

## ORDENAR

$$\boxed{\sqrt{n}, n^3+1, \frac{n^4}{n^2+1}, n \log_2(n^2), n \log_2 \log_2(n^2), 3^{\log_2 n}, 3^n, 2^{100}, n+100}$$

Solución

$$\boxed{2^{100}, \sqrt{n}, n+100, 3^{\log_2 n}, n \log_2 \log_2 n^2, n \log_2 n^2, \frac{n^4}{n^2+1}, n^3+1, 3^n, n^{\log_2 3} = n^{1.58}}$$

$$\boxed{\left( \frac{n(n+1)}{2} - \frac{n^2}{2} \right), n \log_2 n, n^2 \log_2 \log_2(n), \frac{n^4}{n^2+1}, (n^3+n)(11)^n}$$

Solución

$$\boxed{\left( \frac{n(n+1)}{2} - \frac{n^2}{2} \right), n \log_2 n, \frac{n^4}{n^2+1}, n^2 \log_2 \log_2 n, n^3+n, (1.1)^n}$$

$$\log_2(2n \log_2(n)), n \log_2(\sqrt{n}), n\sqrt{n}, 2^{\log_2 n},$$

$$(1.000001)^n, 2^{2\log_2(n)}, n^2 2^{3\log_2 n}$$

$$\log_2(2n \log_2 n) = \log_2 2 + \log_2 n + \log_2 \log_2 n \rightarrow O(\log_2 n)$$

$$n \log_2 \sqrt{n} = \frac{1}{2} n \log_2 n \rightarrow O(n \log_2 n)$$

$$n\sqrt{n} \rightarrow O(n\sqrt{n})$$

$$2^{\log_2 n} = n \rightarrow O(n)$$

$$2^{2\log_2 n} = 2^{\log_2 n^2} = n^2 \rightarrow O(n^2)$$

$$n^2 2^{3\log_2 n} = n^2 2^{\log_2 n^3} = n^2 n^3 = n^5 \rightarrow O(n^5)$$

$$(1.000001)^n \rightarrow O(1.000001^n)$$

Por tanto:

$$\log_2(2n \log_2 n), 2^{\log_2 n}, n \log_2 \sqrt{n}, n\sqrt{n},$$

$$2^{2\log_2 n}, n^2 2^{3\log_2 n}, (1.000001)^n$$

## Ejercicio

~~1~~,  $2^{\log_2 n}$ ,  $2^{2 \log_2 n}$ ,  $\log_2(\sqrt{n})$ ,  $n^2 2^{3 \log_2 n}$ ,  ~~$2^n$~~

Solución:

$$2^{\log_2 n} = n \rightarrow n$$

$$2^{2 \log_2 n} = (2^{\log_2 n})^2 = n^2 \rightarrow n^2$$

$$\log_2 \sqrt{n} = \log_2(n)^{1/2} = \frac{1}{2} \log_2 n \rightarrow \log_2 n$$

$$n^2 2^{3 \log_2 n} = n^2 (2^{\log_2 n})^3 = n^2 \cdot n^3 = n^5 \rightarrow n^5$$

$$2 \rightarrow 1$$

$$1^n \rightarrow 1$$

Por tanto:

2,  $1^n$ ,  $\log_2 \sqrt{n}$ ,  $2^{\log_2 n}$ ,  $2^{2 \log_2 n}$ ,  $n^2 2^{3 \log_2 n}$

~~$4^{\log_2 n}$ ,  $\log_2^2(n)$ ,  $2^4 \log_2(n)$ ,  $9n^{1/4}$ ,  $2^n + \frac{1}{5}n^2$ ,  $\frac{2}{5}n^2 + n^4$~~

$2^4 \log_2 n$ ,  $\log_2^2 n$ ,  $9n^{1/4}$ ,  $\frac{4^{\log_2 n}}{n^2}$ ,  $n^4 + \frac{2}{5}n^2$ ,  $2^n + \frac{1}{5}n^2$

$n^{1.5}$ ,  $\sqrt{n}$ ,  $n \log_2 \log_2 n$ ,  $n^2$ ,  $2^{n/2}$ ,  $n^2 \log_2 n$ ,  $n 2^n$

$\sqrt{n}$ ,  $n \log_2 \log_2 n$ ,  $n^{1.5}$ ,  $n^2$ ,  $n^2 \log_2 n$ ,  $2^{n/2}$ ,  $n 2^n$

$n$ ,  $n^{1.5}$ ,  $n \log_2 n$ ,  $\sqrt{n}$ ,  $n \log_2 \log_2(n)$ ,  $n^2$ ,  $2^{n/2}$ ,  
 $37$ ,  $n^2 \log_2 n$ ,  $n 2^n$ ,  $(1.0000001)^n$

~~37,  $\sqrt{n}$ ,  $n$ ,  $n \log_2 \log_2 n$ ,  $n \log_2 n$ ,  $n^{1.5}$ ,  $n^2$ ,~~  
 $n^2 \log_2 n$ ,  $(1.0000001)^n$ ,  $2^{n/2}$ ,  $n 2^n$

void eliminar (vector L, int x)

{

int aux, p;  $x = \text{fim}(L)$

for ( $p = \text{primero}(L); p \neq \text{fim}(L);$ )

}

aux = elemento ( $p, L$ ),

if ( $\text{aux} == x$ )

borrar ( $p, L$ )

else  $p++$

}

}

for ( $K=1; K \leq n; K \neq 2$ )

for ( $j=1; j \leq K; j++$ )

sum2++

$$\sum_{K=1}^{\log_2 n} \sum_{j=1}^K 1 = \sum_{K=1}^{\log_2 n} K = \sum_{2^i=1}^{\log_2 n} 2^i = \frac{2^{\log_2 n} \cdot 2 - 1}{2 - 1}$$

$(i=0)$

$\approx 2n - 1 \longrightarrow O(n)$

for ( $K=1; K \leq n \neq K \neq 2$ )

for ( $j=1; j \leq n; j++$ )

sum2++

$$\sum_{K=1}^{\log_2 n} \sum_{j=1}^K 1 = \sum_{K=1}^{\log_2 n} n = n \log_2 n$$

$O(n \log_2 n)$

int n, j; int r=1; int x=0;

do {

j = 1;

while (j <= n)

{ j = j \* 2;

x++;

j++;

$O(n \log_2 n)$

} while (j <= n)

—————

int n, j; int r=2; int x=0;

do {

j = 1;

while (j <= r)

{ j = j \* 2;

x++;

j++

} while (r <= n);

—————

$$O\left(\sum_{i=1}^n \log_2 i\right) = O(\log_2 2 + \log_2 3 + \dots + \log_2 n) = \\ = O(\log_2 (2 \cdot 3 \cdot \dots \cdot n)) = O(\log_2 n!) \leq O(n \log_2 n)$$

for ( $i=1; i \leq n; i+=4$ )  $\rightarrow \frac{n}{4}$   
 for ( $j=1; j \leq n; j+=\frac{n}{4}$ )  $\rightarrow 4$   
 for ( $k=1; k \leq n; k*=2$ )  $\rightarrow \log_2 n$   
 $x++ \rightarrow O(1)$

$$\frac{n}{4} \cdot 4 \cdot \log_2 n \cdot 1 \rightarrow O(n \log_2 n)$$

for ( $i=1; i \leq n; i+=4$ )  
 for ( $j=1; j \leq n; j+=\frac{n}{4}$ )  
 for ( $k=1; k \leq n; k*=2$ ).  
 $x++$

Un videoclub está desarrollando una aplicación que permita un acceso rápido a las películas de las que disponen. No tiene claro qué estructura de datos seguir para almacenar los registros (comprendidos por código, título, año ...). La **búsqueda** se hace por el título de cada película. ~~Además~~ ~~se~~ ~~usa~~

Para la **inscripción y bono** de películas, se usa un código único asociado a cada película. Además se necesita una función para **suprimir** de forma ordenada todas las películas.

Analizar la eficiencia de las 4 operaciones si se usa como estructura de datos:

- 1) **vector ordenado**
- 2) **lista ordenada**
- 3) **ABR**
- 4) **DVL**
- 5) **Tabla hash (abierto)**

En función del resultado del análisis, decidir qué estructura de datos resuelve mejor el problema

	Vector ordenado	Lista ordenada	DBB	DVL	Tabla hash
buscar	$O(\log_2 n)$	$O(n) [\log_2 n]$	$O(n)$	$O(\log_2 n)$	$O(n) / O(1)$
insertar	$O(n)$	$O(n)$	$O(n)$	$O(\log_2 n)$	$O(1)$
borrar	$O(n)$	$O(n)$	$O(n)$	$O(\log_2 n)$	$O(n)$
Imprimir	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$[O(n \log_2 n)]$

Orderen:  $2^{\log_2 n}$ ,  $2^{2\log_2 n}$ ,  $\log_2(\sqrt{n})$ ,  $n^2 2^{3\log_2 n}$ ,  $2^{\frac{\log_2 n}{2}}$

$$2^{\log_2 n} = n \longrightarrow n$$

$$2^{2\log_2 n} = (2^{\log_2 n})^2 = n^2 \longrightarrow n^2$$

$$\log_2 \sqrt{n} = \log_2(n)^{1/2} = \frac{1}{2} \log_2 n \longrightarrow \log_2 n$$

$$n^2 2^{3\log_2 n} = n^2 (2^{\log_2 n})^3 = n^2 n^3 = n^5 \longrightarrow n^5$$

$$2^n \longrightarrow 1$$

$1^n$ ,  $\log_2 \sqrt{n}$ ,  $2^{\log_2 n}$ ,  $2^{2\log_2 n}$ ,  $n^2 2^{3\log_2 n}$

int calcularEficiencia (bool existe)

'  
int sum2 = 0; int i, j, n;

if (existe)

for (k=4; k <= n; k\*=2)

for (j=4; j <= k; j++)

sum2++;

else

for (k=4; k <= n; k\*=2)

for (j=4; j <= n; j++)

sum2++;

return sum2;

$$\sum_{k=4}^{\log_2 n} \sum_{j=1}^{2^k} 1 = \sum_{k=1}^{\log_2 n} k = \sum_{2^i=4}^{\log_2 n} 2^i = 2n - 4 [O(n)]$$

$$\sum_{k=1}^{\log_2 n} \sum_{j=1}^n 1 = \sum_{k=1}^{\log_2 n} n = n \sum_{k=1}^{\log_2 n} 1 = n \log_2 n [O(n \log_2 n)]$$

$$O(\max\{n, n \log_2 n\}) \rightarrow O(n \log_2 n)$$

$n=64$      $\overbrace{1 \ 2 \ 4 \ 8 \ 16 \ 32}^K \ 64$   $\rightarrow 2n-1 \quad \left\{ \begin{array}{l} 2n-1 \\ 63 \end{array} \right.$

$j \rightarrow 1+2+4+8+16+32 \rightarrow n-1$

Se dispone de un vector de gran tamaño donde se almacenan claves enteras ordenadas que pueden aparecer de forma consecutiva un número indefinido de veces. P. ej. 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 8, 8, 8, ...

Construir una función que dada una posición  $i$  del vector original devuelva el valor almacenado en esa posición tras diseñar una representación capaz de almacenar los datos del vector con un espacio proporcional a las secuencias existentes.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
1	1	1	1	1	4	4	4	5	5	5	5	5	8	8	8	8	.	.	.	...

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
v	1	5	4	3	5	5	8	4	7	..	..	..	..	..	..	..	..	..	..	...

int valor\_pos (int i )

{ j=0;

while ( i >= 0)

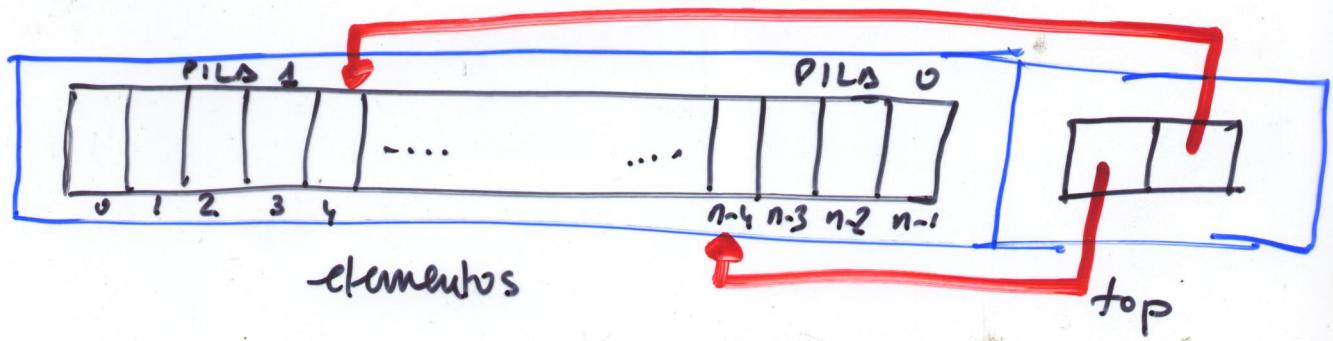
    { i = i - v[j].num

    ++j

    return j-1

}

## Pila doble



private:

T \* elementos;

int top[2];

-----

Template <class T>

void Pila<T>::pop(int indicepila)

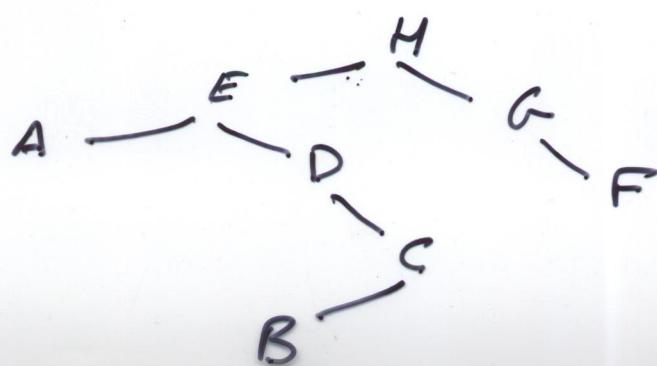
}

top[indicepila] = top[indicepila]  
- (indicepila ? 1 : -1);

S

Post: A B C D E F G H

In: A E D B C H G F



void Pila:: quitar()

```
    assert (nElem > 0); O(1)
    nElem--;
    if (nElem <= reservados / 4) Resize (reservados / 2);
}
```

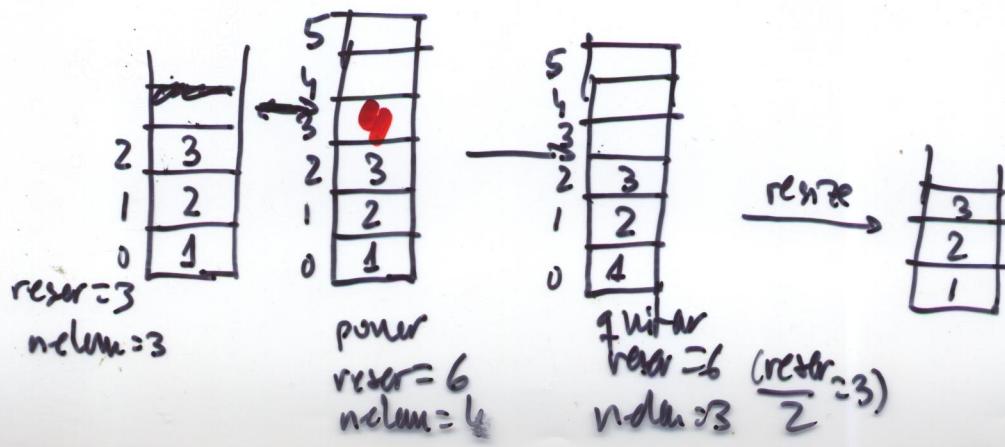
- Comportamiento de operaciones consecutivas poner/quitar  
cuando el vector está completo:

\* Al poner un elemento con el vector lleno, se redimensiona al doble. Si inmediatamente después quitamos un elemento, el vector tendría más de la mitad del tamaño libre y volvería a redimensionarse a la mitad

\* Si continuamos con operaciones poner - quitar - poner - quitar - etc., cada una tendría un costo  $O(n)$

void Pila:: Poner (T c)

```
    if (nElem == reservados) Resize (2 * reservados);
    datos [nElem] = c;
    nElem--;
}
```



Se dice que una pila es inversa a una cola cuando el listado de los elementos de la pila coincide con el listado de los de la cola en orden inverso.

Usando las clases stack y queue de la STL diseñar una función para determinar si una pila y una cola dada son inversas

template <class T>

bool sonInversas(<stack<T>, s, queue<T> q)

{

if (s.size() != q.size()) return false;

else {

stack<T> aux;

while (!q.empty())

{ aux.push(q.front());

q.pop();

}

while (!aux.empty())

{ if (aux.top() != s.top())

return false;

aux.pop();

s.pop();

}

return true;

}

}

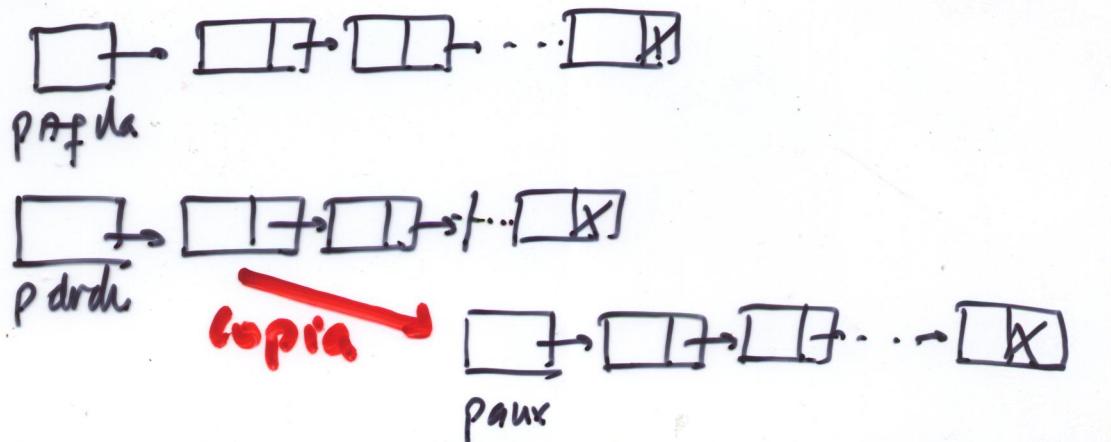
Pila & Pila:: operator = (const Pila & p)

↳ Pila paux(p);  
celdaPila \*aux;  
aux = this → primera;  
this → primera = paux.primera;  
paux.primera = aux;  
return \*this;

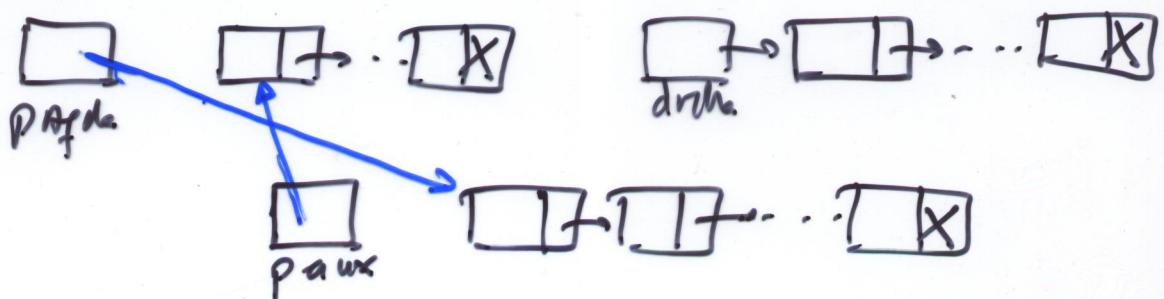
y

haremos pizqda = pdrecha (asignacion de 2 pilas)

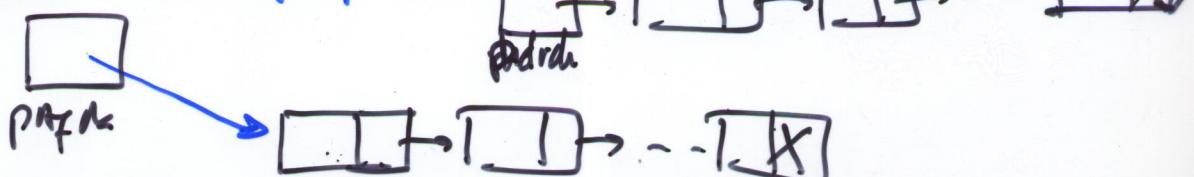
Paso 1 (crea paux como copia de la pila drcha)



Paso 2 (Intercambia los contenidos de paux y pizqda)



Paso 3 (Destruye paux)



(Si destruye paux se destruyen los id's de la pila pizqda  
y pizqda queda con un conjunto de celdas idéntico a pdrcha)

Usando el TDS lista, construir una función que tenga como entrada dos listas y devuelva 1, si la primera está contenida en la segunda (tienen los mismos elementos, consecutivos y en el mismo orden) y 0 en otro caso. P. ej.:  $l_1 = \langle 1, 2, 3 \rangle$

$$l_2 = \langle 0, 1, 2, 3 \rangle \quad \left. \begin{array}{l} \\ \end{array} \right\} \textcircled{1}$$

$$l_1 = \langle 1, 2, 3 \rangle \quad \left. \begin{array}{l} \\ \end{array} \right\} \textcircled{2}$$

bool secuencia (list<int> & l1, list<int> & l2)

' list <int> iterador :: p, q;

int x;

x = l1.front();

p = l1.begin();

q = l2.find (l2.begin(), l2.end(), x),

while ( $p \neq l_1.end()$ )

if ( $*p \neq *q$ )

return false;

else {  
++p;

++q;

'

return true;

5

void invertir (cola d())

{  
    while (! c.empty())

        iut aux = c.front();

        c.pop();

        invertir(c);

        c.push(aux);

}

—————

bool simetria (pila d P, cola d l)

{  
    invertir(l);

    while (! c.empty())

        if ((c.front()) != P.front())

            return false;

    else {  
        c.pop();

        P.pop();

}

    return true;

}

Usando el TDS `list<int>`, construir una función  
`void agrupar_elementos (list<int> &entrada, int k)`  
que agrupe de forma consecutiva en la lista de  
entrada todas las apariciones del elemento `k` en la  
lista, a partir de la primera ocurrancia. P. ej.  
si entrada = `{1, 3, 4, 1, 4}` y `k=1` entonces  
entrada = `{1, 1, 3, 4, 4}`  
• si entrada = `{3, 1, 4, 1, 4, 1, 1}` y `k=1`  
entonces entrada = `{3, 1, 1, 1, 4, 4}`

`void agrupar_elementos (list<int> &entrada, int k)`

```

    list<int> iterator :: p, q;
    p = entrada.find(entrada.begin(), entrada.end(),
                      k),
    q = ++p;
    while (q != entrada.end())
        if (*q == k)
            entrada.insert(p, k),
            q = entrada.erase(q),
        else
            ++q;
    }
```

Usando el TDS `list<T>`, construir una función template `<class T> void dividir-lista (list<T> &lista, T lc)`

que agrupe en la primera parte de la lista los elementos menores que le g an la segunda los mayores o iguales. Han de usarse iteradores, no te permiten estructura auxiliar, no te puedes modificar el tamaño de la lista y la función (va de tar O(n))

Ejemplo,

Entrada = {1, 3, 4, 14, 11, 9, 7, 16, 25, 19, 7, 8, 9}   
lc = 8

Salida = {1, 3, 4, 7, 7, 9, 11, 16, 25, 19, 14, 8, 9}}

typename list<T> :: iterator it = lista.begin();  
while (it != lista.end())

    if (\*it > lc)

        lista.push\_back(\*it);

        it = lista.erase(it);

    else ++it;

}

## Algoritmo.

1. buscar primera aparición de  $k$  en entrada, y almacenar su posición.
2. Recorrer entrada buscando apariciones de  $k$ 
  - 2.1. cuando se encuentre una:
    - 2.1.1. insertar elemento  $a$  en  $P$
    - 2.1.2. borrar elemento.

void agrupar\_elementos (list<int> &entrada,  
int  $k$ )

```
'list<int> iterator::P, q;  
P = entrada.find(entrada.begin(),  
entrada.end(), k);  
q = ++P;  
while (q != entrada.end())  
    if (*q == k)  
        'entrada.insert(P, k);  
        q = entrada.erase(q);  
    else ++q;
```

Dada una lista de enteros con elementos repetidos, diseñar (usando el TDA lista) una función que construya a partir de ella una lista ordenada de listas, de forma que en la lista resultado los elementos iguales, se agrupen en la misma sublista. Ejemplo:

Entrada = {1, 3, 4, 5, 6, 3, 2, 4, 4, 5, 5, 3, 2, 7}  
Salida = {{1, 4, 1, 6}, {2}, {3, 3}, {4, 4}, {5, 5, 5}, {6}, {7}}

```
list < list < int > > final;
list < int > l1, l2;
list < list < int > > iterator::q; list < int > iterator::p;
l1.sort();
while (!l1.empty())
{
    q = final.begin();
    p = l1.begin(); elem = *p;
    while (*p == elem)
        {
            l2.insert(l2.begin(), elem);
            p = l1.erase(p);
        }
    final.insert(q, l2);
    l2.clear();
    ++q;
}
```

```
list<pair<int, float>> lista;
```

```
pair<int, float> par;
```

```
for (int i=0; i<5; ++i)
```

```
{  
    p.first = i;
```

```
    p.second = 0.5 * i;
```

```
    lista.push_back(par);
```

```
}
```

```
list<pair<int, float>>::iterator p;
```

```
for (p = lista.begin(); p != lista.end(); ++p)
```

```
cout << *p;
```

```
for (p = lista.begin(); p != lista.end(); ++p)
```

```
if ((*p).second < 3.0 &&
```

```
(*p).first > 3)
```

```
lista.pop(p);
```

```
bool probable (const ArbolBinario<float> & A)
```

```
    {  
        bool res=false;  
        suceso_probable (A, A.raiz(), res)  
        return res;  
    }
```

```
bool suceso (const ArbolBinario<float> & A,  
             ArbolBinario<float>::Nodo n)
```

```
    {  
        float suceso=1;  
        while (n!=A.raiz())  
            {  
                suceso=suceso * A.etiqueta(n);  
                n=A.padre(n);  
            }  
    }
```

```
        if (suceso > 0.5) return true;  
        return false;
```

```
suceso_probable (const ArbolBinario<float> & A,  
                  ArbolBinario<float>::Nodo n, bool & prob)
```

```
    {  
        if (A.izqda(n)==0) && A.drdg(n)==0)  
            prob=suceso (A, n);
```

```
        if (n!=0 & & !prob)
```

```
            suceso_probable (A, A.izqda(n), prob);
```

```
            suceso_probable (A, A.drdg(n), prob);
```

```
    }
```

```
}
```

## Algoritmo

ArbolBinario<ItemType>:: Nodo AMI (const ArbolBinario<ItemType> & a,

const ArbolBinario<ItemType>:: Nodo v,

const ArbolBinario<ItemType>:: Nodo w )

}

ArbolBinario<ItemType>:: Nodo p;

for ( p = a.padre(v); p != nodo\_nulo; p = a.padre( p ) )

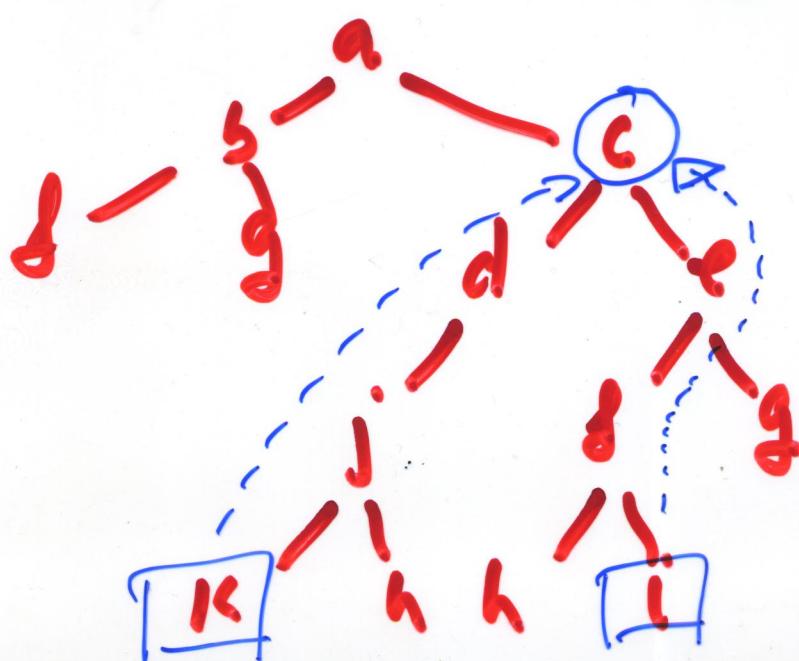
while ( w != nodo\_nulo )

if ( p == a.padre(w) )

return a.padre(w);

else w = a.padre(w);

}



Sea A un árbol binario con n nodos. Se define el ancestro común más cercano (AMC) entre 2 nodos r y w como el ancestro de mayor profundidad que tiene tanto a r como a w como descendientes. P.ej entiende como caso extremo que un nodo es descendiente de si mismo). Definir una función que tenga como entrada un árbol binario de arboles y 2 nodos r y w y como salida el  $\text{AMC}(r, w)$ .

nodo  $\text{AMC}(\text{ArbolBinario } \alpha, \text{nodo } r, \text{nodo } w)$

↳

vector <nodo> rnodos;

while ( $r \rightarrow \text{padre}() \neq 0$ ) ↳

rnodos.insert ( $r \rightarrow \text{padre}()$ );

$r = r \rightarrow \text{padre}();$

↳

bool find = false;

vector <nodo>::iterator it;

while (find  $\& w \rightarrow \text{padre}() \neq 0$ ) ↳

$it = \text{rnodos}.find(w \rightarrow \text{padre}());$

if ( $it \neq \text{rnodos}.end()$ )

    find = true;

else  $w = w \rightarrow \text{padre}();$

if (find) return \*it;

else return nodo();

↳

Construir una función que devuelva para un ABRS,  
el nivel del nodo con mayor valor de la clave

ret. nivel (ABRS < T > A)

↳ Nodo u;

n = A.raiz();

contador = 0;

while (n != nodonulo)

↳ n = A.dcha(n),

contador++;

↳  
return contador;

}

```
bool sesgado (ArbolBinario<T> & A, nodo n)
{
    if (n != 0)
        {
            if (n.izq().esquierdo () >=
                n.drch().esquierdo ())
                return false;
            else
                return sesgado (A, n.izq ()) &&
                    sesgado (A, n.drch ());
        }
    else
        return true;
}
```

nodo AMC (Arbol Binario de T) para, nodo v,  
nodo w)

1 si todos los antecesores de v &/  
vector <nodo> vNodos;

while (v->padre() != 0)

{  
vNodos. insert(v->padre);

r2 v->padre();

{

bool find = false;

while (find == false && w->padre() != 0)

{  
vector <nodo>:: iterator it;

it = vNodos. find(w->padre());

if (it != vNodos. end())

find = true;

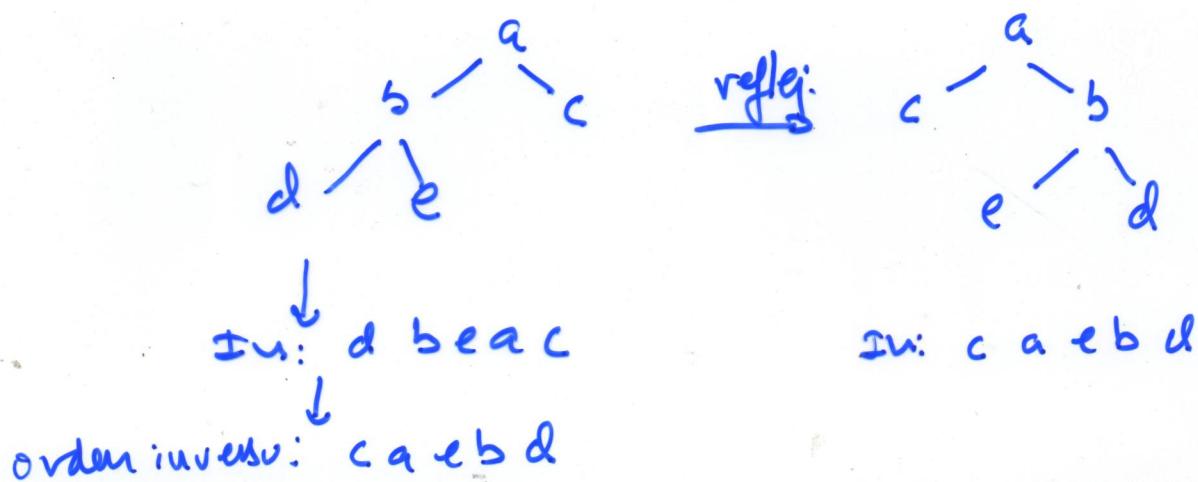
{ else w = w->padre();

{ if (find)

return \*it;

, else return nodo();

- Sea una operación  $f$  que opera sobre un árbol binario  $A$  y que para un recorrido  $r \in \{ \text{Pre, In, Post} \}$  procesa los nodos en orden inverso al que indica  $r$
- sea una operación  $g$  que procesa los nodos en el mismo orden que indica el recorrido  $r \in \{ \text{Pre, In, Post} \}$  sobre el reflejado de  $A$   
¿Para qué recorrido  $r$  es cierto que  $f(r) = g(r)$ ?



Evaluar la expresión <sup>en</sup> postfijo:

$$(x * y) - [(z + w) / (x + y)^x]$$

$$x = y = z = w = 4$$

Solución: 0

Usando el TDS ABB construir una función que tenga como entrada un ABB <int> y que devuelva el número de nodos con un valor de etiqueta dentro de un intervalo  $[a, b]$   $a, b$  enteros

```
int contar (int, a, b, ABB<int> T)
{
    if (T.raiz() == 0) return
    else if (T.etiqueta(T.raiz()) > b)
        return contar (a, b, T.raiz(T.raiz(), T))
    else if (T.etiqueta(T.raiz()) < a)
        return contar (a, b,
                        T.derecha(T.raiz(), T)),
    else return 1 + contar (a, b,
                            T.raiz(T.raiz(), T))
                            + contar (a, b,
                            T.derecha(T.raiz(), T)),
```

g

void inorden no recursivo (Arbol binario <link> A)

{ static <Nodo> piladenodos;

Nodo p = A.root();

white (p != nodonulo)

{ white (p != nodonulo)

{ if (!A.hijodrcha (p))

piladenodos.push (!A.hijodrcha (p));

piladenodos.push (p);

p = A.hijozqda (p);

} //segundo while

pop de un  
nodo sin  
hijo izda

p = piladenodos.pop();  
while (!piladenodos.empty ()) {  
if (cout << A.etiqueta (p);  
p = piladenodos.pop ();

} //tercer while

cout << A.etiqueta (p);

if (!piladenodos.empty ())

p = piladenodos.pop();

else p = nodonulo;

} //primer white (p != nodonulo)

lo hay) } // procedimiento.

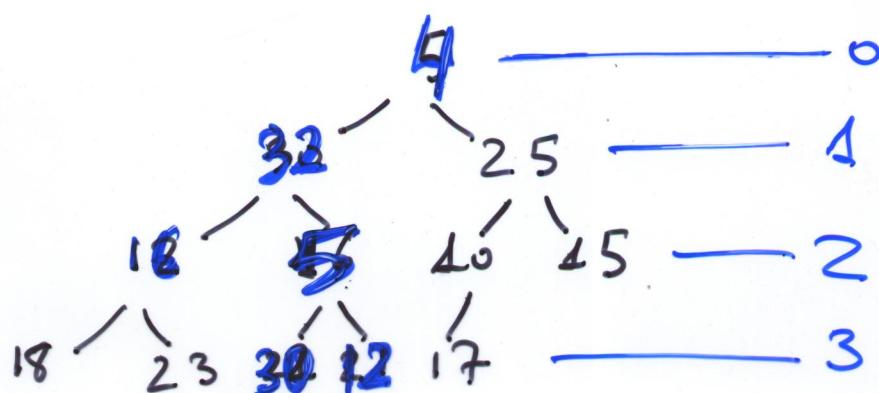
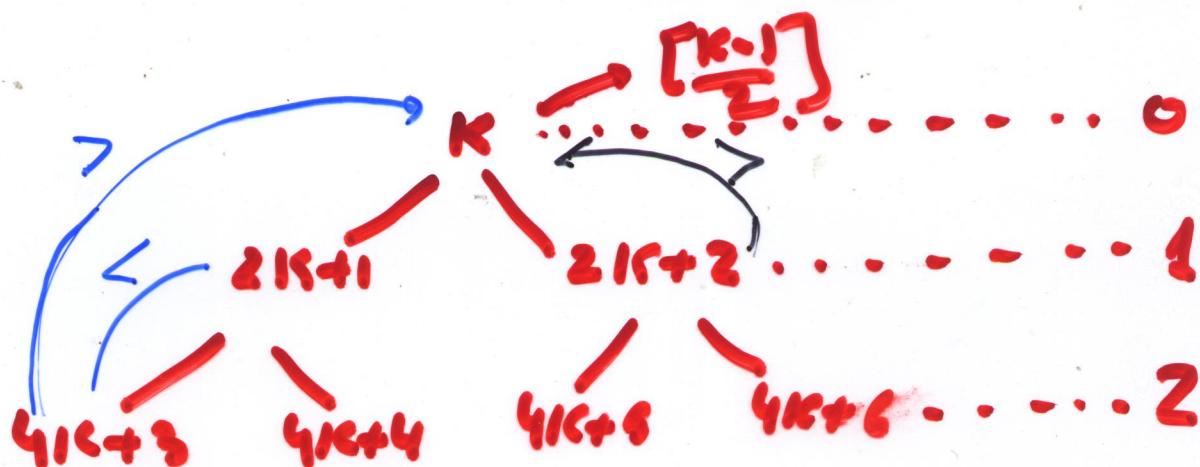
visita todos los  
nodos que no  
tienen hijo a  
la izquierda

visita el primer  
nodo con un  
hijo a la

dcha (si lo hay)

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 si nodo  $\mathbb{E}$  a profundidad par tiene una clave menor que la del padre y mayor que la del abuelo (cuando existan) y si nodo  $\mathbb{E}$  a profundidad impar tiene una clave mayor que la del padre y menor que la del abuelo (cuando existan) con las hojas empujadas a la rrgda

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



In: E D B A H F G C I      }      z h i p o n g d a d i h i a z ?  
Post: E B D H G F I C A

In: E D B A H F G C I

Post: E B D H C F I C A  
↑ ↑  
Tran. hips right Tran. right

```
bool masParesqueImpares (const set<int> &C,  
set<int> &Par,  
set<int> &Impar)
```

```

} s.t <iint>:: iterator p;
for (p = c.begin(); p != c.end(); ++p)
    if (*p % 2 == 0)
        par.insert(*p);
    else
        impar.insert(*p);

```

```
if (par.size() > ciwpar.size())
    return true;
else return false;
```

Usando la clase set de la STL, diseñar una función que determine la intersección de dos conjuntos C1 y C2.

void common (set<int> &C1, set<int> &C2,  
set<int> &resultado)

```
    set<int>::iterator p, q;  
    for (p = C1.begin(); p != C1.end(); ++p)  
        if (C2.count (*p))  
            resultado.insert (*p);
```

Usando la clase set de la STL, diseñar una función que dados 2 conjuntos C1 y C2 determine el conjunto de los elementos de C1 que no están en C2 y todos los de C2 que no están en C1.

set<int> no\_comunes (set<int> C1,  
set<int> C2)

{ set<int>::iterator p, q;

set<int> solucion;

for (p = C1.begin(); p != C1.end(); ++p)

if (!C2.count(\*p))

solucion.insert(\*p);

for (q = C2.begin(); q != C2.end(); ++q)

if (!C1.count(\*q))

solucion.insert(\*q);

return solucion;

Usando la clase set de la STL, construir una función para determinar si un conjunto tiene más de la mitad de sus elementos comunes con otro.

bool masdelamitadcomunes (const set<int>& S,  
const set<int>& T)

4  
set<int>::const\_iterator p, q;  
int n1, n2;  
n1 = 0; n2 = S.size();  
for (p = S.begin(); p != S.end(); ++p)  
if (T.count(\*p))  
n1++;  
if (n1 > n2 / 2) return true;  
else return false;

5

Usando la clase set de la STL, construir una función que divida un conjunto de enteros  $c$  en dos subconjuntos par y impar que contienen respectivamente los elementos pares e impares de  $c$  y que devuelva true si el número de elementos de  $c_{\text{par}}$  es mayor que el de  $c_{\text{impar}}$  y false en caso contrario.

```
bool masparqueimpares (const set<int> & c,  
set<int> & cpar;  
set<int> & cimpar)
```

```
{ set<int> :: iterator p;
```

```
for (p = c.begin(); p != c.end(); ++p)
```

```
if (*p % 2 == 0)
```

```
cpar.insert (*p);
```

```
else
```

```
cimpar.insert (*p);
```

```
if (cpar.size() > cimpar.size())
```

```
return true;
```

```
else return false;
```

```
} // return ((cpar.size() > cimpar.size()))
```

```

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)) > z
}

```

Usando la clase set de la STL, construir una función para determinar si un conjunto tiene todos los elementos pares incluidos dentro de otro.

```
bool inclusionpares (const set<int> &C1,  
                      const set<int> &C2)  
{  
    set<int>::iterator p;  
    for (p = C1.begin(); p != C1.end(); ++p)  
        if (*p % 2 == 0)  
            if (!C2.find(*p)) return false;  
        else ++p;  
    return true;  
}
```

Construir una función para determinar si un  
conjunto está incluido dentro de otro.

```
bool inclusion (const set<int> & (1,  
                           const set<int> & (2))  
{  
    set<int> :: const_iterator p;  
    for (p = (1.begin()); p != (1.end()); ++p)  
        if !(2.count(*p)) return false;  
    return true;  
}
```

bool masparis que imparis (Const set <iut> > r;

set <iut> > & (par);

set <iut> < iimap)

```
{  
    set <iut>:: iterator p;  
    for (p = c.begin(); p != c.end(); ++p)  
        if (*p % 2 == 0)  
            (par.insert (*p));  
        else (imap.insert (*p));  
    if (Cpar.size() > imap.size())  
        return true;  
    else return false;  
}
```

	10	39	6	32	49	18	41	37	
$h_1(k)$	3	6	6	3	4	8	10	2	
$h_0(k)$	4	3	4	3	2	3	4	5	

$$h(k) = (2k + 5) \% 16$$

$$h_i(k) = [h_{i-1}(k) + h_0(k)] \% 4$$

$$h_0(k) = 7 - (k \% 7)$$

$$h(10) = 25 \% 11 = \boxed{3}$$

$$h(39) = 93 \% 11 = \boxed{2}$$

$$h(6) = 12 \% 16 = 6 \rightarrow \text{left}$$

$$h_1(6) = [h(6) + h_0(6)] \% 4 \\ = (6 + 2) \% 4 = \boxed{0}$$

$$h(32) = 69 \% 16 = 3 \rightarrow \text{left}$$

$$h_1(32) = 6 \rightarrow \text{left}$$

$$h_2(32) = \boxed{9}$$

$$h(48) = 4$$

$$h(18) = 8$$

$$h(41) = 10$$

$$h(37) = 2$$

0		
1		
2	37	→ 4
3	10	→ 4
4	49	→ 4
5		
6	39	→ 1
7	6	→ 2
8	18	→ 1
9	32	→ 3
10	41	→ 1

Result =  $\frac{11 \text{ intervals}}{8 \text{ days interval}} = 1.4$

```
void vectorDisperso :: cambiar_valor_defecto (const  
string & nr)  
{  
    r-def = nr;  
    map<int, string>::iterator it = M.begin();  
    while (it != M.end())  
    {  
        if ((*it).second == nr)  
            it = M.erase(it);  
        else ++it;  
    }  
}
```

Disponemos del TDR 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~~:

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

ejecutando el siguiente código

Matriz M;

⋮⋮⋮

Matriz::iterator its

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;
}

Matrix::iterator Matrix::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) //no 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][c]);
            if (*d % 2 == 0) return *this;
        }
        else { //hemos recorrido toda la columna y
            f = 0; //pasamos a la siguiente columna
            c = c + 1;
            d = &(datos[f][c]);
            if (*d % 2 == 0) return *this;
        }
    }
}
```

friend class Matrix;

Para gestionar un documento, se usa un TDA Documento. Este TDA tiene en su representación una tabla Hash en la que cada palabra del documento tiene asociada una lista ordenada con las posiciones en las que aparece la palabra en el mismo.

Implementar una función

`int Documento::min_distancia(string pal1,  
string pal2)`

que devuelva la distancia mínima en la que aparecen las palabras `pal1` y `pal2` en el documento. Para la representación de la tabla Hash se usa hashing abierto

Usamos el TDS map

class Documento {

std::map<string, list<int>> tabla\_hash;

}      ↑  
      unordered\_map // Tabla hash STL

int Documento::min\_distanza (string p1, string p2)

{

list<int> l1 = tabla\_hash[p1];

list<int> l2 = tabla\_hash[p2];

int minima = numeric\_limits<int>::max();

for (list<int>::iterator it1 = l1.begin();  
 it1 != l1.end(); ++it1)

for (list<int>::iterator it2 = l2.begin();

it2 != l2.end(); ++it2)

int d = abs(\*it1 - \*it2);

if (d < minima) minima = d;

}

return minima;

}

## 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  $n-i-1$  ( $i = 0, 1, \dots, n-1$ ).

### Ejemplo 1:

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

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

### Ejemplo 2:

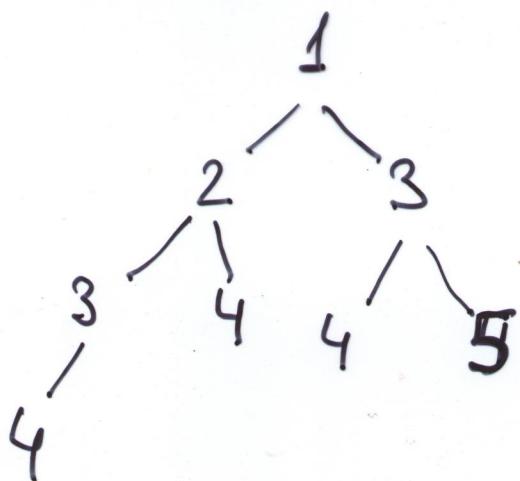
lista inicial: (1, 2, 3, 4, 5)

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

## EJERCICIOS

Dado un árbol binario de enteros, obtener todos los caminos que contengan un valor correcto.

Ejemplo: Los caminos han de llegar hasta una hoja)



$$K=3$$

$\{1, 2, 3, 4\}$   
 $\{1, 3, 4\}$   
 $\{1, 3, 5\}$

$$K=4$$

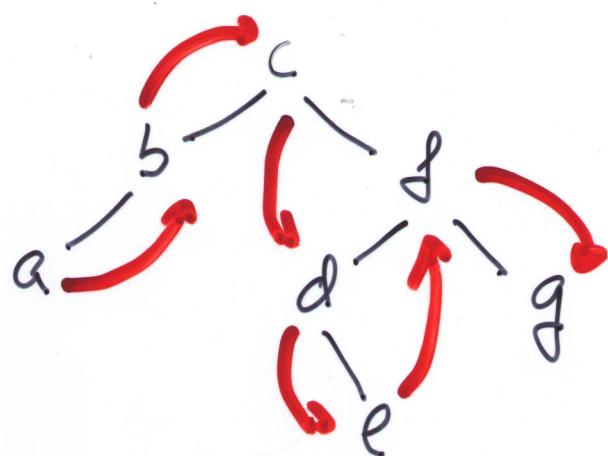
$\{1, 2, 3, 4\}$   
-  $\{1, 2, 4\}$   
 $\{1, 3, 4\}$

$$K=1$$

$\{1, 2, 3, 4\}$   
 $\{1, 2\}$   
 $\{1, 3\}$   
 $\{1, 3, 5\}$

## Ejercicio

Un arbol binario hilvanado es una estructura de datos para la implementacion de un ABRS que facilita ciertos recorridos. Su caracteristica esencia es que <sup>en</sup> el registro que define un nodo se añade un nuevo puntero hilvan que apunta al siguiente nodo en el recorrido en orden del ABRS. Usando el TDS ABRS, diseñar un procedimiento para insertar un nodo en un arbol binario hilvanado



- \* Diseñar una función booleana que devuelva true si en una expresión matemática, los paréntesis, corchetes y llaves están colados de forma correcta. A cada símbolo abierto le corresponde uno cerrado del mismo tipo.
- \* Diseñar una función  
que ne  $\langle \text{int} \rangle$  bajar (que ne  $\langle \text{int} \rangle$  q1, que ne  $\langle \text{int} \rangle$  q2) que devuelva una cola mezcla de q1 y q2 de forma que en la cola de salida los elementos de q2 ocupen las posiciones impares y los elementos de q1 las posiciones pares
- \* Dada una pila p de números reales y un valor x, implementar una función  
 $\text{float multiplica} (\text{stack } \langle \text{float} \rangle p, \text{float } x)$  que multiplique todos los elementos de p por el valor x.  
(no pueden usarse estructuras auxiliares)

Dadas 2 colas con elementos repetidos implementar la función:  
(y ordenados)

que  $\text{queue} < \text{int} \times \text{int} \rangle$  multiintersección (que  $\text{queue} < \text{int} \rangle q_1$ ,  
que  $\text{queue} < \text{int} \rangle q_2$ )

que calcule la multiintersección de 2 colas  
 $q_1$  y  $q_2$  y devuelva el resultado en otra  
cola.

$q_1: [2, 2, 3, 3]$

$q_2: [1, 2, 3, 3, 3, 4]$

multiintersección  $[2, 3, 3]$

Diseñar una función que dada una lista  $L$   
devuelva otra lista  $R$  conteniendo los elementos  
repetidos de  $L$ . Si no hay elementos repetidos  
 $R$  será la lista vacía.

$L = \{5, 2, 7, 2, 5, 5, 1\}$

$R = \{5, 2\}$

Diseñar una función orden : a

int orden (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.

Una lista de frecuencias es un TDS que mantiene un listado alfabético de todas las palabras que aparecen en un conjunto de textos. Por razones de eficiencia, y pensando que se va a utilizar dicho tipo para almacenar frecuencias de palabras de muchos documentos, se ha optado por utilizar varias listas pequeñas ordenadas en lugar de una sola lista con todas las palabras. De esta forma, se tendrá una lista ordenada de las palabras que comienzan por 'a', otra lista ordenada con las palabras que empiezan por 'b' etc. Para cada palabra, se almacena el número de documentos en que aparece y con qué frecuencia. Indicar cuál sería la mejor representación para el TDS lista de frecuencias.

① Pasar la expresión Postfijo:  
 $abc + /ef*gh*- -$

a prefijo

② Pasar de  $\Delta G$  a  $\Delta B$  → hijo más a  
la agda → hijo-izqda  
hermano  
derecha → hijo-dcha

? Relación entre recorridos?

③ TDS bicola: estructura lineal en la que la inserción se hace por los extremos pero el borrado y  
acceso a los elementos solo por uno: el frente.

Eficacia de las operaciones: insertar\_frente,  
insertar\_final, borrar y frente si la bicola se  
representa mediante: vector dinámico, lista  
dblemente enlazada y matriz circular

	Vector Dinámico	Lista Doble	Matriz Circular
nodo frente			
inserción final			
inserción frente			
borrado frente			