

soluciones1.pdf



Anónimo



Estructuras de Datos



2º Grado en Ingeniería Informática



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



MARVEL STUDIOS

Bruja Escarlata y Visión

© 2020 MARVEL
12

Disney+

Serie Original ya disponible en exclusiva

ORDENAR

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

Solución

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

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

Solución

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

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

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

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

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

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

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

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

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

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

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

Solución

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

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

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

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

$$2 \longrightarrow 1$$

$$1^n \longrightarrow 1$$

Por tanto:

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

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

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

$n^{1.5}$, \sqrt{n} , $n \log_2 \log_2 n$, n^2 , $2^{n/2}$, $n^2 \log_2 n$, $n2^n$

\sqrt{n} , $n \log_2 \log_2 n$, $n^{1.5}$, n^2 , $n^2 \log_2 n$, $2^{n/2}$, $n2^n$

n , $n^{1.5}$, $n \log_2 n$, \sqrt{n} , $n \log_2 \log_2(n)$, n^2 , $2^{n/2}$,
 37 , $n^2 \log_2 n$, $n2^n$, $(1.0000001)^n$

37 , \sqrt{n} , n , $n \log_2 \log_2 n$, $n \log_2 n$, $n^{1.5}$, n^2 ,
 $n^2 \log_2 n$, $(1.0000001)^n$, $2^{n/2}$, $n2^n$

MARVEL STUDIOS
Bruja Escarlata
y Visión



Disney+

Serie Original ya disponible
en exclusiva

void eliminar (vector L, int x)

{
int aux, p; $x = \text{null}$
for (p = primero(L); p != \text{fin}(L);)

{
aux = elemento (p, L);
if (aux == x)
borrar (p, L)
else p++

}

for (K=1; K <= n; K*=2)

{
for (j=1; j <= K; j++)

sum2++

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

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

for (K=1; K <= n & K != 2) {
for (j=1; j <= 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;

 r++;

 i++;

$O(n \log_2 n)$

} while (i <= n)

—————

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

do {

 j = 1;

 while (j <= r)

 j = j * 2;

 r++;

 i++;

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

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

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

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 (compuestos por código, título, año ...). La **búsqueda** se hace por el título de cada película.

Para la **inscripción y borrado** de películas, se usa un código único asociado a cada película. Además se necesita una función para **imprimir** 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) **AB3**
- 4) **DVL**
- 5) **Tabla hash (abierto)**

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

MARVEL STUDIOS

Bruja Escarlata
y Visión

Serie Original ya disponible en exclusiva

Disney+

	Vector ordenado	Lista ordenada	DBB	AVL	Tabla hash
buscar	$O(\log_2 n)$	$O(n) / O(\log_2 n)$	$O(n)$	$O(\log_2 n)$	$O(n) / O(1)$
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	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$[O(n \log_2 n)]$

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

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

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

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

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

$$1^n \longrightarrow 1$$

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=1; k <= n; k*=2)

for (j=1; j <= k; j++)

sum2++;

else

for (k=1; k <= n; k*=2)

for (j=1; j <= n; j++)

sum2++;

return sum2;

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

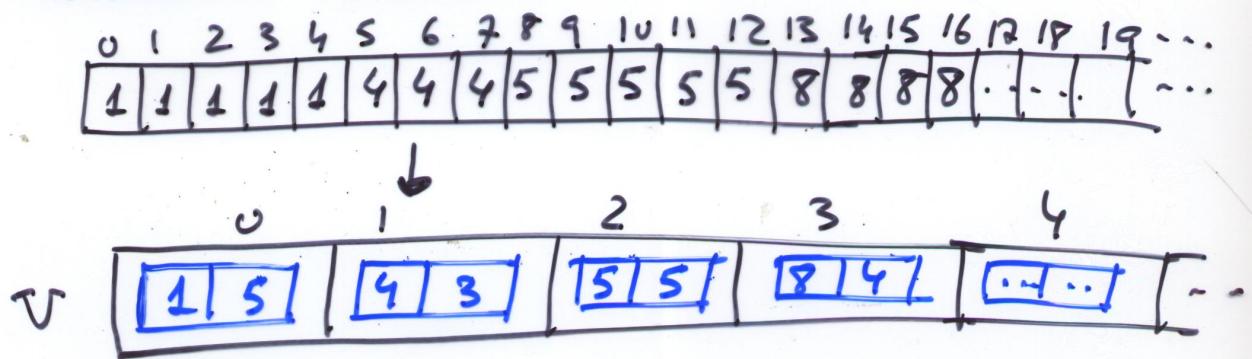
$$\sum_{k=1}^{\log_2 n} \sum_{j=1}^n 1 = \sum_{k=1}^{\log_2 n} n = n \sum_{k=1}^{\log_2 n} k = n \log_2 n [O(n \log_2 n)]$$

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

$$\begin{array}{cccccccccc} \underline{n=64} & \overbrace{1 \ 2 \ 4 \ 8 \ 16 \ 32}^K \ 64 & \rightarrow 2n-1 & \{2n-1 \\ j \rightarrow 1+2+4+8+16+32 \rightarrow n-1 & & & \{63 \end{array}$$

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. 5, 1, 1, 1, 1, 1, 4, 4, 4, 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. Hay distintas representaciones capaz de almacenar los datos del vector con un espacio proporcional al número de elementos existentes.



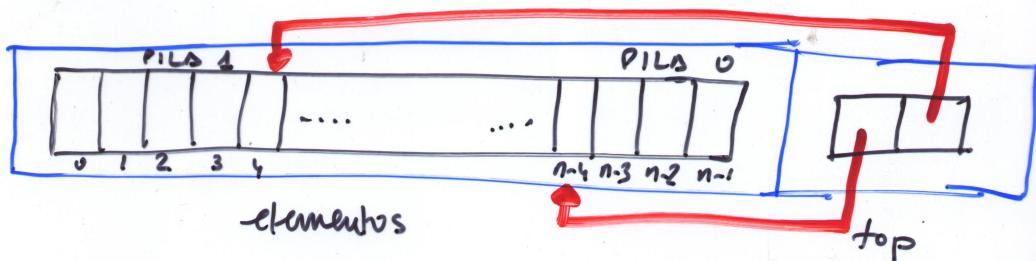
```

int valor_pos (int i)
{
    int 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)

}

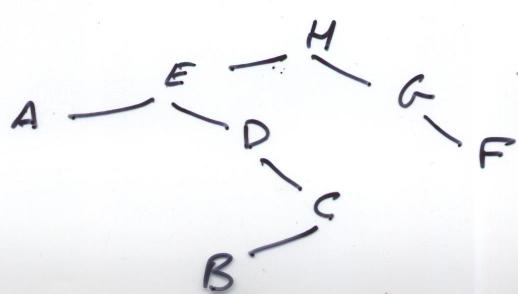
top [indicepila] = top [indicepila]

- (indicepila ? 1 : -1);

5

Post: A B C D E F 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);
}
```

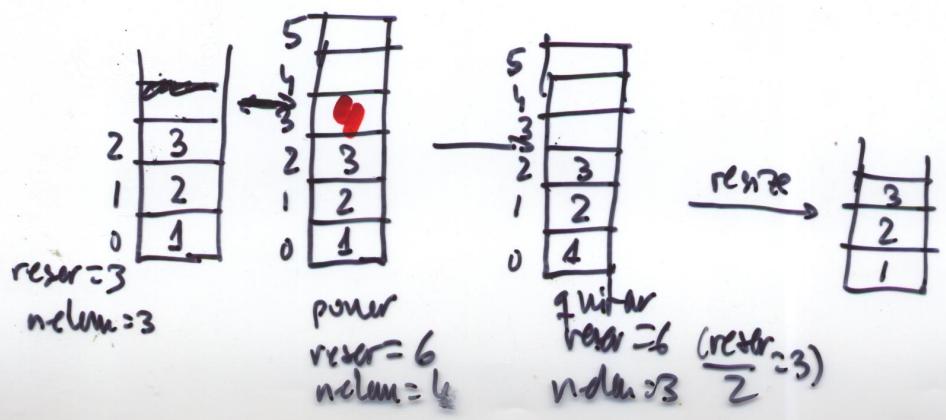
- Comportamiento de operaciones consecutivas poner/quitar cuando el vector está completo:

* Si ponemos 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;

}

}

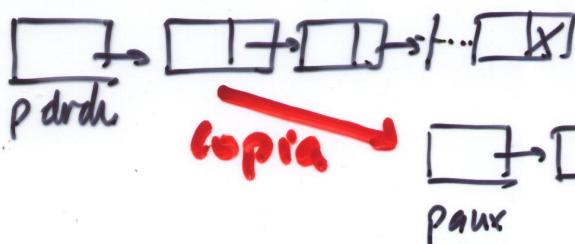
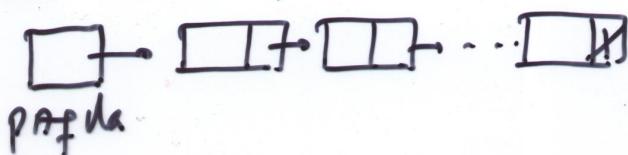
Pila & Pila:: operator = (const Pila & p)

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

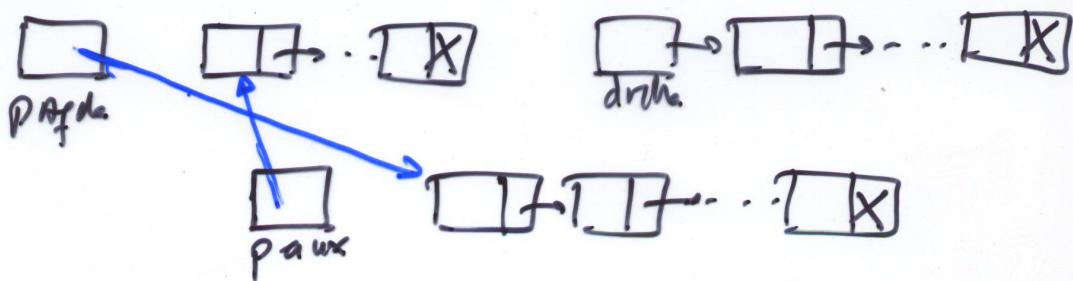
↳

hacemos pizqda = pdrecha (asignacion de 2 pilas)

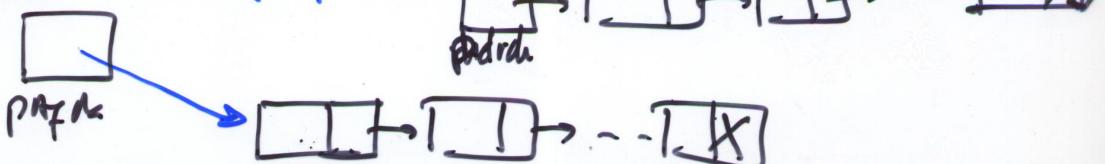
Paso 1 (crea paux como copia de la pila drcha)



Paso 2 (Intercambia los contenidos de paux y pdrecha)



Paso 3 (Destruye paux)



(SI destruirse paux se destruyen los celdas de la pila pizqda y pizqda queda con un conjunt de celdas idéntico a pdrecha)

MARVEL STUDIOS
Bruja Escarlata
y Visión



Disney+

Serie Original ya disponible
en exclusiva

Usando el TDS lista, construir una función que tenga como entrada dos listas y devuelva 1, si la primera está contenida en la segunda (tiene los mismos elementos, consecutivos y en el mismo orden) y 0 en otro caso. P. ej:

$$\left. \begin{array}{l} l1 = \langle 1, 2, 3 \rangle \\ l2 = \langle 0, 1, 2, 3 \rangle \end{array} \right\} \textcircled{1}$$

$$\left. \begin{array}{l} l1 = \langle 1, 2, 3 \rangle \\ l2 = \langle 1, 2, 2, 3 \rangle \end{array} \right\} \textcircled{0}$$

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

```
    list<int> iterator :: p, q;
```

```
    int x;
```

```
    x = l1.front();
```

```
    p = l1.begin();
```

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

```
    while (p != l1.end())
```

```
        if (*p != *q)
```

```
            return false;
```

```
        else { ++p;
```

```
            ++q;
```

```
    }
```

```
    return true;
```

```
5
```

void invertir (cola d())

{ while (! c.empty())

{ int aux = c.front();

c.pop();

invertir();

c.push(aux);

}

—————

bool compilar (pila d P, cola d())

{ invertir();

while (! c.empty())

{ if (c.front() != P.front())

return false;

else { c.pop();

P.pop();

}

return true;

}

Usando el TDA `list<int>`, construir una función
`void agrupar_elemento (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 ocurrencia. P. ej.

si `entrada = {1, 3, 4, 1, 4}` y `k=1` entonces
`entrada = {1, 1, 3, 4, 4}`

o si `entrada = {3, 1, 4, 1, 4, 1, 1}` y `k=1`
`entrada = {3, 1, 1, 1, 1, 4, 4}`

`void agrupar_elemento (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;`

5

Usando el TDS `list<T>`, construir una función template <class T> `void dividir_lista (list<T> lista, T k)`

que agrupe en la primera parte de la lista los elementos menores que k y en la segunda los mayores o iguales. Han de usarse iteradores, no te permitan estructura auxiliares, no te puedes modificar el tamaño de la lista y la función (ia de $O(n)$)

Ejemplo,

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

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

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

```
    if (*it > k)
        lista.push_back (*it),
        it = lista.erase (it)
    else
        ++it;
```



Disney+

Serie Original ya disponible
en exclusiva

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 en p
 - 2.1.2. borrar elemento.

void agrupar_elemento (list<int> &entrada,
int k)

list<int> iterator::p, q;

$p = \text{entrada}.\text{find}(\text{entrada}.\text{begin}(),$
 $\text{entrada}.\text{end}(), k);$

$q = ++p;$

while ($q != \text{entrada}.\text{end}()$)

{ if ($*q == k$)

{ entrada.insert (p, k);

$q = \text{entrada}.\text{erase}(q);$

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, 1, 4, 5, 5, 3, 2, 7}
Salida = {{1, 1, 1, 1}, {2}, {3, 3}, {4, 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 <int>:: Nodo *MS (const ArbolBinario <int> &ar,

const ArbolBinario <int>:: Nodo *v,

const ArbolBinario <int>:: Nodo *w)

{ ArbolBinario <int>:: Nodo *p;

for (p = a. padre (v); p != nodo_nulo ; p = a. padre (p))

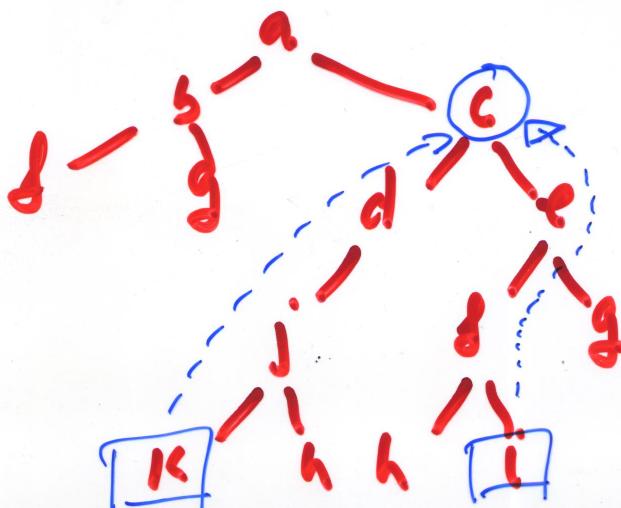
while (w != nodo_nulo)

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. ℓ se entiende como (el) extremo que un nodo es descendiente de si mismo). Dibujar una función que toga como entrada un árbol binario de arriba y 2 nodos r y w y como salida el $AMC(r, w)$.

Nodo AMC (ArbolBinario ℓ , nodo r , 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 \neq $w \rightarrow \text{padre}() \neq 0$) ↳

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

if ($it \neq rnodos.end()$)

 find = true;

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

↳ if (find) return *it;

else return nodo();

↳

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

int nivel ($ABR \leftarrow A$)

↳ Nodo u;

$n = A.raiz();$

contador = 0;

while ($n \neq \text{nodo nulo}$)

↳ $n = A.dcha(n);$

contador++;

↳

return contador;

;

```
bool sesgado (ArbolBinario<T> *A, nodo n)
{
    if (n != 0)
    {
        if (n.izqdo().altura() >
            n.dcha().altura())
            return false;
        else
            return sesgado (A, n.izqdo()) &&
            sesgado (A, n.dcha());
        else
            return true;
    }
}
```



node amc (ArbolBinario < T > padre, nodo v,
nodo w)

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

while (v → padre () != 0)

{
vNodos. insert (v → padre);
v = 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();

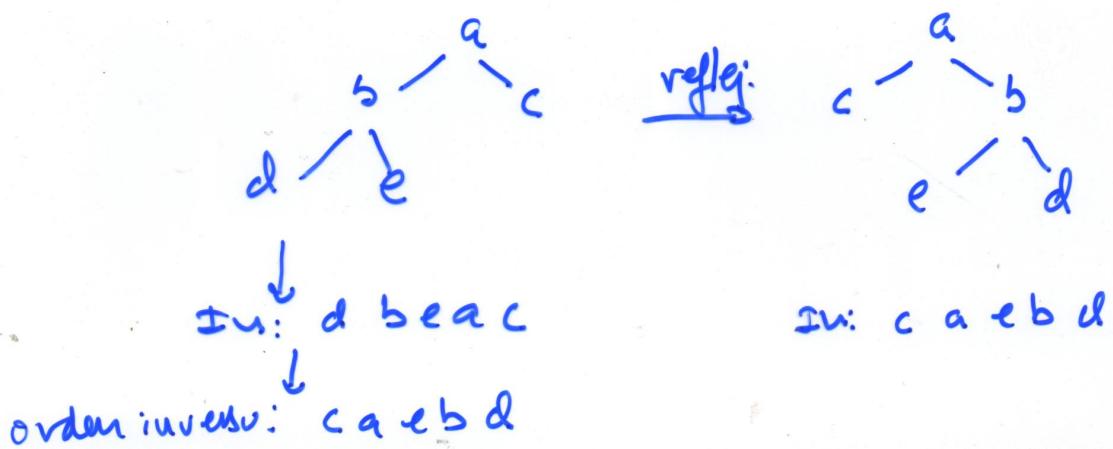
{ if (find)

return ⁢

, 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 $f(r) = g(r)$?



Evaluando la expresión ^{en} postfijo:

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

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

Solución: 0

Usando el TDA 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.dcha(T.raiz(), T)),
    else return 1 + contar (a, b,
                            T.raiz(T.raiz(), T))
                            + contar (a, b,
                            T.dcha(T.raiz(), T)),
}
```

g

```

void ordenarNoRecursivo (ArbolBinario<T> A)
{
    stack<Nodo> piladenodos;
    Nodo p = A.root();
    while (p != nodonulo)
    {
        while (p != nodonulo)
        {
            if (A.hijoDcha(p))
                piladenodos.push (A.hijoDcha(p));
            piladenodos.push (p);
            p = A.hijoIzqda(p);
        }
        // segundo while
        p = piladenodos.pop();
        while (!piladenodos.empty())
        {
            cout << A.etiqueta(p);
            p = piladenodos.pop();
        }
        // hacer white
        cout << A.etiqueta(p);
        if (!piladenodos.empty())
        {
            p = piladenodos.pop();
            else p = nodonulo;
        }
        // primer white (p != nodonulo)
        // procedimiento
    }
}

```

Met en la pila el hijo dcha
 y el propio nodo

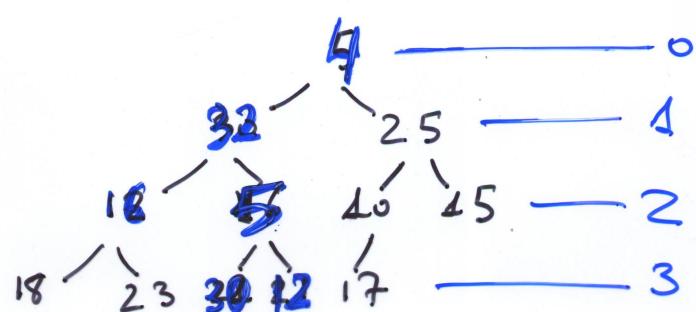
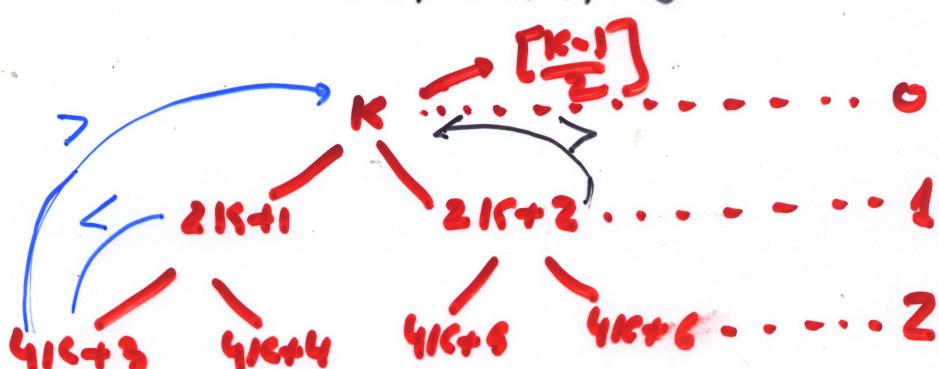
Pop de un nodo sin hijo izqda
 visita todos los nodos que no tienen hijo a la dcha

visita el primer nodo con un hijo a la dcha (si lo hay) y // procedimiento



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 π a profundidad par tiene una clave menor que la del padre y mayor que la del abuelo (cuando existen) y si 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 rrgda

Algoritmo para insertar una clave. Optar por crear un heap-doble con 430, 25, 12, 16, 10, 15, 5, 18, 23, 32, 4, 175



In: E D B A H F G C I } ¿ hijo nro de la raiz?
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 G F I C A
↑ Traz hijo nro
raiz

bool masParesQueImpares (const set<int> & c,
set<int> & cpar,
set<int> & ciimpar)

```
{  
    set<int>::iterator p;  
    for (p = c.begin(); p != c.end(); ++p)  
        if (*p % 2 == 0)  
            cpar.insert(*p);  
        else  
            ciimpar.insert(*p);
```

```
    if (cpar.size() > ciimpar.size())  
        return true;  
    else  
        return false;
```

}

Usando la clase set de la STL, diseñar una función que determine la intersección de dos conjuntos C_1 y C_2 .

```
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 C_1 y C_2 determine el conjunto de los elementos de C_1 que no están en C_2 y todos los de C_2 que no están en C_1 .

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;

5

MARVEL STUDIOS
Bruja Escarlata
y Visión



Disney+

Serie Original ya disponible
en exclusiva

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 masde la mitad comunes (const set<int> & (1),  
const set<int> & (2))
```

```
4 set<int>:: const_iterator p, q;  
int n1, n2;  
n1 = 0; n2 = (2. size());  
for (p = (2. begin()); p != (2. end()); ++p)  
if ((2. count (*p))  
n2++;  
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 cpar es mayor que el de cimpar y false en caso contrario.

bool masparseguimpares (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 = c2.begin(); p != c2.end(); ++p)
        if (!c1.count(*p)) solution.insert(*p);
    for (q = c1.begin(); q != c1.end(); ++q)
        if (!c2.count(*q)) solution.insert(*q);
    return solution;
}

```

```

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

bool es_raiz_3nodo (const ArbolBinario<int> & A, int z)
{
    return abs (contar (A.izqda (A.raiz()), A) -
                contar (A.dcha (A.raiz()), A)) > 2
}

```

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> & (1),
const set<int> & (2))

```
{  
    set<int>::iterator p;  
    for (p = (1).begin(); p != (1).end(); ++p)  
        if (*p % 2 == 0)  
            if !(2).find(*p) return false;  
        else ++p;  
    return true;  
}
```

MARVEL STUDIOS
Bruja Escarlata
y Visión



Disney+

Serie Original ya disponible
en exclusiva

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

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

bool masparesqueimpares (const set<int> & s,

set<int> & p);

set<int> & impar)

{ 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;

}

	10	39	6	32	49	18	41	37	
$h_{11}(k)$	3	6	6	3	4	8	10	2	
$h_0(k)$	4	3	1	3	7	3	4	5	

$$h(k) = (2k + 5) \% 11$$

$$h_{11}(k) = [h_{c_1}(k) + h_0(k)] \% 11$$

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

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

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

$$h(6) = 13 \% 11 = 6 \rightarrow \text{ultimo}$$

$$h_1(6) = [h(6) + h_0(6)] \% 11$$

$$= (6 + 2) \% 11 = \boxed{8}$$

$$h(32) = 69 \% 11 = 3 \rightarrow \text{ultimo}$$

$$h_1(32) = 1 \rightarrow \text{ultimo}$$

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

$$h(48) = 4$$

$$h(18) = 8$$

$$h(41) = 10$$

$$h(37) = 2$$



$$\text{Rend} = \frac{11 \text{ instantes}}{8 \text{ deng cuy clave}} = 1.4$$

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

MARVEL STUDIOS
Bruja Escarlata
y Visión



Disney+

Serie Original ya disponible
en exclusiva

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~~:

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

ejecutando el siguiente código

Matriz M;

⋮⋮⋮

Matriz::iterator it;

```
for (it = M.begin(); it != M.end(); ++it)  
    cout << *it;
```

se imprimirá sobre la salida estándar:

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 () {
    Matrix::iterator it;
    it.d = & datos[nf-1][nc];
    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 - &*latents[0][0]) % n_c; //columna donde  
//aparece
```

while (true) {

if ($f >= n - 1$ $\&$ $c >= n - 1$) // no hay mas
// elementos

$\delta = \ell(\text{datos}[\text{inf} : \text{c}])$;

return *this; //devolveemos el

۶

else

if ($f < \inf - \epsilon$) {

$f = f + 1$; // en la misma columna, el siguiente

d = & (datos [f] [c]);

if (*d % 2 == 0) return *this;

4

else } // hemos recorrido toda la columna y

$f = 0$; "pasamos a la siguiente columna"

$$C = C + 1;$$

$d = \varrho(\text{data}[\text{f}][\text{c}]),$

if (*d % 2 == 0) return *this;

5

٤

4

friend class Maths;

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::miu_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 TDA map

class Documento {

std::map<string, list<int>> tabla_hash;

}; **↑** **unordered_map //Tabla hash STL**

int Documento::min_distanza (string par1, string par2)

{

list<int> l1 = tabla_hash [par1];

list<int> l2 = tabla_hash [par2];

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 esté 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:

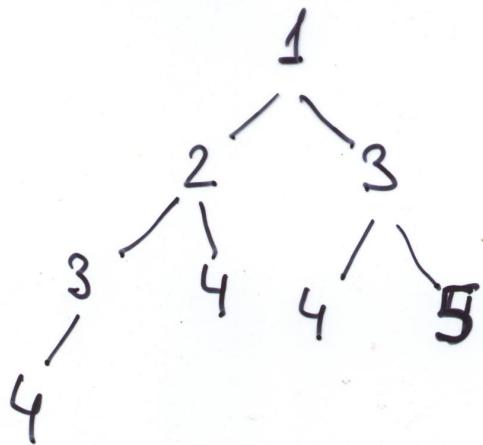
lista inicial: (1, 2, 3, 4, 5)

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

EJERCICIOS

Dado un árbol binario de enteros, obtener todos los caminos que contengan un valor concreto k

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}

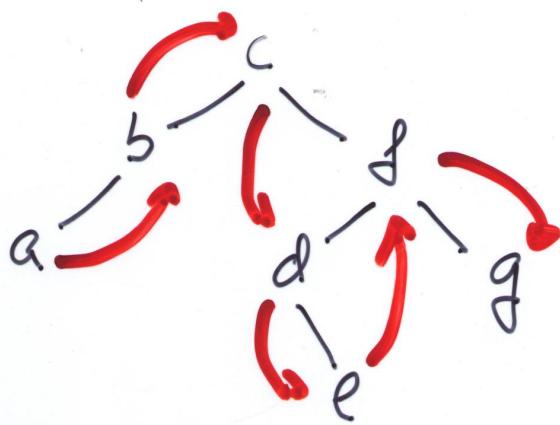
{1, 2, 4}

{1, 3, 4}

{1, 3, 5}

Ejercicio

Un árbol binario hilvanado es una estructura de datos para la implementación de un ABBS que facilita ciertos recorridos. Su característica 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 árbol binario hilvanado



MARVEL STUDIOS
Bruja Escarlata
y Visión



Disney+

Serie Original ya disponible
en exclusiva

- * Diseñar una función booleana que devuelva true si en una expresión matemática, los paréntesis, corchetes y llaves estan colados de forma correcta (a cada símbolo abierto le corresponde uno cerrado del mismo tipo).
- * Diseñar una función
que reciba $l_{int} > 0$ (que sea $l_{int} > q_1$, que sea $l_{int} > q_2$)
que devuelva una cola mezcla de q_1 y q_2
de forma que en la cola de salida los elementos
de q_2 ocupen las posiciones impares y los
elementos de q_1 las posiciones pares
- * Dada una pila p de números reales y un
valor x , implementar una función
 $float multiplica (stack <float> p, float x)$
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 $queue<int>$ multiintersección ($queue<int> q1$, $queue<int> q2$)

que calcule la multiintersección de 2 colas $q1$ y $q2$ y devuelva el resultado en otra cola.

$q1: [2, 2, 3, 3]$

$q2: [4, 2, 3, 3, 3, 4]$

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.

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

$R = \{5, 2\}$

Disetiar 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 3 si no está ordenada de ninguna forma.

Una lista de frecuencias es un ~~TDS~~ que ~~contiene~~ ^{que} un listado alfabetico 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 ΔG a ΔB \rightarrow hijo más a la agda \rightarrow hijo-rgda

hermano
derecha \rightarrow hijo-drcha

? Relación entre reconocidos?

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

Eficiencia de las operaciones: insertar frente, insertar_final, borrar y frente si la bicola se representa mediante: vector dinámico, lista doblemente enlazada y matriz circular

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