Pregunta **1**

No s'ha respost

Puntuat sobre 1,00

Tiempo estimado: 5 minutos

Penalización: no

Queremos implementar una clase dequeue (doble cola) en la que se puedan hacer inserciones, borrados y consultas en ambos extremos de la secuencia. Es decir, tendremos métodos para insertar por el principio y por el final, y para eliminar el principio y el final.

Elige las opciones adecuadas para que las implementaciones de los cuatro métodos mencionados sean eficientes (coste constante), sin gastar mas memoria de la necesaria.

```
class dequeue {
private:
        struct nodo {
                Elem info;
```

Tria...

```
                nodo* seg;
        };
        nodo* primer; // apuntador al primero
        nodo* ult; // apuntador al ultimo elemento
        int sz; // numero de elementos
```

Tria...

```
public:
        ...
}
```

Pregunta **2**

No s'ha respost

Puntuat sobre 1,00

Tiempo estimado: 5 minutos

Penalización: no

Considera las siguientes funciones añadidas a la clase Arbre (cuya especificación encontrarás aquí).

```
template <typename T> class Arbre {
private:
 struct node_arbre {
 T info;
 node_arbre* segE;
 node_arbre* segD;
 };
 node_arbre* primer_node;
 ...
```

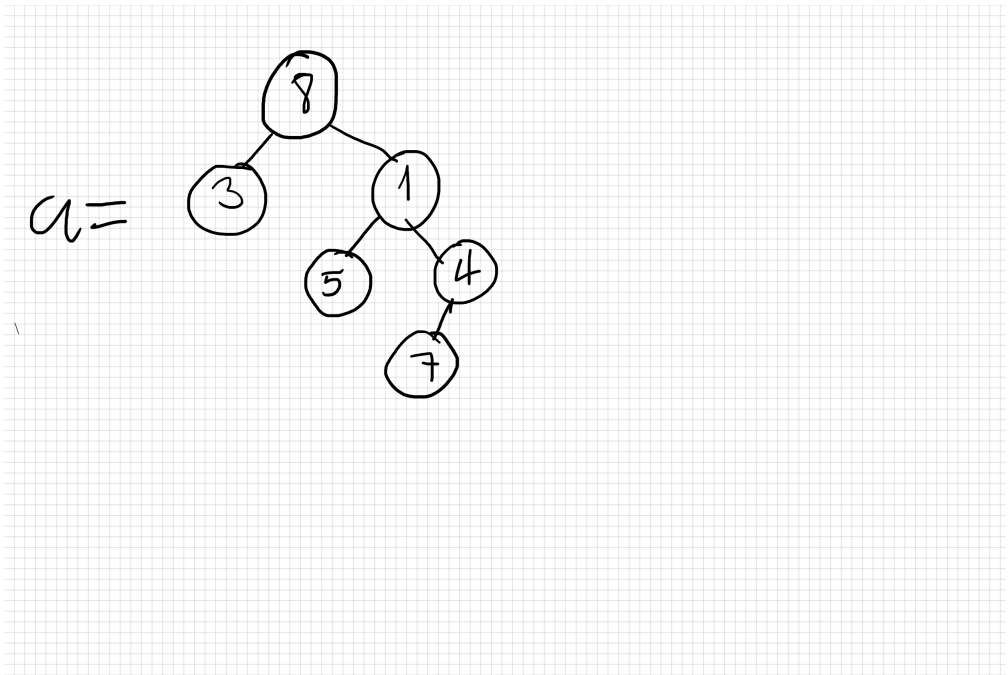
```
// Pre: la jerarquia de nodes a la qual apunta p conté com a molt
// un cop el valor x
// Post: ...
 static node_arbre* i_transforma(node_arbre* p, const T& x) {
```

```
      if (p != nullptr) {
         if (p -> info == x) return zumba(p);
         else {
             p -> segE = i_transforma(p -> segE, x);
             p -> segD = i_transforma(p -> segD, x);
             return p;
          }
       else {
         return nullptr;
       }
}
```

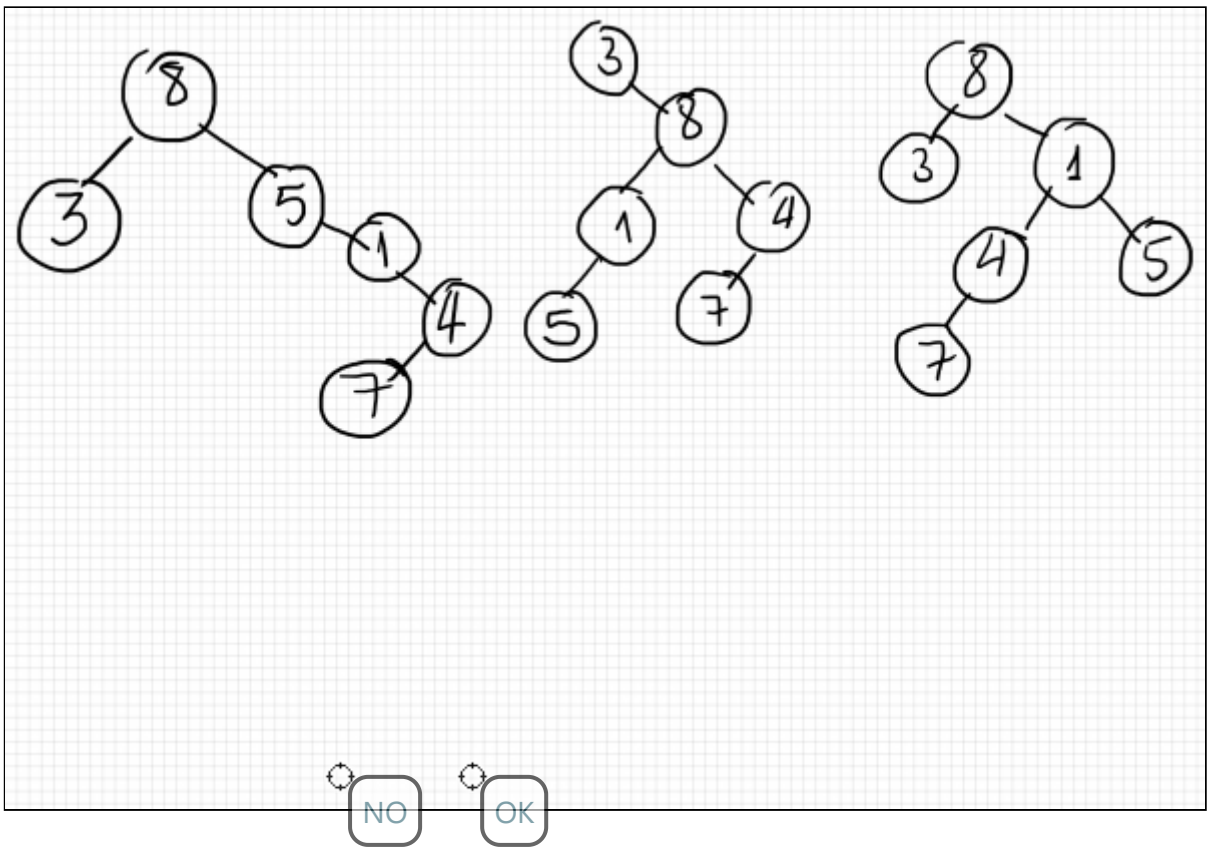
```
// Pre: p != nullptr
// Post: ...
static node_arbre* zumba(node_arbre* p) {
node_arbre* q = p -> segE;
if (q == nullptr) return p;
p -> segE = q -> segD;
q -> segD = p;
return q;
}
```

```
public:
 ...
// Pre: l'Arbre implícit no és buit i conté exactament una x
// Post: ...
 void transforma(const T& x) {
        primer_node = i_transforma(primer_node, x);
 }
};
```

Si a es el árbol en esta figura



marca con OK cúal de los siguientes tres árboles es el resultado de hacer a.transforma(1); y con NO los otros dos. Arrastra los marcadores OK y NO para que caigan en las raíces de los respectivos árboles.


NO
OK

Pregunta **3**

No s'ha respost

Puntuat sobre 1,00

Tiempo estimado: 10 minutos

Penalizaciones: no hay

Tenemos una clase CjtListas en la que cada objeto consiste en N listas secuenciales, siendo N un parámetro dado al crear el CjtListas.

Una de las operaciones es *localiza*, que dado un índice i, 1 <= i <= N, y un string s, devuelve cierto si y solo si el string s aparece en la lista i-esima del CjtListas. Pero además, en caso de que s esté en la lista entonces s debe reubicarse como primer elemento de la lista. Cada una de las listas está implementada como una lista simplemente encadenada. Completad el método CjtListas::localiza haciendo las elecciones apropiadas.

```
class CjtListas {
private:
  struct nodo {
      string info;
      nodo* seg;
  };
  int N;
  vector<nodo*> sent; // sent[i] apunta al centinela de
 // la lista (i+1)-esima, 0<= i < N
public:
 ...
 // Ctor. Crea un CjtListas con N listas vacías
 CjtListas(int N);
 // Pre: 1 <= i <= N
   // no es const para poder reordenar la lista
 // Post: devuelve cierto si y solo si la lista i-ésima
 // contiene el string s; si s está en la lista, dicha
 // lista se reorganiza en caso necesario para que s sea
 // el primer elemento de la lista
 bool localiza(int i, string s);
}
```

```
bool CjtListas::localiza(int i, string s) {
 nodo* p = Tria... ;
 while ( Tria... ) {
 p = p -> seg;
 }
 if ( Tria... ) {
 return p -> seg != nullptr;
 } else {
 nodo* q = p -> seg -> seg;
  Tria...
 p -> seg = q;
 return true;
 }
}
```

Pregunta **4**

No s'ha respost

Puntuat sobre 0,50

Tiempo estimado: 5 minutos

Penalización: -0.5 las respuesta incorrectas, NS/NC o no escoger opción no penaliza

Considera las siguientes funciones añadidas a la definición estándar de Arbre (puedes consultarla en este enlace):

```
template <typename T> class Arbre {
private:
   struct node_arbre {
```

```
T info;
```

```
node_arbre *segE, *segD;
```

```
};
```

```
node_arbre* primer_node;
```

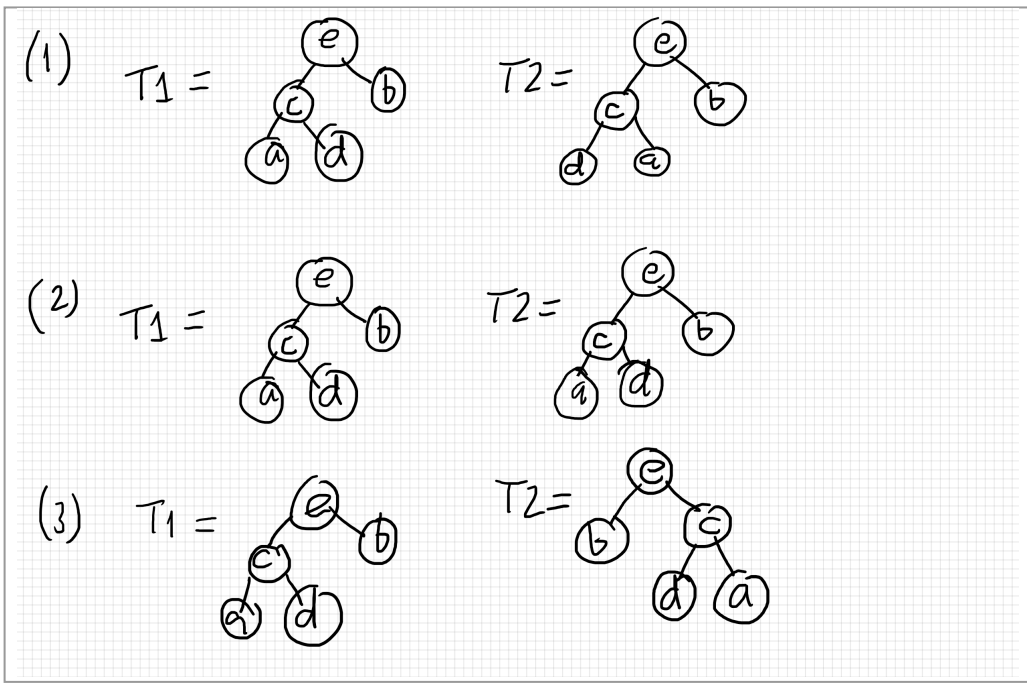
```
// Pre: cierto
// Post: ....
static bool i_misterio(node_arbre* p, node_arbre* q) {
    if (p == nullptr or q == nullptr)
return p == q;
    else if (p -> info != q -> info)
return false;
    else return i_misterio(p -> segE, q -> segE)
and i_misterio(p -> segD, q -> segD);
}
```

```
public:
        ...
```

```
// Pre: cierto
// Post: ....
   static bool misterio(const Arbre& a, const Arbre& b) {
     return i_misterio(a.primer_node, b.primer_node);
   }
};
```

```
```

¿Qué devuelve la función misterio(T1, T2) en los casos (1) a (3)?



Trieu-ne una:

- (1) true, (2) true, (3) false
- NS/NC
- (1) true, (2) false, (3) false
- (1) false, (2) true, (3) false
- (1) false, (2) true, (3) true
- (1) true, (2) false, (3) true
- (1) false, (2) false, (3) true

Pregunta **5**

No s'ha respost

Puntuat sobre 2,00

Tiempo estimado: 15 minutos

En este ejercicio tendrás que implementar un nuevo método para una clase Pila

```
template <typename T> class Pila {
     struct nodo {
           T info;
          nodo* sig;
     };
     nodo* cima;
public:  ...
```

// Pre: cierto

// Post: si x está en la pila implícita todo elemento apilado
// después de la x más cercana a la cima se elimina,
// y la x más cercana a la cima pasa a ser la cima; si x
// no está en la pila implícita no se hace nada
void pop_and_stop(const T& x);
...
}

El apuntador cima de un objeto P de la clase Pila apunta al último elemento apilado o bien a nullptr si P es una pila vacía. Las operaciones de apilar, desapilar, cima, etc. son como las de pilas ordinarias implementadas con una secuencia enlazada de nodos. El último nodo de la cadena de nodos que representa a pila, es el primer elemento que se apiló y no tiene sucesor (su atributo sig es nullptr).

El método pop_and_stop(const T& x) se aplica sobre una Pila cualquiera (puede ser vacía). Si x no está en la pila entonces el método la deja intacta. Pero si x está en la pila entonces desapilará todos los elementos que están "por encima" de la x más cercana a la cima.

Por ejemplo si la pila es P = [3, 2, 0, 4, 2, 8, -1, 4, 7, 2, -1], siendo 3 la cima de la pila, entonces después de hacer P.pop_and_stop(4); la pila es P = [4, 2, 8, -1, 4, 7, 2, -1].

Se valorará la eficiencia de tu solución. El método debería "visitar" exclusivamente los elementos que desapilará (o la pila entera si x no está), no debería recorrer ningún otro elemento. Si la pila tiene N elementos y la x más cercana a la cima tiene r elementos "por encima" entonces el coste ha de ser proporcional a r; el coste será proporcional a N si x no está en la pila.

Puedes definir nuevas operaciones privadas (static o no) si lo quieres, pero en tal caso deberás especificarlas con detalle además de implementarlas.

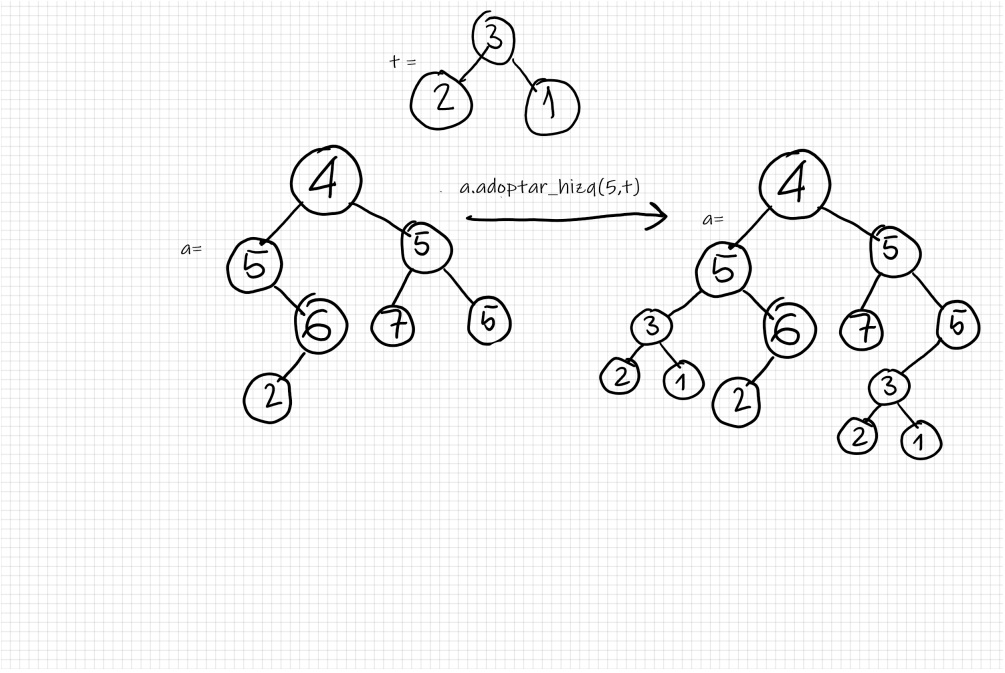
Pregunta **6**

No s'ha respost

Puntuat sobre 2,00

Tiempo estimado: 20 minutos

Queremos ampliar la clase Arbre<T> de árboles binarios con un nuevo método adopta_hizq que, dados t, que es un Arbre<T>, y un elemento x de tipo T, modifica el objeto sobre el que aplicamos el método de tal manera que todo nodo que contega x y no tenga hijo izquierdo pasa a tener a t como hijo izquierdo. En este enlace puedes acceder a la definición de la clase Arbre vista en clase. Puedes utilizar cualquiera de los métodos públicos o privados definidos en la clase Arbre<T> con el fin de implementar adopta_hizq, pero en tu solución debes aprovechar que estás implementando métodos con **acceso a la representación**.



Utiliza la plantilla dada para tu solución. Te será útil definir una función de inmersión privada que recibe un apuntador a la raíz del (sub)árbol donde se quiere aplicar la transformación.

```
template <typename T> class Arbre {
private:
    struct node_arbre {
        T info;
        node_arbre* segE;
        node_arbre* segD;
    };
    node_arbre* primer_node;
    static node_arbre* copia_node_arbre(node_arbre* m);
    static void esborra_node_arbre(node_arbre* m);
// no afegiu cap atribut nou ni modifiqueu
// els de la classe Arbre
// completa la capçalera i implementació de la funció
// d'immersió privada escriu el tipus retornat, si és static
// o no i si és const o no
// Pre: escribe la precondición
// Post: escribe la postcondición
..... i_adopta_hizq(node_arbre* p, const T& x, const Arbre<T>& t) ... {
```

```
      // tu implementación aquí
}
public:
```

```
   ...
```

```
// Pre: cierto
// Post: en el Arbre implícito todo nodo que contiene a x y no tiene
// hijo izquierdo pasa a tener un hijo izquierdo idéntico a t; ningún
// otro nodo del Arbre implícito cambia
void adopta_hizq(const T& x, const Arbre<T>& t) {
      // tu implementación aquí
```

```
 }
}
```

Pregunta **7**

No s'ha respost

Puntuat sobre 1,00

Tiempo estimado: 10 minutos

Tenemos una clase BagOfWords en la que almacenamos un conjunto de strings. BagOfWords está implementada como una secuencia encadenada de nodos en la que guardamos un string en cada nodo. Además, esa secuencia está ordenada en orden alfabético creciente. Cada nodo (excepto el último) apunta a su sucesor en la secuencia. La cadena se inicia con un nodo centinela que apunta al nodo con el primer string (el menor en orden alfabético) del conjunto. La cadena se cierra circularmente de manera que el nodo con el último string del conjunto apunta al centinela. Si el conjunto de strings a representar fuera vacío entonces la cadena contiene un único nodo, el centinela, que se tiene a sí mismo como sucesor.

Habréis de emplear las siguientes definiciones en C++, no se pueden alterar en modo alguno. Puedes añadir nuevos métodos privados (static o no) pero **NO** atributos nuevos ni modificar la definición del struct nodo. Si quieres usar métodos privados tendrás que especificarlo e implementarlos con detalle.

```
class BagOfWords {
private:
    struct nodo {
           string palabra;
           nodo* sig;
    };
    nodo* centinela; // las palabras en la cadena de nodos
 // a continuación del centinela están
 // en orden alfabético creciente
    int num_pal;
public:
    // Ctor. Crea una BagOfWords vacía
    BagOfWords() {
         centinela = new nodo; centinela -> sig = centinela; num_pal = 0;
 }
```

```
    // Añade el string s a la BagOfWords implícita si s no estaba
    void inserta(const string& s);
    ...
}
```

Implementa el método inserta que añade una string s a la BagOfWords si s no estaba presente (y lo hace en el sitio que le corresponde para preservar el invariante de representación!); el método no hace nada si s ya estaba presente.

```
void BagOfWords(const string s) {
      nodo* p = centinela -> sig; nodo* prev = centinela;
      while ( Tria... ) {
 // Inv: todos los nodos entre los apuntados por el centinela (exclusive) y prev
 // contienen strings menores que s; prev apunta al antecesor del nodo apuntado por p
              Tria...
      }
```

```
      if ( Tria... ) {
         prev -> sig = Tria... ;
 prev -> sig -> info = s;
 prev -> sig -> sig = Tria... ;
      }
}
```

Pregunta **8**

Completa

Puntuat sobre 1,50

Temps estimat: 15 minuts

Donat un arbre general (en aquest enllaç podeu accedir a la definició de la classe) volem afegir una consultora

// Pre: l'ArbreGen implícit no és buit

int ArbreGen::aritat_maxima() const;

// Post: torna la aritat màxima d'un dels nodes de l'arbre implícit

Feu un disseny recursiu, fent servi una funció d'immersió *i_aritat_maxima(p)* que ens resolgui el mateix problema, però en el subarbre arrelat al node apuntat per p. Utilitzeu la plantilla donada.

```
template <typename T> class ArbreGen {
private:
 struct node_arbreGen {
 T info;
 vector<node_arbreGen*> seg;
 };
 node_arbreGen* primer_node;
 static node_arbreGen* copia_node_arbreGen(node_arbreGen* m);
 static void esborra_node_arbreGen(node_arbreGen* m);
```

```
// no afegiu cap atribut nou ni modifiqueu
// els de la classe ArbreGen
// completa la capçalera i implementació de la funció
// d'immersió privada
// escriu el tipus retornat, si és static o no i si
// és const o no
```

... i_aritat_maxima(...) .... {

// implementeu aquest mètode

}

public:

...

```
// Pre: l'ArbreGen implícit no és buit
// Post: torna la aritat màxima d'un dels nodes de l'arbre implícit
int aritat_maxima() const {
 // implementeu aquest mètode,
 // utilitzant la funció i_aritat_maxima
}
```

```
}
```