**Estructuras de Datos**

**Memoria PL2 2021-2022**

Marco González Martínez 09064995M

Contenido

[**Detalles y justificación de la implementación** 2](#_Toc90576876)

[Especificación de los TADs implementados, definición de las operaciones 2](#_Toc90576877)

[**Dificultades encontradas: soluciones adoptadas** 3](#_Toc90576878)

[Generación de IDs y habitaciones aleatorias (Corrección de la PL1) 3](#_Toc90576879)

[Eliminación de Pacientes del árbol 3](#_Toc90576880)

[**Métodos más destacados** 5](#_Toc90576881)

[Arbol::insertar() 5](#_Toc90576882)

[Arbol::eliminarPaciente() 5](#_Toc90576883)

[Arbol::getTamanno() 6](#_Toc90576884)

[**Comportamiento del programa** 7](#_Toc90576885)

[Menú principal del programa 7](#_Toc90576886)

[Opción M 7](#_Toc90576887)

[Opción N 8](#_Toc90576888)

[Opción O 8](#_Toc90576889)

[Opción P 8](#_Toc90576890)

[Opción Q 8](#_Toc90576891)

[Opción R 9](#_Toc90576892)

[Opción T 9](#_Toc90576893)

[Opción U 9](#_Toc90576894)

[Opción V (Incluida adicionalmente) 10](#_Toc90576895)

[Opción W (Incluida adicionalmente) 10](#_Toc90576896)

[**Bibliografía** 10](#_Toc90576897)

# **Detalles y justificación de la implementación**

# Especificación de los TADs implementados, definición de las operaciones

espec ARBOL[PACIENTES]

usa entero

parámetro

genero pacientes

genero cola

operaciones

Arbol: 🡪 arbol {genera un árbol vacío}

getTamanno: arbol 🡪 int {Calcula el tamaño del árbol, la cantidad de nodos}

contarHabImpares: arbol 🡪 int {Cuenta la cantidad de nodos que contienen un paciente que reside en una habitación impar}

altura: arbol 🡪 int {Calcula la altura del árbol}

subArbolIzq: arbol 🡪 arbol {Devuelve el subárbol Izquierdo del árbol}

subArbolDer: arbol 🡪 arbol {Devuelve el subárbol derecho del árbol}

mínimo: arbol 🡪 arbol {Devuelve el subárbol cuyo paciente reside en la habitación de número mínimo}

máximo: arbol 🡪 arbol {Devuelve el subárbol cuyo paciente reside en la habitación de número máximo}

insertar: paciente, arbol 🡪 arbol {Inserta un paciente en el árbol}

eliminarPaciente: arbol, int 🡪 arbol {Elimina el nodo cuyo paciente resida en la habitación introducida (Si existe)}

buscarPaciente: arbol, int 🡪paciente {Devuelve el paciente que reside en la habitación introducida (Si existe)}

inorden Izq: arbol 🡪 {Muestra el inorden del subárbol izquierdo}

inordenDer: arbol 🡪 {Muestra el inorden del subárbol derecho}

inorden: arbol 🡪 {Muestra el inorden del árbol}

preorden: arbol 🡪 {Muestra el preorden del árbol}

postorden: arbol 🡪 {Muestra el postorden del árbol}

mostrarHojas: arbol 🡪 {Muestra todos los nodos hojas del árbol}

dibujar: arbol 🡪 {Dibuja un grafo que representa el árbol}

finoperaciones

finespec

A la hora de incluir el TAD en el código, se utiliza también la clase NodoArbol, el cual tiene los siguientes métodos:

hayHabImpares() 🡪 bool {Calcula si hay habitaciones impares en algún subárbol del nodo}

esHoja() 🡪 bool {Calcula si el nodo se trata de un nodo hoja}

getPaciente() 🡪 puntero a paciente {Devuelve un puntero al paciente contenido por el nodo}

# **Dificultades encontradas: soluciones adoptadas**

## Generación de IDs y habitaciones aleatorias (Corrección de la PL1)

He encontrado una dificultad a la hora de crear pacientes con IDs y Números de Habitación aleatorios para resolver el problema he utilizado el objeto vector para almacenar en 4 vectores de enteros distintos los IDs que había libres en cada momento para pacientes con cada enfermedad, y de igual forma con las habitaciones en las que pueden residir los pacientes.

Con el uso de vectores, al asignarle un ID y una habitación a un paciente, se extrae un número de las listas correspondientes a su enfermedad, y sólo se volverá a insertar en el caso de que un paciente reciba el alta médica. Tanto al inicio de la ejecución como en el momento de eliminar un paciente del sistema, la lista se reordena aleatoriamente, para garantizar que la elección del ID y la habitación sea aleatoria.

## Eliminación de Pacientes del árbol

También otra dificultad fue la de eliminar un Paciente de un árbol, para lograr superar el problema, utilicé los siguientes tres métodos dentro de la clase Arbol.  
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Con estos métodos (el primero público y los otros dos privados) a partir de un número de habitación dado, conseguía eliminar el paciente que residiese en esa habitación. El funcionamiento del código es el siguiente: el método eliminarPaciente() comprueba si el valor de la habitación introducida debe de estar en su subárbol izquierdo, el derecho o en su mismo nodo. Esto se logra gracias a la peculiaridad de los árboles binarios de búsqueda que ordenan en su subárbol izquierdo los elementos menores o iguales al mismo nodo, y en el derecho los mayores. En caso de que se deba eliminar el nodo que está siendo evaluado, el método llama al método de eliminarNodo() a partir de ese nodo para devolver el nodo adecuado, si se encuentra en algún subárbol llama al método eliminarPaciente() a partir del subárbol correspondiente, si alguno de estos fuese nulo, devolvería el nodo sin alterarlo.

El método eliminarNodo() devuelve el nodo alterando el contenido del nodo, si el nodo introducido se trata de un nodo nulo, o un nodo hoja, devuelve su nodo izquierdo (para devolver siempre un nodo nulo) en el caso de que no tenga nodo izquierdo devuelve el nodo derecho ya que no es posible encontrar un valor apropiado para reemplazar el nodo alterado,   
y en caso contrario, colocaría el paciente máximo del subárbol izquierdo y lo eliminaría.

# **Métodos más destacados**

## Arbol::insertar()

Texto

Descripción generada automáticamente

El método Arbol::insertar() se encarga de insertar un paciente en el Árbol. Consta de una estructura recursiva en la que el método principal llama a otro que va insertando los elementos a partir de una raíz dada. El nodo con el elemento insertado se asignará al nodo que ha sido modificado. Como el Árbol binario de búsqueda requiere cierto orden, la forma recursiva va comparando en que subárbol de cada nodo debe insertar el nuevo elemento, y en el caso de no existir un nodo crea un nodo nuevo.

## Arbol::eliminarPaciente()

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente  
La estructura del método Arbol::eliminarNodo() Consta de un método principal y otros dos que se van llamando de forma recursiva. Para eliminarlo correctamente, este método consulta en cuál de los 2 subárboles estaría el nodo en caso de que existiese, y si encuentra el paciente borra su nodo, asegurándose de que el TAD mantiene la estructura de Árbol Binario de Búsqueda.

## Arbol::getTamanno()

Texto

Descripción generada automáticamente

El método Arbol::getTamanno() devuelve el tamaño del árbol, es decir la cantidad de nodos contenidos en él, al igual que los métodos, anteriores, su estructura es recursiva con un método principal que llama a este otro método. El método cuenta los el tamaño de los dos subárboles y el mismo nodo, para lograr la recursión.

# **Comportamiento del programa**

## Menú principal del programa

Texto

Descripción generada automáticamente

Este menú muestra un resumen de los datos almacenados en el sistema y un listado de las opciones que el sistema puede realizar desde la A hasta la W (V y W incluidas de forma adicional por el alumno). Al introducir una letra asociada a una acción del sistema, se ejecutará.

## Opción M

Texto

Descripción generada automáticamente

La opción M construye un árbol binario de búsqueda a partir de los pacientes almacenados en las diferentes listas del sistema, también dibuja un grafo representante del árbol. Aunque no muestra los enlaces del árbol, los coloca en las posiciones adecuadas y siguiendo el orden del árbol.

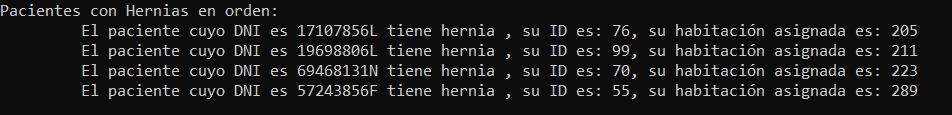
## Opción N

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

La opción N muestra los datos de todos los pacientes enfermos de apendicitis ordenados por su número de habitación de manera ascendente. Cómo existe un nodo con un paciente ficticio con número de habitación 200 y los enfermos de apendicitis descansan en habitaciones de números inferiores a 200, esto es tan fácil como mostrar el Inorden del subárbol izquierdo del árbol general.

## Opción O



La opción O muestra los datos de todos los pacientes enfermos de hernias ordenados por su número de habitación de manera ascendente. Cómo existe un nodo con un paciente ficticio con número de habitación 200 y los enfermos de hernias descansan en habitaciones de números superiores a 200, esto es tan fácil como mostrar el Inorden del subárbol derecho del árbol general.

## Opción P

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

La opción P muestra los datos de todos los pacientes siguiendo el criterio de inorden, al tratarse de un árbol binario de búsqueda esto es lo mismo que mostrar los pacientes ordenados por su número de habitación de manera ascendente.

## Opción Q

Texto

Descripción generada automáticamente con confianza media

La opción Q muestra los datos de 4 pacientes específicos:  
 - Aquel enfermo de apendicitis que resida en la habitación con menor número  
 - Aquel enfermo de apendicitis que resida en la habitación con mayor número  
 - Aquel enfermo de hernia que resida en la habitación con menor número  
 - Aquel enfermo de hernia que resida en la habitación con mayor número

## Opción R



La opción R cuenta la cantidad de pacientes que residan en habitaciones con números impares.

## Opción T

Texto

Descripción generada automáticamente con confianza media

La opción T muestra los pacientes almacenados en nodos que cumplan la condición de ser hojas, es decir que no tengan subárboles.

## Texto Descripción generada automáticamenteTexto Descripción generada automáticamenteOpción U

La opción U elimina un paciente a partir de su número de habitación pedido por consola. Mostrará también grafos del árbol antes y después de eliminar el nodo correspondiente, y en caso de que no exista el nodo indicado, no eliminará ningún paciente. En caso de tratarse de un paciente que no es un nodo hoja, para que el árbol siga manteniendo su estructura, se reubicará un paciente válido en la posición del nodo que quedaría vacío.

## Opción V (Incluida adicionalmente)

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

La opción V muestra los pacientes del árbol ordenados según el criterio de preorden. No se excluye al paciente ficticio por lo que aparece en la primera posición.

## Opción W (Incluida adicionalmente)

Imagen que contiene Texto

Descripción generada automáticamente

La opción W muestra los pacientes del árbol ordenados según el criterio de postorden. No se excluye al paciente ficticio por lo que aparece en la última posición.­­

# **Bibliografía**

Para el desarrollo de la práctica utilicé los apuntes que figuran el aula virtual de Blackboard facilitados por el profesor, además de algunas búsquedas por la web, figuran los enlaces a las páginas utilizadas y aparecen comentados también en las piezas de código que recopilé de sitios ajenos.

<https://stackoverflow.com/questions/6926433/how-to-shuffle-a-stdvector>

<https://www.geeksforgeeks.org/initialize-a-vector-in-cpp-different-ways/>