delta(i) * A(3-i-1)$$

$$\boxed{\Delta(7) = 429}$$

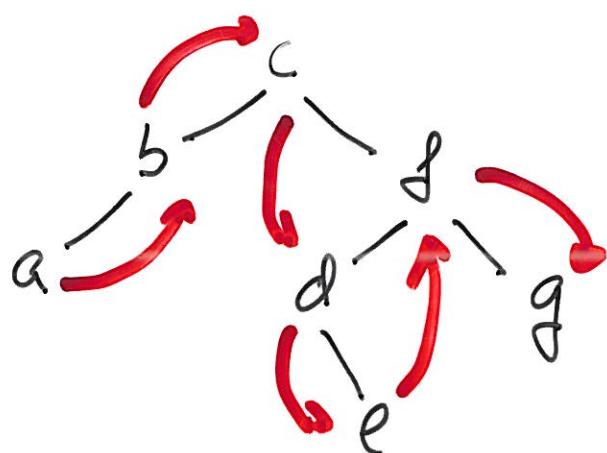
$$\Delta(4) = \sum_{i=0}^3 \Delta(i) * \Delta(4-i-1)$$

$$\Delta(5) = \sum_{i=0}^4 \Delta(i) * \Delta(5-i-1)$$

$$\Delta(6) = \sum_{i=0}^5 \Delta(i) * \Delta(6-i-1)$$

Ejercicio

Un arbol binario hilvanado es una estructura de datos para la implementacion de un ABBS que facilita ciertos recorridos. Su caracteristica esencial es que ^{en} el registro que define un nodo se añade un nuevo puntero hilvan que apunta al siguiente nodo en el recorrido en orden del ABBS. Usando el TDA ABBS, diseñar un procedimiento para insertar un nodo en un arbol binario hilvanado



Construir un iterador para una clase diccionario cuya definición privada es:

~~template <class T, class V>~~

class diccionario {

private:

list <data <T, V> > datos;

};

y;

con data definido como:

template < class T, class V >

struct data {

T clave;

list <V> info_asoci;

};

Diccionario <string, string> D;

Clave { Programa

- Plan, proyecto o declaracion de algo que se piensa hacer
- Distribucion de materias de un curso
- Anuncio del reporte y anuncio final de un aspecto

```
class iterator {
    private:
        typename list<data<T,U>>::  
        iterator punt;
    public:
        iterator() {}
        iterator & operator++() {
            punt++;
            return *this;
        }
        iterator & operator--() {
            punt--;
            return *this;
        }
        bool operator==(const iterator &it) {
            return it.punt == punt;
        }
        bool operator!=(const iterator &it) {
            return it.punt != punt;
        }
        data<T,U> & operator*() {
            return *punt;
        }
    friend class Diccionario;
    friend class const_iterator;
```

```
iterator begin() {
    iterator it;
    it.punt = datos.begin();
    return it;
}
```

```
iterator end() {
    iterator it;
    it.punt = datos.end();
    return it;
}
```

ostream & operator << (ostream & os, const Diccionario<string, string> & D) {

```
Diccionario<string, string>::const_iterator it;
for (it = D.begin(); it != D.end(); ++it) {
    list<string>::const_iterator it_s;
    os << endl << (*it).clave << endl << "
        informacion asociada:" << endl;
    for (it_s = (*it).info_asoci.begin(); it_s != (*it).info_asoci.end(); ++it_s) {
        os << (*it_s) << endl;
    }
}
```

os << "-----" << endl;
return os;

Disponemos del TDRs matriz de enteros (se almacenan los datos por filas) y se quiere definir un iterador que itere por columnas sobre los elementos pares de la matriz. Para ello hay que implementar los operadores `++` y `*`, así como las funciones `begin()` y `end()` en la clase `matriz`. P. ej. si la matriz M ~~es~~:

$$\begin{bmatrix} 5 & 4 & 3 \\ 2 & 1 & 2 \\ 9 & 0 & 2 \\ 8 & 9 & 1 \end{bmatrix} \quad \downarrow$$

ejecutando el siguiente código

Matriz M;

⋮⋮⋮

Matriz::iterator it;

for (it = M.begin(); it != M.end(); ++it)

cout << *it;

se imprimirá sobre la salida estandar:

2, 8, 4, 0, 2, 2

```

class Matrix {
    int ** datos;
    int nf, nc;
}

public:
    class iterator {
private:
    int * d;
}

public:
    int & operator [] () const;
    iterator & operator++ ();
}

iterator end () {
    return &d[nf-1][nc-1];
}

Matrix::iterator it = &datos[nf-1][nc-1];
return it;
}

iterator begin () {
    Matrix::iterator it;
    it.d = &datos[0][0];
    if (*it.d) % 2 == 0 return it;
    else { ++it; // pasa al siguiente par
            return it;
    }
}

```

Matrix: iterator & operator $\> \>$ () {

```

        int f = (d - & (datos[0][0])) / nc; // fila donde
                                                // apunta
        int c = (d - & (datos[0][0])) % nc; // columna donde
                                                // apunta
        while (true) {
            if (f >= nf - 1 && c >= nc - 1) // ya hay mas
                                                // elementos
                ^ d = & (datos[nf - 1][nc - 1]);
                return *this; // devolvemos end
            }
            else
                if (f < nf - 1) {
                    f = f + 1; // en la misma columna, el siguiente
                    d = & (datos[f][cc]);
                    if (*d % 2 == 0) return *this;
                }
                else { // hemos recorrido toda la columna
                    f = 0; // pasamos a la siguiente columna
                    c = c + 1;
                    d = & (datos[f][c]);
                    if (*d % 2 == 0) return *this;
                }
        }
    }

    friend class Matrix;

```

construir una clase guia_de_teléfonos que de
soporte al manejo de información del tipo:

Susana Amiba 958665544

Alejandra Santillan 653453111

Carlos Pineda 606112233

Carolina Fuentes 958334455

Fernando Ruiz 651234567

: - : = : (sin nombres repetidos)

Con funciones para:

- Devolver un tfno asociado a un nombre
- Insertar un nuevo elemento en la guia
- borrar un elemento de la guia
- Imprimir la guia

```
#include <map>
#include <iostream>
#include <string>
```

```
class Guia_Tlf {
```

```
private:
```

```
map<string, string> datos;
```

```
public:
```

```
string & operator[] (const string & nombre)
```

```
    { return datos[nombre]; }
```

```
}
```

```
pair<map<string, string>::iterator, bool>
```

```
insert (pair<string, string> p)
```

```
{
```

```
pair<map<string, string>::iterator, bool> ret;
```

```
ret = datos.insert(p);
```

```
return ret;
```

```
}
```

```
void bonar (const string & nombre)
{
    map<string, string>::iterator it_low =
        datos.lower_bound(nombre),
    map<string, string>::iterator it_upper =
        datos.upper_bound(nombre),
    datos.erase(it_low, it_upper);
}
```

```
friend ostream & operator << (ostream & os,
                                const T & g)
{
    map<string, string>::iterator it;
    for (it = g.datos.begin();  

         it != g.datos.end(); ++it) {
        os << it->first << "\t" <<  

        it->second << endl;
    }
    return os;
}
```

Se quiere construir el tipo de dato vectorDisperso de string, que se caracteriza porque la mayoría de los elementos toman el mismo valor (que llamaremos valor por defecto). Para representar dicho valor es más eficiente almacenar solo los elementos distintos que lo que se propone la siguiente representación

```
class VectorDisperso {
```

```
    :::
```

```
private:
```

```
    map <int, string> M; //map que alberga los  
    string v_def; //valor por defecto //elementos no nulos
```

```
};
```

donde, para un vectorDisperso v , el elemento de la posición i -ésima del vector, $v[i]$ sería:

$$v[i] = \begin{cases} M[i] & \text{si } M.\text{find}(i) \neq M.\text{end}() \\ v_def & \text{en otro caso} \end{cases}$$

Implementar un método que cambie el valor por defecto del vector (teniendo cuidado con los elementos del vector disperso cuyo valor anterior fuese nr)

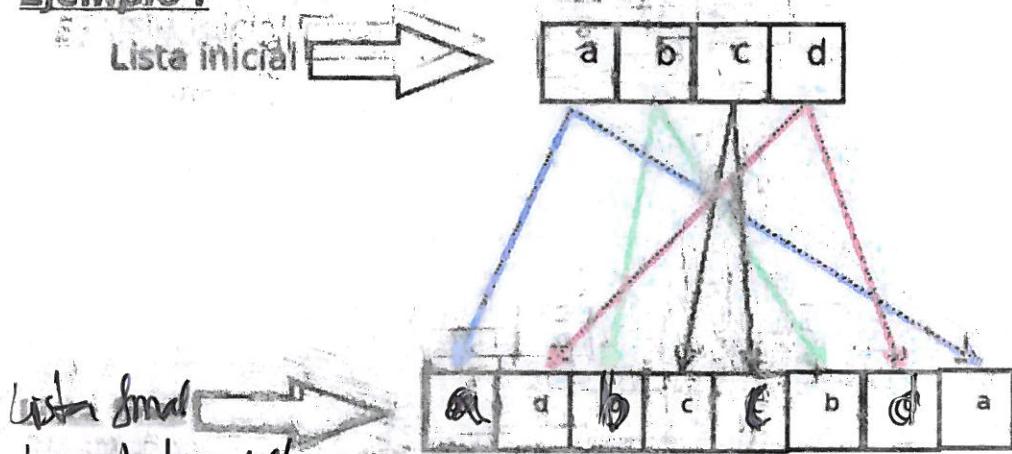
void VectorDisperso::Cambiar_valor_defecto (const string & nr)

```
void vectorDisparo::cambiar_valor_defecto (const
{
    map<int, string>::iterator it;
    for (it = M.begin(); it != M.end(); )
    {
        if (M[*it] == nv) // (*it).second == nv
            M.erase(it);
        else it++;
    }
    v_def = nv;
}
```

Ejercicio Duplicar lista

Usando la clase `list<T>`, construir una función que permita "duplicar" una lista intercalando alternativamente tras cada elemento en la posición i , el elemento que está en la posición $n - i - 1$ ($i = 0, 1, \dots, n - 1$).

Ejemplo :



template <class T>
void duplicar(push_t list<T> &lista_inicial,

list<T> &lista_final);
assert (lista_inicial.size() > 0);

list<T> copia_lista(lista_inicial);

typename list<T>::const_iterator rit;
typename list<T>::reverse_iterator ritr;

for (rit = lista_inicial.begin(), ritr = copia_lista.rbegin();
rit != lista_inicial.end(), ritr != copia_lista.rend();
++rit, ++ritr)
 lista_final.push_back(*rit);
 lista_final.push_back(*ritr);

}

Diseñar una función que dada una lista de enteros elimine todos aquellos que no sean más grandes que todos los anteriores.

$$\langle 1, 3, 4, 2, 4, 7, 7, 1 \rangle \longrightarrow \langle 1, 3, 4, 7 \rangle$$

void subsecuencia (list<int> & l)

 list<int>::iterator it = l.begin();

 while (it != l.end())

 list<int>::iterator next = it;

 ++next;

 if (next != l.end() && *it > *next)

 l.erase(next);

 else it=next;

}

}

Vectores de vectores: matrices

vector <vector<T>>

una matriz es un vector de "filas" donde cada "fila" es un vector de elementos. La diferencia de los arrays bidimensionales de C++ no existe obligación de que todas las filas tengan el mismo número de elementos: cada vector fila es independiente de los demás y la única restricción es que todos contengan elementos del mismo tipo

Crear una matriz de n filas por m columnas no es más que crear un vector de n filas, donde cada uno de los elementos de ese vector es una fila de m columnas

Usar vector <vector<T>> es algo menos eficiente que usar arrays al guardar las filas por separado y tener que realizar varios accesos de memoria para acceder a una posición (i,j). A cambio las matrices son más versátiles que los arrays.

Vectores de vectores: matrices

typedef vector<int> Fila;

typedef vector<Fila> Matriz;

int main()

Matriz M(5); //Matriz de 5 filas vacias

for (int i = 0; i < M.size(); ++i)

M[i].resize(3); //Matriz 5x3

Matriz M2(5, Fila(3)) //otra forma de tener
//una matriz 5x3

for (int i = 0; i < M.size(); ++i)

for (int j = 0; j < M[i].size(); ++j)

M[i][j] = M2[i][j] = i + j;

//se rellenan las matrices

}

typedef vector<int> File;

typedef vector<File> Matrix;

Matrix products (const Matrix & ma,
const Matrix & mb)

{

int nfa = ma.size(),
nca = ma[0].size(),
ncb = mb[0].size();

Matrix m (nfa, File(ncb));

for (int i = 0; i < nfa; ++i)

 for (int j = 0; j < ncb; ++j)

 for (int k = 0; k < nca; ++k)

 m[i][j] += ma[i][k] * mb[k][j];

return m;

}

//ma y mb rectangulares y el número

// de columnas de ma es igual al número

// de files de mb

```

set<int> nu_comunes (set<int> c1, set<int> c2)
{
    set<int>::iterator p, q;
    set<int> solution;
    for (p = c1.begin(); p != c1.end(); ++p)
        if (!c2.count(*p)) solution.insert(*p);
    for (q = c2.begin(); q != c2.end(); ++q)
        if (!c1.count(*q)) solution.insert(*q);
    return solution;
}

```

```

int ArbolBinario<int>::contar (Nodon, const ArbolBinario<int> &q)
{
    if (n == &) return 0;
    else return 1 + contar (q.izqda(n), q)
              + contar (q.dcha(n), q),
}

```

```

bool es_raiz_2nodo (const ArbolBinario<int> &A, int z)
{
    return abs (contar (A.izqda (A.raiz()), A) -
                contar (A.dcha (A.raiz()), A)) > 2
}

```

```
template <class T>
list <T> repetidos(list <T> l)
{
    set <T> elem_dif;
    set <T> rep;
    list <T> l_rep;
    for (typename list <T>::iterator it =
        l.begin(); it != l.end(); ++it)
    {
        if (elem_dif.find (*it) == elem_dif.end())
            elem_dif.insert (*it);
        else
            rep.insert (*it);
    }
    for (typename set <T>::iterator it = rep.begin();
        it != rep.end(); ++it)
        l_rep.push_back (*it);
    return l_rep;
}
```

queue<int> multiIntersection(queue<int> q1,
queue<int> q2)

```
{  
    queue<int> q;  
    while (!q1.empty() && !q2.empty()) {  
        if (q1.front() == q2.front()) {  
            q.push(q1.front());  
            q1.pop();  
            q2.pop();  
        }  
        else if (q1.front() < q2.front())  
            q1.pop();  
        else  
            q2.pop();  
    }  
    return q;  
}
```

```
int orden(list<int> L)
```

```
{  
    bool es_ascendente = true,  
    es_descendente = true;
```

```
    list<int>::iterator it1 = L.begin();
```

```
    list<int>::iterator it2 = L.begin();
```

```
    if (L.empty()) return 4;
```

```
    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;
```

```
}
```

Ejercicio

Usando la clase `list<T>`, construir una función que permita "duplicar" una lista intercalando alternativamente tras cada elemento en la posición i , el elemento que está en la posición $4-i-1$ ($i=0, 1, \dots, 4-1$).

Ejemplo 1:

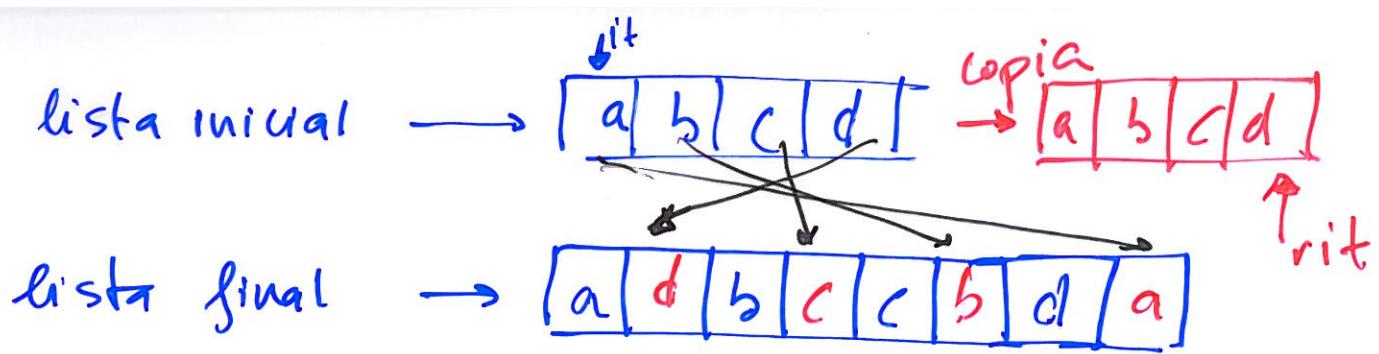
lista inicial: (a, b, c, d)

lista final: (a, **d**, b, **c**, c, **b**, d, **a**)

Ejemplo 2:

lista inicial: (1, 2, 3, 4, 5)

lista final: (1, **5**, 2, **4**, 3, **3**, 4, **2**, 5, **1**)



template<class T>

void duplicar (const list<T>& lista_inicial,
list<T>& lista_final)

{

assert (lista_inicial.size() > 0);

list<T> copia_lista(lista_inicial);

typename list<T>::const_iterator it;

typename list<T>::reverse_iterator rit;

for (it = lista_inicial.begin(); rit != copia_lista.
rbegin();
it++ = lista_inicial.end(); rit = copia_lista.
rend(),

++it; ++rit)

{

lista_final.push_back(*it);

lista_final.push_back(*rit);

}

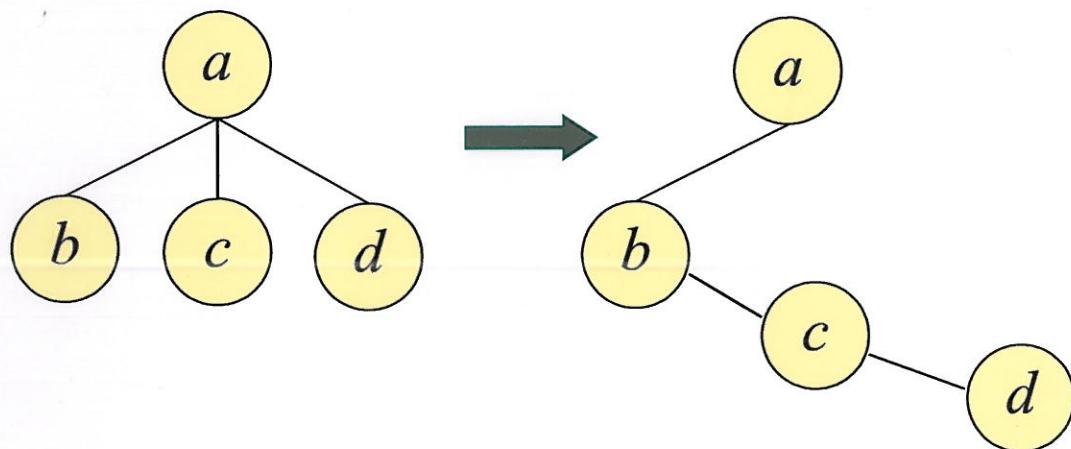
}

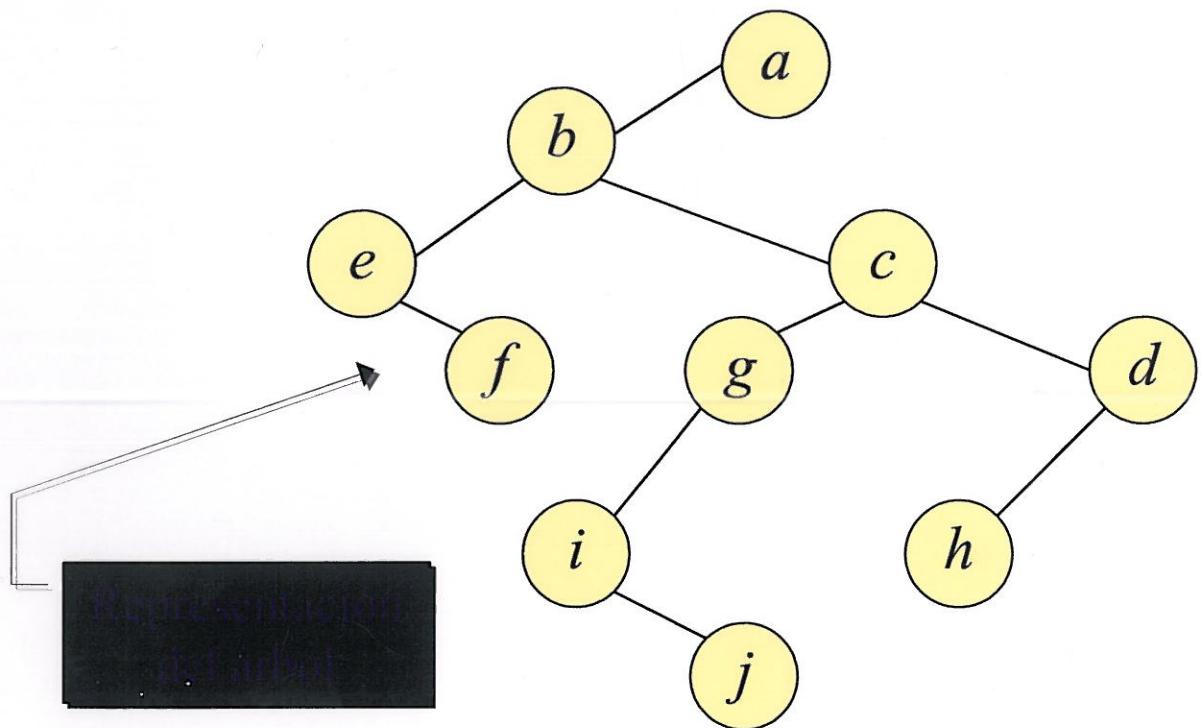
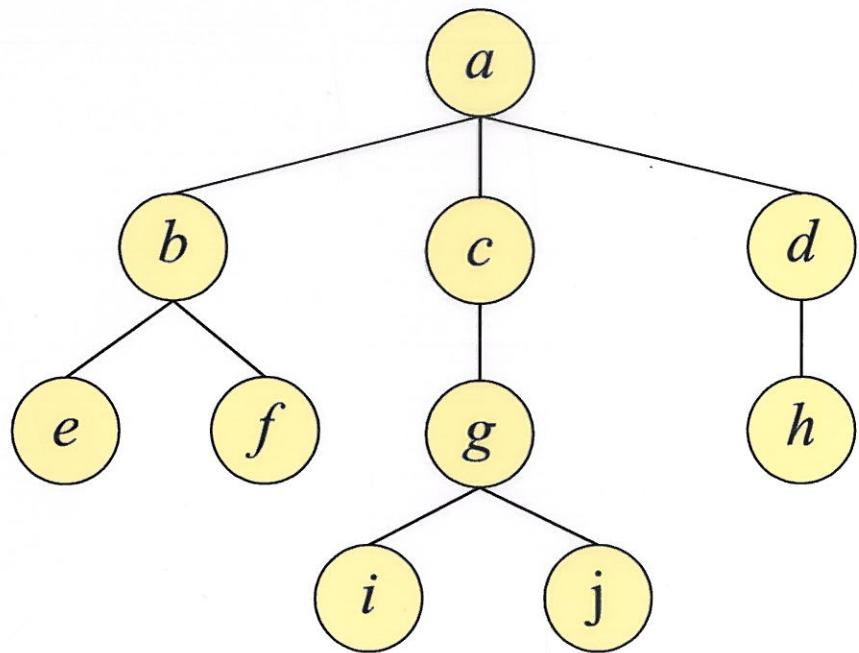
Representación de árbol n-arios: *Basada en nodos binarios*

Los nodos del árbol n-ario se representan mediante nodos binarios.

Para representar el árbol, los nodos binarios se enlazan de la forma siguiente:

- primer hijo a la izquierda (del padre)
- hermanos a la derecha (del hermano anterior)







Pre: $\boxed{a, b, c, d} \rightarrow \boxed{a, b, c, d}$

In: $b, a, c, d \rightarrow \boxed{b, c, d, a}$

Post: $\boxed{b, c, d, a} \rightarrow d, c, b, a$

Pre (General) = Pre (binario)

Post (General) = In (binario)

bool inclusion (const conjunto & (1),
const conjunto & (2))

{
for (wurst &BB::wurst iterator p = (1).begin();
p != (1).end(); ++p)

if !(2).pertenece(*p) return false;
return true;}

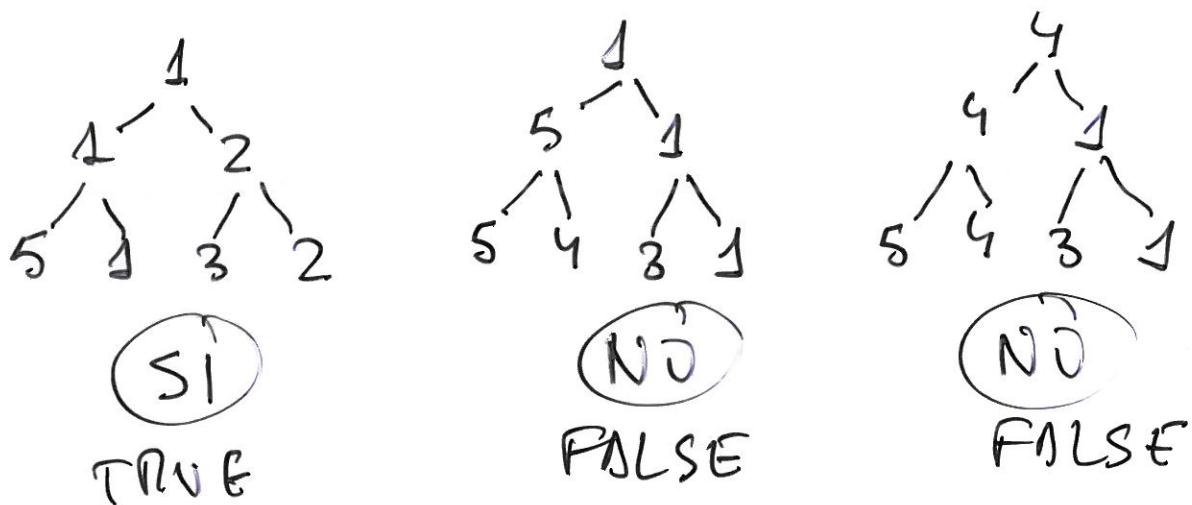
}

Se define un arbol ISWM como un AB3 en el que la altura de los subárboles es igual y dentro de cada nodo puede diferir como máximo en 2.

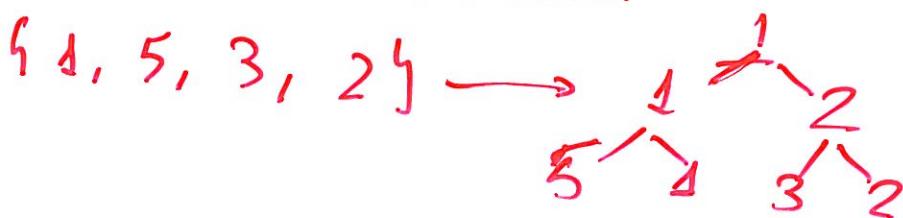
Constuir el arbol ISWM asociado al conjunto de claves: 4, 11, 10, 4, 5, 2, 6, 7, 9, 8

Un arbol de selección es un arbol binario en el que ~~que~~ cada nodo tiene la etiqueta del menor de sus hijos. Discutir una función que determine si un arbol binario T es un arbol de selección.

bool selección (biutree<Clnt> &T);

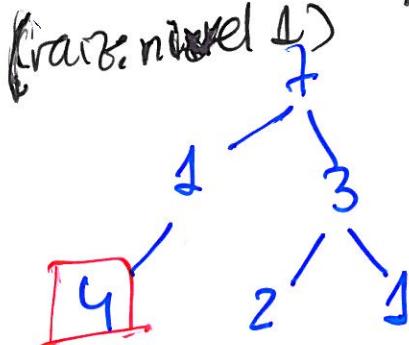


Dado un vector de enteros, construir a partir de ellos un arbol de selección



Se define la trayectoria de una hoja en un arbol binario como la suma del contenido de todos los nodos desde la raiz hasta la hoja multiplicada por el nivel en que se encuentra.

Suplementar un procedimiento que, dado un
árbol binario ~~en el que~~ la ~~el~~ trayectoria ~~de los~~ ~~los~~
devuelva la hoja con mayor trayectoria



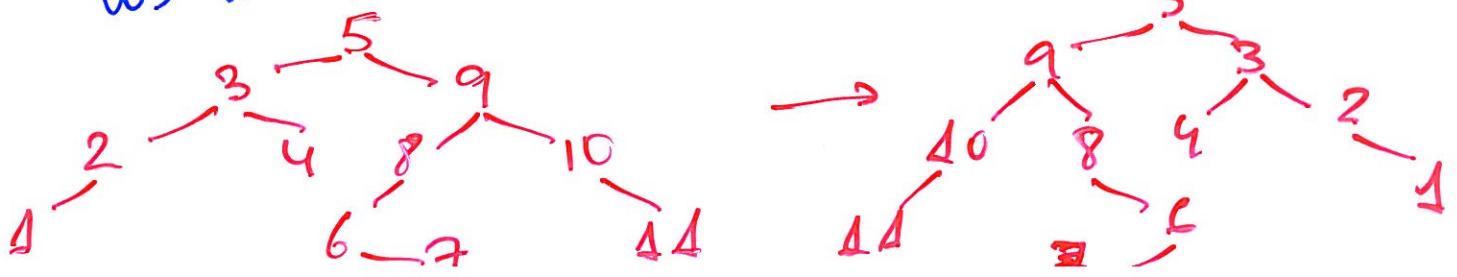
$$\text{Tray}(4) = 4 \times 3 + 1 \times 2 + 7 \times 1 = 21$$

$$\text{tray}(2) = 2 \times 3 + 3 \times 2 + 7 \times 4 = 19$$

$$\text{Tray (1)} = 1 + 3 + 3 \times 2 + 7 \times 1 = 16$$

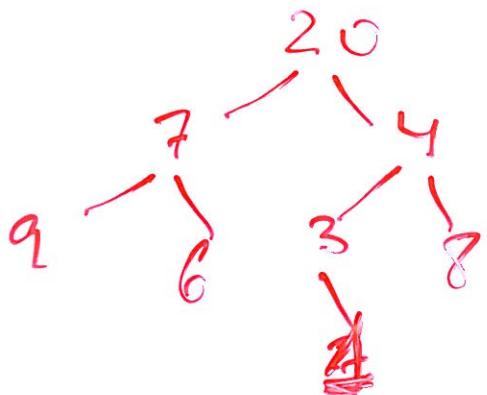
Implementar la función que dado un ABB

Implementar el ejercicio:
construya su "ordenador" y distar un algoritmo
para recorrer el "ordenador" de forma que visitante
los nodos en orden ascendente de valor



Un "recorrido guiado" sobre un árbol binario comienza listando la raíz para a continuación en cada iteración seleccionar el nodo más pequeño de entre los nodos disponibles en ese momento que no hayan sido listados independientemente de en qué orden se encuentren. Se entiende por nodo disponible aquel cuyo padre ya ha sido procesado (excluyendo la raíz).

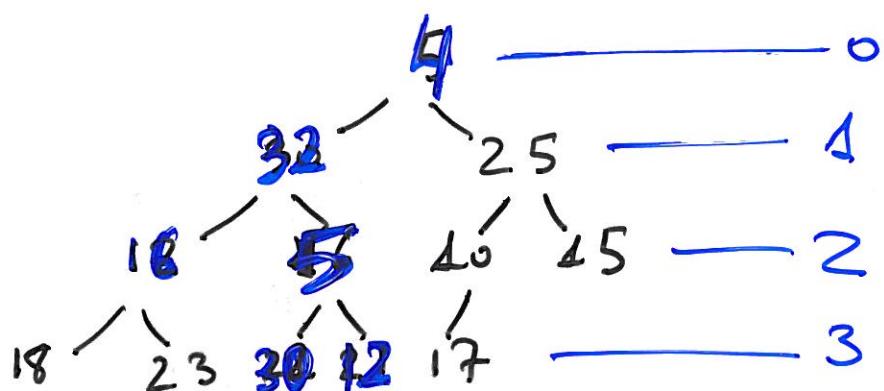
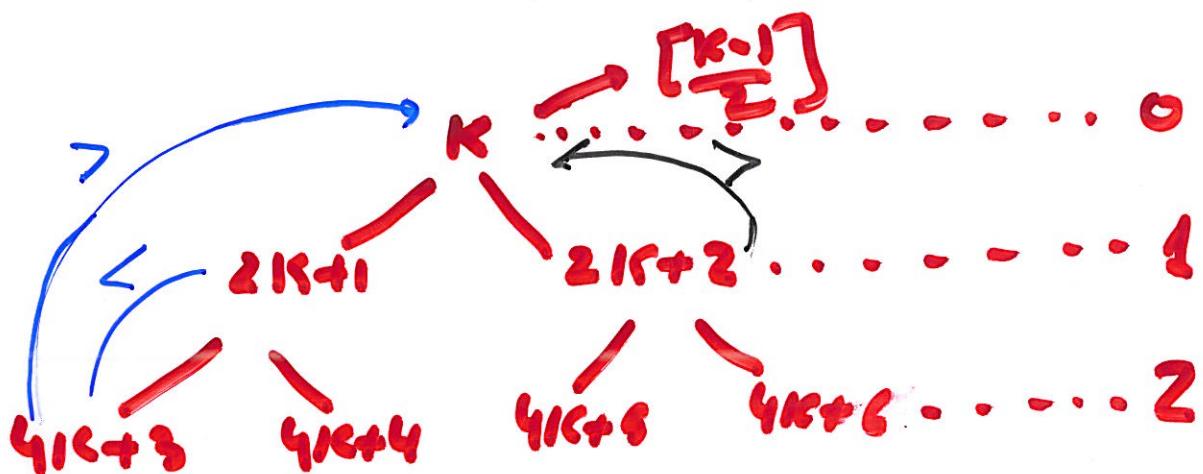
Construir una función que permita hacer un recorrido guiado en un árbol. Pueden usarse estructuras auxiliares.

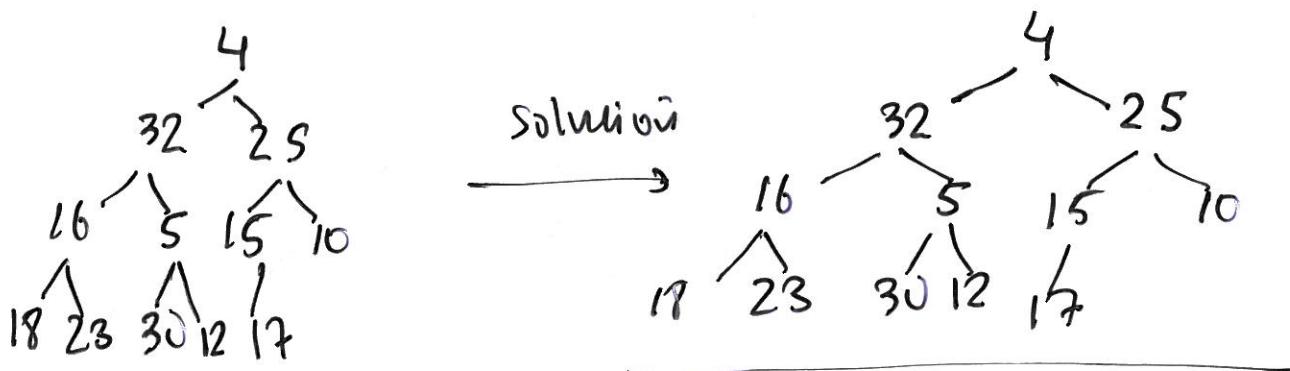
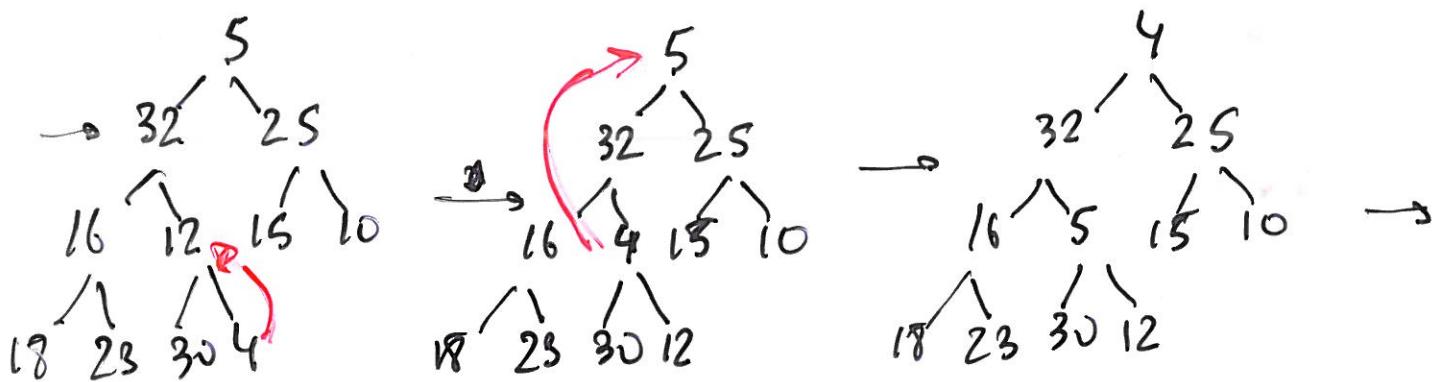
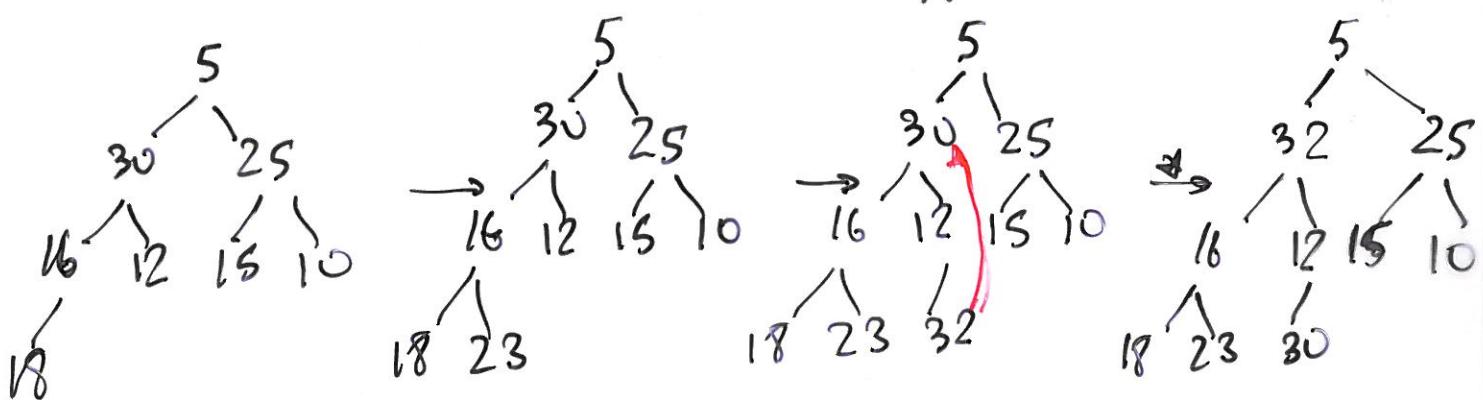
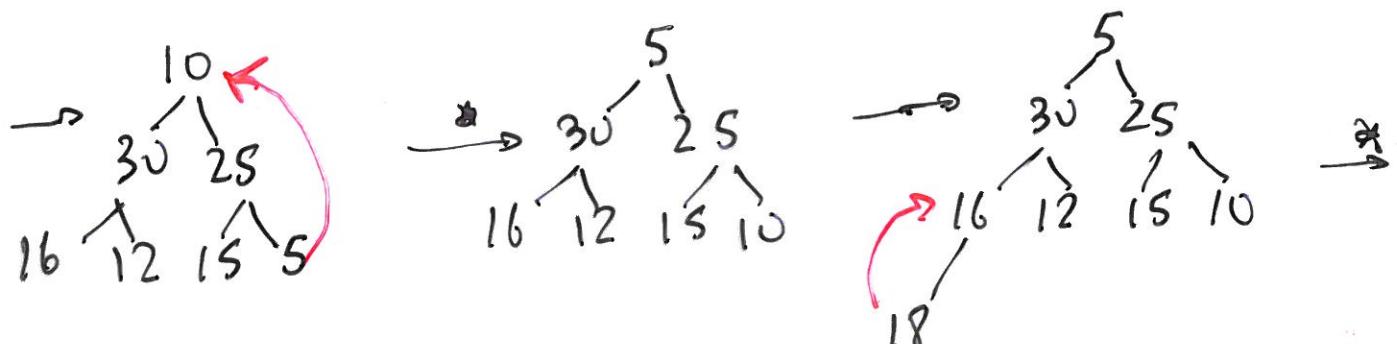
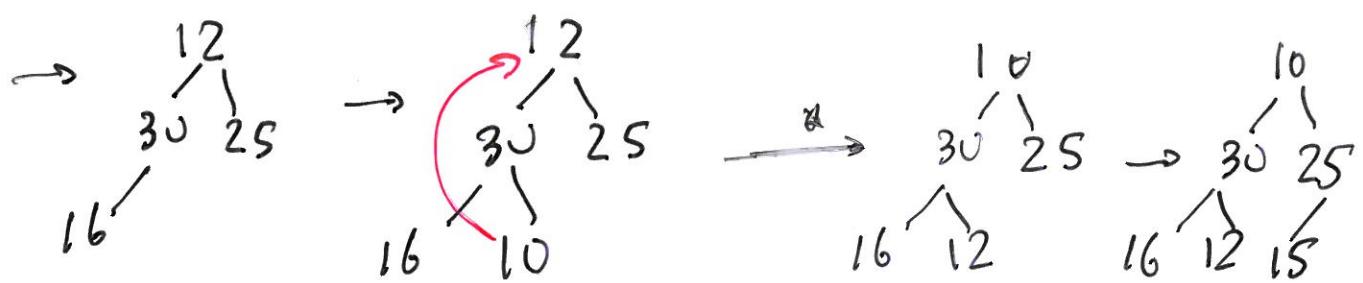
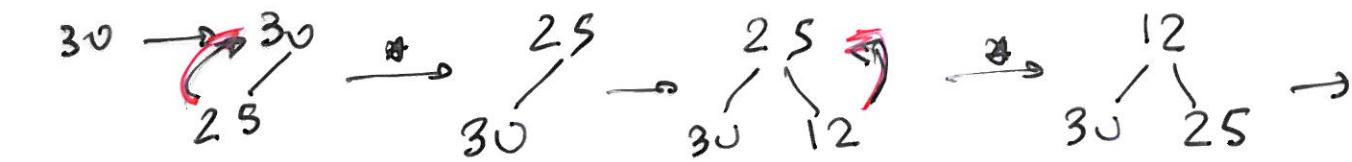


Recorrido guiado: 20, 4, 3, 1, 7, 6, 8, 9

Un heap-doble es una estructura que permite realizar las operaciones eliminar-minimo y eliminar-maximo en $O(\log_2 n)$. Tiene la propiedad de que A nodo π a profundidad par tiene una clave menor que la del padre y mayor que la del abuelo (cuando existen) y A nodo π a profundidad impar tiene una clave mayor que la del padre y menor que la del abuelo (cuando existen) con las hojas empujadas a la raíz.

Algoritmo para insertar una clave . Optéarlo para crear un heap-doble con {30, 25, 12, 16, 10, 45, 5, 18, 23, 32, 4, 17}





Sea T un árbol binario con n nodos. Se define un ~~B~~-nodo como un nodo v que cumple la condición de que el número de descendientes en el subárbol izquierdo de v difiere del número de descendientes del subárbol derecho en al menos ~~5~~. Usando el TDA binario construir una función que encuentre los 5 nodos de T

~~bool cincionodo (const bintree <int>::nodo n)~~

```
if abs (contar (n.left ()) - contar (n.right ())) >= 5  
    return true  
else return false;  
}
```

~~int contar (bintrie <int>::nodo n)~~

```
if n = null ()  
    return 0;  
else return 1 + contar (n.left ())  
      + contar (n.right ());  
}
```

~~int recuento (bintrie <int>::nodo a)~~

```
int numero = 0;  
bintrie <int>::preorder_iterator p = a.begin ();  
while (p != a.end ())  
    if cincionodo (*p) numero++;  
    ++p;
```

Dado un árbol binario de enteros, implementar una función que cuente el número de caminos cuya suma de valores de las etiquetas de los nodos que los componen sea exactamente K

int numeroCaminos (biutree<int> &ab, int k,
biutree<int>::nodo n)

{

if (n.left() == biutree<int>::null() &&
n.right() == biutree<int>::null())
if (*n == k) return 1
else return 0;

else {

int contador = 0;

if (n.left() != biutree<int>::null())

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

if (n.right() != biutree<int>::null())

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

return contador;

}

{

```
multiset<int> multi_intersection (const multiset<int>& m1, const multiset<int> m2)
```

{

```
    multiset<int>::iterator i1 = m1.begin();  
    multiset<int>::iterator i2 = m2.begin();
```

```
    while ((i1 != m1.end()) && (i2 != m2.end()))
```

```
        if (*i1 == *i2)
```

{

```
            result.insert (*i1);
```

```
            i1++;
```

```
            i2++;
```

}

```
        else if (*i1 < *i2)
```

{

```
            i1++;
```

{

```
        else i2++;
```

}

```
    return result;
```

}

$x = x + 4$

*++
++x

Son iguales cuando preceden o siguen a un operando
pero hay diferencias si se usan en una expresión

- * cuando un operador de incremento o decremento precede a su operando ($++x$, $--x$), el ++ lleva a cabo la operación antes de utilizar el valor del operando
- * cuando un operador de incremento o decremento siguen a su operando ($x++$, $x--$), el ++ usa su valor antes de incrementarlo o decrementarlo

$x = 10;$
 $y = ++x$ } y toma el valor 11 } En ambos casos
 $x = 10;$
 $y = x++$ } y toma el valor 10 } $x = 11$

La diferencia estriba en cuando cambia de valor

¿Por que $++P$ en lugar de $P++$?



+ facil de implementar:

$$++P \left\{ \begin{array}{l} P = P + 1 \\ \text{return } P \end{array} \right.$$

$$P++ \left\{ \begin{array}{l} \cancel{\text{return } P} \\ \cancel{P = P + 1} \end{array} \right. \rightarrow \left\{ \begin{array}{l} \text{tmp} = P \\ P = P + 1 \\ \text{return tmp} \end{array} \right.$$

Ejemplo:

$$\boxed{S + a++}$$

$$\rightarrow \left\{ \begin{array}{l} S + a \equiv 3 \rightarrow \text{La suma es 3} \\ a++ \equiv 3 \rightarrow a \text{ queda con valor 3} \end{array} \right.$$

$$\boxed{\begin{array}{l} S = 1 \\ a = 2 \end{array}}$$

$$\boxed{S + ++a}$$

$$\rightarrow \left\{ \begin{array}{l} * + a = 3 \rightarrow a \text{ queda con valor 3} \\ S + 3 \equiv 4 \rightarrow \text{La suma es 4} \end{array} \right.$$

Para distinguir las llamadas una vez sobrecargadas:

operator $++()$ $\rightarrow ++P$

operator $++(\text{int})$ $\rightarrow P++$

↑ Artificio técnico

ENFOQUE GREEDY

PRIMER EJEMPLO: ALGORITMO DE DIJKSTRA

SEGUNDO EJEMPLO: REPARTO Y TEMPORIZACION DE TAREAS

EJECUTAR TAREAS j_1, j_2, \dots, j_n EN TIEMPOS t_1, t_2, \dots, t_n

¿CUAL ES EL ORDEN OPTIMO DE EJECUCION?

tarea	j_1	j_2	j_3	j_4	
tiempo	15	8	3	10	
					<u>TIEMPOS DE ESPERA</u>
					$t_1 = 15$
					$t_1 + t_2 = 23$
					$t_1 + t_2 + t_3 = 26$
					$t_1 + t_2 + t_3 + t_4 = 36$
					————— 400

	j_1	j_2	j_3	j_4	
0	15	23	26	36	
					<u>TIEMPO MEDIO POR TRABAJO</u> $\frac{100}{4} = 25$ UNIDADES

SUPONDREMOS AHORA QUE ORDENAMOS LAS TAREAS

tarea	j_3	j_2	j_4	j_1	
tiempo	3	8	10	15	
					<u>TIEMPOS DE ESPERA</u>
					$t_1 = 3$
					$t_1 + t_2 = 11$
					$t_1 + t_2 + t_3 = 21$
					$t_1 + t_2 + t_3 + t_4 = 36$
					————— 74

	j_3	j_2	j_4	j_1	
0	3	4	21	36	
					<u>TIEMPO MEDIO POR TRABAJO</u> $\frac{74}{4} = 18.75$ UNIDADES

DE AHI LA FILOSOFIA DE ACTUACION DE LOS ORDENADORES A TIEMPO COMPUTADO O MULTITAREA A LA HORA DE TRABAJAR CON CUDEN DE EJECUCION DE LAS TAREAS.

ENFOQUE DIVIDE Y VENCERÁS

* PRIMER EJEMPLO: ALGORITMO QUICKSORT

* SEGUNDO EJEMPLO: MULTIPLICACIÓN DE ENTEROS DE TAMAÑO ARBITRARIO

x, y ENTEROS POSITIVOS

$x * y \rightarrow O(n^2)$ operaciones al tener que multiplicar (en principio) cada dígito de x por cada dígito de y .

EJEMPLO

$$\left. \begin{array}{l} x = 61438521 \\ y = 94736407 \end{array} \right\} xy = 5820464730934047$$

OTRA IDEA

DIVIDIR x e y EN 2 MITADES QUE CONSTAN RESP. DE LOS DÍGITOS MÁS Y MENOS SIGNIFICATIVOS

$$\left. \begin{array}{l} x \rightarrow x_l = 6143 \\ \quad \quad \quad x_r = 8521 \end{array} \right\} \Rightarrow x = x_l \cdot 10^4 + x_r$$

$$\left. \begin{array}{l} y \rightarrow y_l = 9473 \\ \quad \quad \quad y_r = 6407 \end{array} \right\} \Rightarrow y = y_l \cdot 10^4 + y_r$$

Así

$$xy = \underbrace{x_l y_l}_{\text{Multiplicaciones de tamaño } \frac{n}{2}} \cdot 10^8 + (\underbrace{x_l y_r + x_r y_l}_{\text{Sumas y productos por } 10^4}) \cdot 10^4 + \underbrace{x_r y_r}_{\text{Sumas y productos por } 10^8}$$

$\rightarrow O(n^2)$

$\rightarrow O(n)$

Asi: $T(n) = 4T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n^2)$!! ¿QUÉ ASUMIMOS?

RECORDEMOS:

$$xy = x_e y_e \cdot 10^8 + (x_e y_r + x_r y_e) \cdot 10^4 + x_r y_r$$

PERO:

$$x_e y_r + x_r y_e = \underbrace{(x_e - x_r)(y_r - y_e)}_{\text{+ productos nuevos}} + \underbrace{x_e y_e + x_r y_r}_{\text{productos ya calculados}}$$

ASÍ:

$$xy = \underline{x_e y_e} \cdot 10^8 + [(\underline{x_e - x_r})(\underline{y_r - y_e}) + \underline{x_e y_e + x_r y_r}] \cdot 10^4 + \underline{x_r y_r}$$

ASI:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$



$$T(n) = O(n^{\log_2 3}) \approx O(n^{1.59})$$



GANAMOS EN EFICIENCIA

PERDIMOS AL $O(n^2)$ INICIAL

VEAMOS UN EJEMPLO:

$$\begin{array}{r|rr} & \overbrace{x_e} & \overbrace{x_r} \\ x = & 6 & 1 & 4 & 3 & 8 & 5 & 2 & 1 \\ & \underline{y_e} & \underline{y_r} \end{array}$$
$$y = \underline{9} \underline{4} \underline{7} \underline{3} \underline{6} \underline{4} \underline{0} \underline{7}$$

Y LA TABLA DE CALCULOS SERIA:

FUNCION	VALOR	COMPLEJIDAD
x_e	6143	—
x_r	8521	—
y_e	9473	—
y_r	6408	—
$d_1 = x_e - x_r$	-2378	$O(n)$
$d_2 = y_r - y_e$	-3066	$O(n)$
$x_e y_e$	58192639	$T(n/2)$
$x_r y_r$	54594047	$T(n/2)$
$d_1 d_2$	7290948	$T(n/2)$
$d_3 = d_1 d_2 + x_e y_e + x_r y_r$	120077634	$O(m)$
$x_r y_r$	54594047	—
$d_3 \cdot 10^4$	1200776340000	$O(n)$
$x_e y_e \cdot 10^8$	58192639000000000	$O(n)$
$x_e y_e \cdot 10^8 + d_3 \cdot 10^4 + x_r y_r$	5820464730934047	$O(n)$
xy	5820464730934047	$3T(n/2) + O(n)$

ITERATIVIDAD VERSUS RECURSIVIDAD

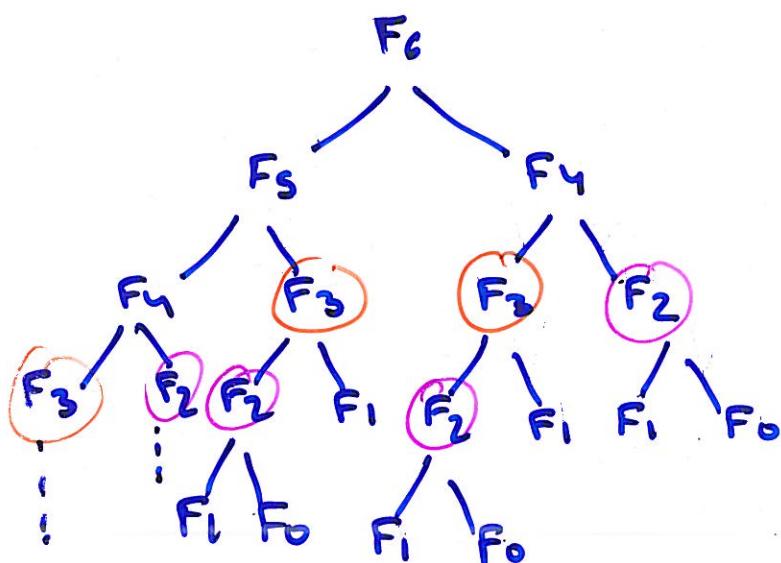
EJEMPLO

SUCESION DE FIBONACCI.

```

function fib (n: integer): integer;
begin
  if (n=0) or (n=1)
  then fib:=1
  else fib:=fib(n-1)+fib(n-2);
end;
  
```

(VERSION RECURSIVA)



F_n llama a F_{n-1} y F_{n-2}

↓

1. llama a F_{n-2} y F_{n-3}

2. llamadas distintas

↓

SINFILIGRINA

ES PREFERIBLE LA VERSION ITERATIVA.