

Materia:

ESTRUCTURA DE DATOS

Docente:

María Jacinta Martinez Castillo

Integrantes:

Hernández Acevedo Brayan

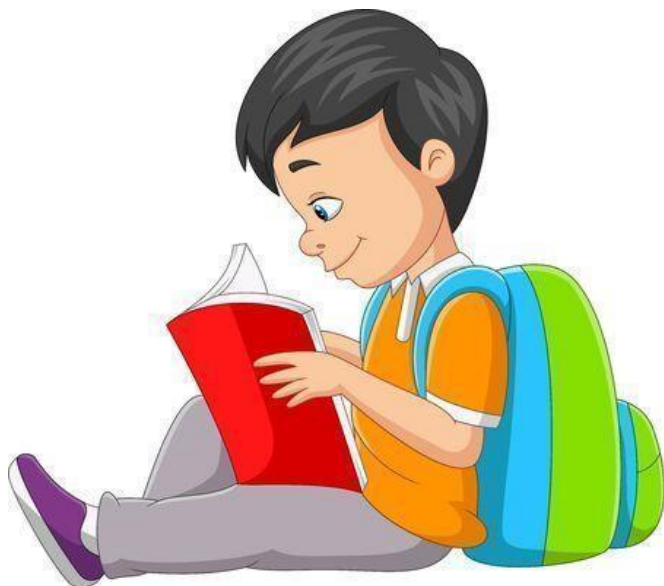
Ríos Méndez Cesar

Sentíes Robles Pedro


Tezoco Cruz Pedro

Hora:

17:00 - 18:00



REPORTE DE PRÁCTICA

	UNIDAD: Tema 6	NO. DE CONTROL Y NOMBRE: HERNANDEZ ACEVEDO BRAYAN 21010962 RIOS MENDEZ CESAR 21011029 SENTIES ROBLES PEDRO 21011047 TEZOCO CRUZ PEDRO
	NOMBRE DE PRÁCTICA: BÚSQUEDAS ESTRUCTURA DE DATOS	
ENE-JUN 2023	DOCENTE: M.A.T.I. MARÍA JACINTA MARTINEZ CASTILLO	FECHA ENTREGA: 22/05/23

INTRODUCCIÓN

En el campo de la informática y la programación, la estructura de datos juega un papel fundamental en la organización y manipulación eficiente de la información. Entre las diversas estructuras de datos, los métodos de búsqueda se destacan por su relevancia en la recuperación y localización de elementos en conjuntos de datos. Estos métodos proporcionan algoritmos y técnicas que permiten encontrar rápidamente elementos específicos dentro de una colección, optimizando así la eficiencia de las operaciones de búsqueda.

La búsqueda de elementos es una tarea común en muchas aplicaciones y sistemas, desde bases de datos hasta motores de búsqueda en Internet. La elección del método de búsqueda adecuado puede marcar una gran diferencia en el rendimiento y la velocidad de respuesta de un sistema, especialmente cuando se trabaja con conjuntos de datos grandes y complejos.

En esta exploración de los métodos de búsqueda, examinaremos algunas de las técnicas más utilizadas y ampliamente conocidas, así como también comprenderemos sus ventajas, desventajas y aplicaciones prácticas. Desde los métodos de búsqueda lineal hasta los métodos de búsqueda binaria y los árboles de búsqueda, nos adentraremos en el funcionamiento de estos algoritmos y su influencia en la eficiencia y escalabilidad de los sistemas de búsqueda.

Además, exploramos conceptos clave como el ordenamiento de datos, la complejidad algorítmica y la optimización de búsquedas. También discutiremos las situaciones en las que cada método es más apropiado y las consideraciones a tener en cuenta al seleccionar el enfoque de búsqueda más adecuado según las necesidades y características del problema.

Al comprender los métodos de búsqueda y su aplicabilidad, los desarrolladores y profesionales de la informática pueden tomar decisiones informadas sobre cómo diseñar y optimizar sistemas de búsqueda eficientes y escalables. Estos métodos son fundamentales para mejorar el rendimiento y la capacidad de respuesta de aplicaciones y sistemas en un amplio espectro de campos, incluyendo bases de datos, procesamiento de imágenes, algoritmos de ordenamiento y mucho más.

A lo largo de esta exploración, descubriremos cómo los métodos de búsqueda son una herramienta esencial en la gestión y manipulación de datos, y cómo su correcta implementación puede conducir a resultados más rápidos y eficientes en la búsqueda de información dentro de conjuntos de datos cada vez más grandes y complejos.

OBJETIVO

El objetivo de este estudio es analizar y comprender los diferentes métodos de búsqueda utilizados en estructuras de datos, con el fin de conocer sus características, ventajas y desventajas, así como su aplicabilidad en diferentes escenarios. A través de este objetivo, se busca adquirir conocimientos que permitan seleccionar el método de búsqueda más adecuado según las necesidades y características de un problema, optimizando así la eficiencia y el rendimiento en las operaciones de búsqueda de elementos dentro de conjuntos de datos. Al finalizar este estudio, se espera tener una comprensión sólida de los métodos de búsqueda y su impacto en la eficiencia de los sistemas de búsqueda, lo que facilitará la toma de decisiones informadas al diseñar y desarrollar sistemas que requieran la recuperación eficiente de información.

MARCO TEÓRICO

Los métodos de búsqueda son algoritmos y técnicas utilizados para localizar elementos específicos dentro de una estructura de datos. Estos métodos son fundamentales en la informática y la programación, ya que permiten acceder eficientemente a la información almacenada y optimizar las operaciones de búsqueda en conjuntos de datos de distintas magnitudes. En este marco teórico, explicaremos algunos de los métodos de búsqueda más comunes y ampliamente utilizados, así como los conceptos fundamentales asociados a ellos.

1. Búsqueda Lineal:

La búsqueda lineal, también conocida como búsqueda secuencial, es el método más básico y directo de búsqueda. Consiste en examinar secuencialmente cada elemento de la estructura de datos hasta encontrar el elemento deseado o llegar al final de la estructura. La búsqueda lineal es sencilla de implementar, pero puede resultar ineficiente para conjuntos de datos grandes, ya que requiere recorrer todos los elementos en el peor caso.

2. Búsqueda Binaria:

La búsqueda binaria es un método eficiente para buscar en estructuras de datos ordenadas, como arreglos o listas. Se basa en dividir repetidamente la estructura a la mitad y comparar el valor deseado con el elemento central. Con cada comparación, se reduce a la mitad el espacio de búsqueda hasta encontrar el elemento deseado o determinar que no se encuentra en la estructura. La búsqueda binaria tiene una complejidad de tiempo logarítmica y es especialmente útil en conjuntos de datos grandes.

3. Árboles de Búsqueda:

Los árboles de búsqueda, como los árboles binarios de búsqueda (BST), son estructuras de datos jerárquicas que permiten una búsqueda eficiente. Estos árboles organizan los elementos en forma de árbol, donde cada nodo tiene uno o dos nodos hijos que cumplen ciertas condiciones de ordenamiento. Esto permite realizar búsquedas rápidas utilizando comparaciones y aprovechando la estructura jerárquica del árbol. Los árboles de búsqueda son especialmente eficientes para conjuntos de datos ordenados.

4. Hashing:

El hashing es una técnica que utiliza una función de hash para asignar claves a posiciones en una estructura de datos, como una tabla hash. Estas posiciones, conocidas como "buckets" o casillas, almacenan los elementos de manera que su acceso y búsqueda sean rápidos. La función de hash transforma la clave en un valor numérico único, que se utiliza para determinar la ubicación de almacenamiento. El hashing proporciona una búsqueda de tiempo constante en promedio, lo que lo hace muy eficiente en conjuntos de datos grandes.

5. Otros métodos de búsqueda:

Además de los métodos mencionados, existen otros algoritmos y estructuras de datos utilizados para la búsqueda eficiente, como el índice invertido, el árbol B, el árbol AVL y los árboles de intervalos. Cada uno de estos métodos tiene sus propias características y aplicaciones específicas.

DESARROLLO

En esta unidad vimos el tema de Métodos de búsqueda en el cual como ejercicio principal es el del pdf que nos proporciono la maestra, en este tenemos que crear varias clases una de ellas es el de:

NODO:

```
1 package EntradaSalida;
2
3 public class Nodo<T>{
4
5     public T info;
6     public Nodo sig;
7
8     public Nodo (T info) {
9         this.info=info;
10        this.sig=null;
11    }
12
13    public T getInfo() {
14        return info;
15    }
16    public void setInfo(T info) {
17        this.info=info;
18    }
19
20    public Nodo getSig() {
21        return sig;
22    }
23    public void setSig(Nodo sig) {
24        this.sig=sig;
25    }
26 }
27
```

Este código representa la definición de una clase llamada "Nodo" que puede almacenar cualquier tipo de dato (genérico). La clase tiene dos variables públicas: "info" que almacena el valor del nodo y "sig" que representa el enlace al siguiente nodo en una estructura de datos. El constructor de la clase toma un argumento "info" y lo asigna a la variable correspondiente. Establece el enlace "sig" como nulo.

La clase también tiene métodos de acceso para obtener y establecer el valor de "info" y para obtener y establecer el enlace "sig".

ELECTRODOMÉSTICO:

```
1 package listaEnlazada;
2
3 public class Electrodomestico {
4     public static int folio = 100;
5     private String tipo;
6     protected String marca;
7     protected double potencia;
8
9
10
11 public Electrodomestico(String tipo, String marca, double potencia) {
12     this.tipo = tipo;
13     this.marca = marca;
14     this.potencia = potencia;
15     folio++;
16 }
17 public Electrodomestico() {
18     this.tipo = "";
19     this.marca = "";
20     this.potencia = 0.0;
21 }
22
23 public String getTipo() {
24     return tipo;
25 }
26
27 public void setTipo(String tipo) {
28     this.tipo = tipo;
29 }
30
31
32 }
33
34
35 public void setMarca(String marca) {
36     this.marca = marca;
37 }
38
39 public double getPotencia() {
40     return potencia;
41 }
42
43 public void setPotencia(double potencia) {
44     this.potencia = potencia;
45 }
46
47 public static int getFolio() {
48     return folio;
49 }
50
51 public String toString() {
52     return "Folio: " + folio + ", Tipo: " + tipo + ", Marca: " + marca + ", Potencia: " + potencia;
53 }
54
55 public double calcularConsumo(int horas) {
56     return potencia * horas;
57 }
58
59 public double calcularCosteConsumo(int horas, double costeHora) {
60     return potencia * horas * costeHora;
61 }
62 }
```

Este código representa la definición de una clase llamada "Electrodomestico". La clase tiene varios campos y métodos para representar las características y comportamientos de

un electrodoméstico.

Los campos de la clase incluyen:

- "folio": una variable estática que lleva un registro de los electrodomésticos creados y asigna un número de folio único a cada uno. Se inicializa en 100 y se incrementa en uno cada vez que se crea un nuevo electrodoméstico.
- "tipo": una variable de tipo String que almacena el tipo de electrodoméstico.
- "marca": una variable de tipo String que almacena la marca del electrodoméstico.
- "potencia": una variable de tipo double que almacena la potencia del electrodoméstico.

El constructor de la clase acepta los parámetros "tipo", "marca" y "potencia" y los asigna a los campos correspondientes. Además, incrementa el valor del folio en uno. También hay un constructor sin argumentos que inicializa los campos con valores predeterminados. La clase proporciona métodos de acceso (getters) y métodos de modificación (setters) para los campos "tipo", "marca" y "potencia", lo que permite obtener y establecer sus valores. El método "toString" devuelve una representación en forma de cadena de texto de un objeto "Electrodomestico", incluyendo su folio, tipo, marca y potencia.

La clase también tiene dos métodos adicionales:

- "calcularConsumo" acepta un parámetro entero "horas" y calcula el consumo de energía del electrodoméstico multiplicando su potencia por el número de horas.
- "calcularCosteConsumo" acepta dos parámetros: "horas" (entero) y "costeHora" (double), y calcula el costo de consumo del electrodoméstico multiplicando su potencia por el número de horas y el costo por hora.

LAVADORA:


```
1 package listaEnlazada;
2
3 public class Lavadora extends Electrodomestico {
4     private double precio;
5     private boolean aguaCaliente;
6
7
8     public Lavadora(String marca, double potencia) {
9         super("Electrodomestico", marca, potencia);
10        this.aguaCaliente = false;
11    }
12
13    public Lavadora(String marca, double precio, double potencia, boolean aguaCaliente) {
14        super("Electrodomestico", marca, potencia);
15        this.precio = precio;
16        this.aguaCaliente = aguaCaliente;
17    }
18
19    public Lavadora(String marca, double potencia, double precio) {
20        super("Electrodomestico", marca, potencia);
21        this.precio = precio;
22        this.aguaCaliente = false;
23    }
24
25
26
27    public double getPrecio() {
28        return precio;
29    }
30
31
32
33
34
35    public boolean isAguaCaliente() {
36        return aguaCaliente;
37    }
38
39    public void setAguaCaliente(boolean aguaCaliente) {
40        this.aguaCaliente = aguaCaliente;
41    }
42
43    @Override
44    public double calcularConsumo(int horas) {
45        if (aguaCaliente) {
46            return horas * (getPotencia() + (getPotencia() * 0.20));
47        } else {
48            return horas * getPotencia();
49        }
50    }
51
52    @Override
53    public String toString() {
54        return super.toString() + " Precio: " + precio + " Agua caliente: " + aguaCaliente;
55    }
56 }
57
```

Este código representa la definición de una clase llamada "Lavadora" que extiende la clase "Electrodomestico". La clase "Lavadora" hereda todos los campos y métodos de la clase

"Electrodomestico" y también tiene campos y métodos adicionales específicos de una lavadora.

Los campos adicionales de la clase "Lavadora" son:

- "precio": una variable de tipo double que almacena el precio de la lavadora.
- "aguaCaliente": una variable de tipo booleano que indica si la lavadora tiene la función de agua caliente o no.

La clase "Lavadora" tiene varios constructores que permiten crear objetos "Lavadora" con diferentes combinaciones de argumentos. Los constructores llaman al constructor de la clase "Electrodomestico" utilizando la palabra clave "super" para inicializar los campos heredados. También asignan valores a los campos específicos de la clase "Lavadora".

La clase proporciona métodos de acceso (getters) y métodos de modificación (setters) para los campos "precio" y "aguaCaliente", lo que permite obtener y establecer sus valores. Además, la clase "Lavadora" anula (override) dos métodos heredados de la clase "Electrodomestico":

- "calcularConsumo" calcula el consumo de energía de la lavadora en función del número de horas de uso. Si la lavadora tiene agua caliente, se agrega un 20% adicional de la potencia al cálculo.
- "toString" genera una representación en forma de cadena de texto de un objeto "Lavadora", utilizando el método "toString" heredado de la clase "Electrodomestico" y agregando información sobre el precio y si tiene agua caliente.

ListaLavadora:

```
30 import javax.swing.JOptionPane;
7
8
9 public class ListaLavadoras implements OperTDA<Lavadora> {
10     private Nodo<Lavadora> first;
11     private Nodo<Lavadora> last;
12
13     public ListaLavadoras() {
14         this.first = null;
15         this.last = null;
16     }
17
18     @Override
19     public boolean isListaVacia() {
20         return first == null;
21     }
22
23     @Override
24     public void insertFrente(Lavadora dato) {
25         Nodo<Lavadora> nod = new Nodo<>(dato);
26         if (isListaVacia()) {
27             first = nod;
28             last = nod;
29         } else {
30             nod.setSig(first);
31             first = nod;
32         }
33     }
34 }
```

Este código representa la definición de una clase llamada "ListaLavadoras" que implementa una interfaz genérica "OperTDA" con el tipo de dato "Lavadora". La clase se utiliza para representar una lista de lavadoras.

La clase tiene dos campos de tipo "Nodo<Lavadora>", llamados "first" y "last". Estos campos representan el primer y último nodo de la lista respectivamente.

El constructor de la clase inicializa los campos "first" y "last" como nulos, lo que indica que la lista está vacía al momento de su creación.

La clase implementa métodos de la interfaz "OperTDA<Lavadora>", que define operaciones para trabajar con la lista. A continuación, se explica el primer método implementado:

- "isListaVacia" es un método sobrescrito que verifica si la lista está vacía. Retorna true si el campo "first" es nulo, lo que significa que la lista no tiene ningún nodo.
- "insertFrente" es un método sobrescrito que permite insertar un nuevo nodo al frente de la lista. Recibe un parámetro "dato" de tipo "Lavadora". El método crea un nuevo

nodo "nod" con el dato proporcionado. Si la lista está vacía, asigna tanto el campo "first" como el campo "last" al nuevo nodo. Si la lista no está vacía, establece el siguiente nodo del nuevo nodo como el actual primer nodo y actualiza el campo "first" para que apunte al nuevo nodo.

```
35  @Override
36  public void insertFinal(Lavadora dato) {
37      Nodo<Lavadora> nod = new Nodo<>(dato);
38      if (isListaVacía()) {
39          first = nod;
40          last = nod;
41      } else {
42          last.setSig(nod);
43          last = nod;
44      }
45  }
46
47  @Override
48  public void eliminaLista(Nodo pos) {
49      if (pos == first) {
50          first = first.getSig();
51          if (first == null) {
52              last = null;
53          }
54      } else {
55          Nodo<Lavadora> prev = dirAntes(pos);
56          prev.setSig(pos.getSig());
57          if (pos == last) {
58              last = prev;
59          }
60      }
61  }
```

El código muestra dos métodos adicionales sobrescritos en la clase "ListaLavadoras".

- "insertFinal" es un método que permite insertar un nuevo nodo al final de la lista. Recibe un parámetro "dato" de tipo "Lavadora". El método crea un nuevo nodo "nod" con el dato proporcionado. Si la lista está vacía, asigna tanto el campo "first" como el campo "last" al nuevo nodo. Si la lista no está vacía, establece el siguiente nodo del último nodo como el nuevo nodo y actualiza el campo "last" para que apunte al nuevo nodo.
- "eliminaLista" es un método que permite eliminar un nodo de la lista. Recibe un parámetro "pos" de tipo "Nodo" que representa la posición del nodo que se desea eliminar. Si la posición es igual al primer nodo, se actualiza el campo "first" para que apunte al siguiente nodo. Si después de esta operación el campo "first" es nulo,

significa que la lista está vacía y se actualiza también el campo "last" para que sea nulo. Si la posición no es el primer nodo, se obtiene el nodo anterior a "pos" utilizando el método "dirAntes(pos)" (no se muestra en el código proporcionado) y se actualiza el enlace del nodo anterior para que omita el nodo que se desea eliminar. Además, si el nodo que se va a eliminar es el último nodo de la lista, se actualiza el campo "last" para que apunte al nodo anterior.

```
@Override
public Nodo<Lavadora> busSecuencial(int folio) {
    Nodo<Lavadora> currNode = first;
    System.out.println("Informacion del Nodo"+currNode.getInfo() + "Inf de FOLIO\n"+Electrodomestico.
    while (currNode != null && first.getInfo().folio!= folio) {
        currNode = currNode.getSig();
    }
    return currNode;
}

@Override
public String imprimeLista() {
    String listaStr = "";
    Nodo<Lavadora> currNode = first;
    while (currNode != null) {
        listaStr += currNode.getInfo().toString() + " --> ";
        currNode = currNode.getSig();
    }
    listaStr += "null";
    return listaStr;
}

@Override
public void vaciarLista() {
    first = null;
    last = null;
}
```

El código muestra tres métodos adicionales sobrescritos en la clase "ListaLavadoras".

- "busSecuencial" es un método que realiza una búsqueda secuencial en la lista en busca de un nodo con un folio específico. Recibe un parámetro "folio" de tipo entero que representa el folio a buscar. El método comienza por el primer nodo y compara el folio del nodo actual con el folio proporcionado. Si el folio coincide o encuentra un nodo con ese folio, se devuelve el nodo actual. Si no coincide, se avanza al siguiente nodo hasta recorrer toda la lista. Si se llega al final de la lista sin encontrar el folio deseado, se devuelve null.
- "imprimeLista" es un método que devuelve una representación en forma de cadena de texto de todos los nodos en la lista. El método recorre la lista comenzando por el primer nodo y concatenando la representación en forma de cadena de texto de cada

nodo al resultado final. Cada nodo se representa utilizando el método "toString()" de la clase "Lavadora". Después de recorrer toda la lista, se agrega "null" al final para indicar que no hay más nodos.

- "vaciarLista" es un método que vacía la lista, es decir, elimina todos los nodos. Simplemente asigna los valores de null a los campos "first" y "last", lo que indica que no hay nodos en la lista.

```
94         return;  
95     }  
96  
97     Lavadora lavadora = nodo.getInfo();  
98     boolean salir = false;  
99     do {  
100         int opcion = Tools.LeerInt("Elige una opción:\n" +  
101             "1. Modificar marca (actual: " + lavadora.getMarca() + ")\n" +  
102             "2. Modificar potencia (actual: " + lavadora.getPotencia() + ")\n" +  
103             "3. Modificar precio (actual: " + lavadora.getPrecio() + ")\n" +  
104             "4. Modificar agua caliente (actual: " + lavadora.isAguaCaliente() + ")\n" +  
105             "5. Salir");  
106  
107         switch (opcion) {  
108             case 1:  
109                 String nuevaMarca = Tools.LeerString("Nueva marca: ");  
110                 lavadora.setMarca(nuevaMarca);  
111                 break;  
112             case 2:  
113                 double nuevaPotencia = Tools.LeerDouble("Nueva potencia: ");  
114                 lavadora.setPotencia(nuevaPotencia);  
115                 break;  
116             case 3:  
117                 double nuevoPrecio = Tools.LeerDouble("Nuevo precio: ");  
118                 lavadora.setPrecio(nuevoPrecio);  
119                 break;  
120             case 4:  
121                 boolean nuevoAguaCaliente = Boolean.parseBoolean(JOptionPane.showInputDialog(null, "Tiene agua caliente?", Boolean.TRUE.toString()));  
122                 lavadora.setAguaCaliente(nuevoAguaCaliente);  
123                 break;  
124             case 5:  
125                 salir = true;  
126                 break;  
127             default:  
128                 Tools.imprimeMsje("Opción inválida");  
129                 break;  
130         }  
131     } while (!salir);
```

El código muestra dos métodos adicionales en la clase "ListaLavadoras".

- "modificaLavadora" es un método que permite modificar los atributos de una lavadora específica en la lista. Recibe un parámetro "folio" de tipo entero que representa el folio de la lavadora a modificar. El método comienza llamando al método "busSecuencial(folio)" para obtener el nodo correspondiente a ese folio. Si el nodo es nulo, significa que no se encontró una lavadora con el folio proporcionado, se muestra un mensaje indicando que no se encontró la lavadora y se termina el método. Si se encuentra el nodo, se obtiene la instancia de la lavadora correspondiente. Luego, se muestra un menú de opciones al usuario para elegir qué atributo de la lavadora desea modificar. Dependiendo de la opción seleccionada, se solicita al usuario el nuevo valor y se realiza la modificación correspondiente en la lavadora. El ciclo se repite hasta que el usuario elija la opción "Salir". Finalmente, se

muestra un mensaje indicando que la lavadora con el folio especificado ha sido modificada.

- "dirAntes" es un método privado que devuelve el nodo anterior a la posición proporcionada como parámetro. Recibe un parámetro "pos" de tipo "Nodo" que representa la posición actual. El método comienza desde el primer nodo y recorre la lista hasta que encuentra el nodo cuyo siguiente nodo es igual a la posición proporcionada. En ese punto, se devuelve el nodo actual, que es el nodo anterior a la posición deseada.

MENULAVADORAS:

```
1 package listaEnlazada;
2
3 import javax.swing.JOptionPane;
4
5
6
7
8
9
10 public class MenuLavadoras {
11     public static void main(String[] args) {
12
13         ListaLavadoras lista = new ListaLavadoras();
14
15         int opcion;
16         do {
17             opcion = Tools.LeerInt("Elige una opción:\n" +
18                 "1. Inserta frente\n" +
19                 "2. Inserta final\n" +
20                 "3. Eliminar\n" +
21                 "4. Modificar\n" +
22                 "5. Consulta individual\n" +
23                 "6. Salir");
24
25             switch (opcion) {
26                 case 1:
27                     Lavadora lavadoraFrente = crearLavadora();
28                     lista.inserFrente(lavadoraFrente);
29                     Tools.imprimeMsje("Lavadora insertada al frente");
30                     break;
31                 case 2:
32                     Lavadora lavadoraFinal = crearLavadora();
33                     lista.inserFinal(lavadoraFinal);
34                     Tools.imprimeMsje("Lavadora insertada al final");
35                     break;
36                 case 3:
37                     int folioEliminar = Tools.LeerInt("Ingresa el folio de la lavadora a eliminar:");
38                     Nodo<Lavadora> nodoEliminar = lista.busSecuencial(folioEliminar);
39
40                     if (nodoEliminar != null) {
41                         lista.eliminaLista(nodoEliminar);
42                         Tools.imprimeMsje("La lavadora con folio " + folioEliminar + " ha sido eliminada.");
43                     } else {
44                         Tools.imprimeMsje("No se encontró ninguna lavadora con el folio " + folioEliminar + ".");
45                     }
46                 }
47             }
48         } while (opcion != 6);
49     }
50 }
```

```
51         int folio = Tools.leerInt( "Folio de la lavadora a modificar: ");
52         lista.modificaLavadora(folio);
53     }
54     break;
55
56     case 5:
57         if (lista.isListaVacía()) {
58             Tools.imprimeMsje("La lista está vacía.");
59         } else {
60             int folioBuscar = Tools.leerInt("Ingrese el folio de la lavadora a buscar: ");
61             Nodo<Lavadora> nodoLavadora = lista.busSecuencial(folioBuscar);
62             if (nodoLavadora == null) {
63                 Tools.imprimeMsje("No se encontró ninguna lavadora con el folio " + folioBuscar);
64             } else {
65                 Lavadora lavadora = nodoLavadora.getInfo();
66                 String mensaje = "Datos de la lavadora:\n" +
67                     "Folio: " + lavadora.getFolio() + "\n" +
68                     "Marca: " + lavadora.getMarca() + "\n" +
69                     "Potencia: " + lavadora.getPotencia() + "\n" +
70                     "Precio: " + lavadora.getPrecio() + "\n" +
71                     "Agua caliente: " + (lavadora.isAguaCaliente() ? "Sí" : "No");
72                 Tools.imprimeMsje(mensaje);
73             }
74         }
75     }
76     break;
77 }
78 } while (opcion != 6);}
79
80 public static Lavadora crearLavadora() {
81     String marca = Tools.leerString("Ingrese la marca de la lavadora:");
82     double potencia = Double.parseDouble(JOptionPane.showInputDialog("Ingrese la potencia de la lavadora:"));
83     double precio = Double.parseDouble(JOptionPane.showInputDialog("Ingrese el precio de la lavadora:"));
84     boolean aguaCaliente = Boolean.parseBoolean(JOptionPane.showInputDialog("¿La lavadora funciona con agua caliente? (true/false)"));
85     Lavadora lavadora = new Lavadora(marca, potencia, precio);
86     lavadora.setAguaCaliente(aguaCaliente);
87
88     return lavadora;
89 }
```

El código muestra la clase "MenuLavadoras" que contiene el método principal "main" para ejecutar el programa de manejo de lavadoras. Aquí se presenta un menú interactivo donde el usuario puede elegir diferentes opciones para interactuar con la lista de lavadoras.

- El programa comienza creando una instancia de la clase "ListaLavadoras" llamada "lista".
- Luego, se inicia un ciclo "do-while" que muestra el menú de opciones y espera la entrada del usuario.
- Dependiendo de la opción seleccionada, se realizan diferentes acciones:
 1. Opción 1: Inserta una lavadora al frente de la lista. Se llama al método "crearLavadora" para obtener los datos de la lavadora y luego se utiliza el método "insertFrente" de la lista para insertarla al frente.
 2. Opción 2: Inserta una lavadora al final de la lista. Se llama al método "crearLavadora" para obtener los datos de la lavadora y luego se utiliza el método "insertFinal" de la lista para insertarla al final.
 3. Opción 3: Elimina una lavadora de la lista. Se solicita al usuario que ingrese el folio de la lavadora a eliminar. Luego se utiliza el método "busSecuencial"

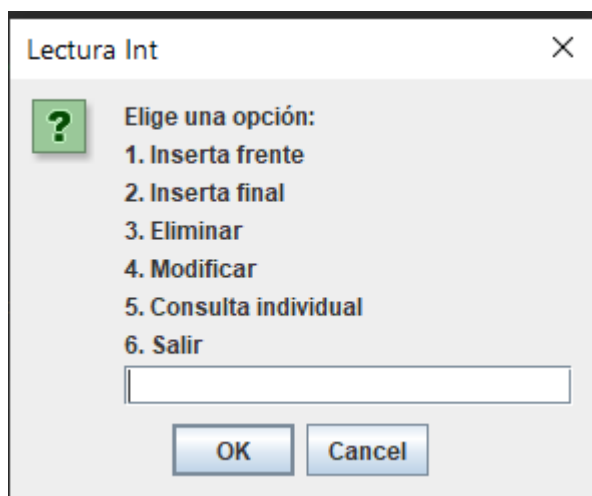
para buscar el nodo correspondiente al folio. Si se encuentra, se llama al método "eliminaLista" de la lista para eliminarlo. Se muestra un mensaje indicando si la eliminación se realizó correctamente o si no se encontró la lavadora.

4. Opción 4: Modifica los atributos de una lavadora. Se verifica si la lista está vacía. Si no lo está, se solicita al usuario que ingrese el folio de la lavadora a modificar. Luego se llama al método "modificaLavadora" de la lista para realizar la modificación de la lavadora correspondiente.
5. Opción 5: Consulta los datos de una lavadora. Se verifica si la lista está vacía. Si no lo está, se solicita al usuario que ingrese el folio de la lavadora a buscar. Luego se utiliza el método "busSecuencial" para buscar el nodo correspondiente al folio. Si se encuentra, se muestra un mensaje con los datos de la lavadora. Si no se encuentra, se muestra un mensaje indicando que no se encontró la lavadora.

- El ciclo se repite hasta que el usuario elija la opción 6, que es "Salir".
- Además, la clase "MenuLavadoras" contiene un método adicional llamado "crearLavadora", que se utiliza para solicitar al usuario los datos de una lavadora y crear una instancia de la clase "Lavadora" con esos datos.

EJECUCIÓN:

Se muestran las salidas a consola



Lectura String

?

Ingrese la marca de la lavadora:

OK Cancel

Input

?

Ingrese la potencia de la lavadora:

OK Cancel

Input

?

Ingrese el precio de la lavadora:

OK Cancel

Input

?

¿La lavadora funciona con agua caliente? (true/false)

OK Cancel

Message

i

Lavadora insertada al frente

OK

Lectura String

?

Ingrese la marca de la lavadora:

OK Cancel

Input

?

Ingrese la potencia de la lavadora:

100

OK Cancel

Input

?

Ingrese el precio de la lavadora:

50

OK Cancel

Input

?

¿La lavadora funciona con agua caliente? (true/false)

false

OK Cancel

Message

i

Lavadora insertada al final

OK

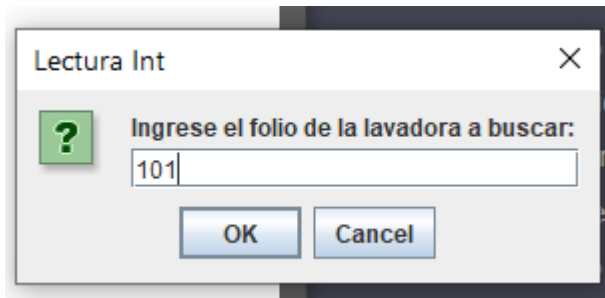
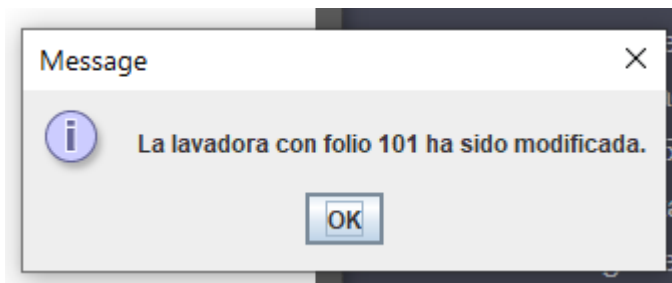
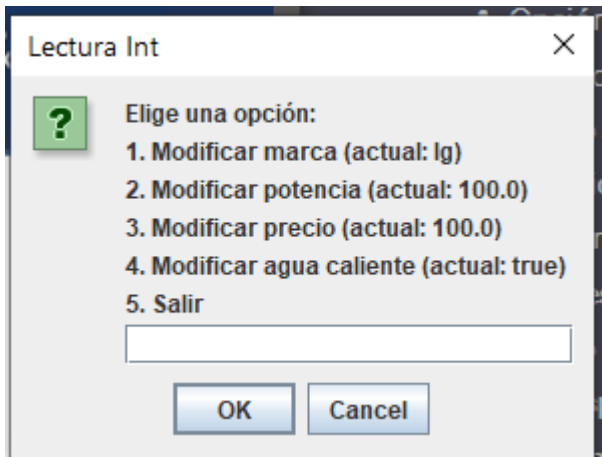
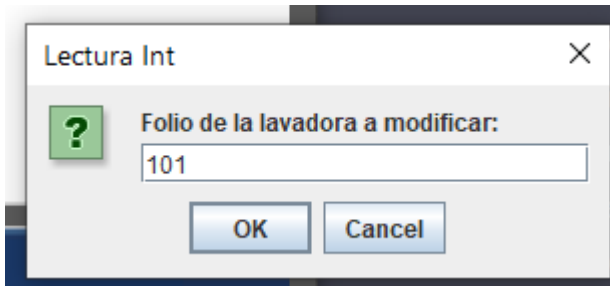
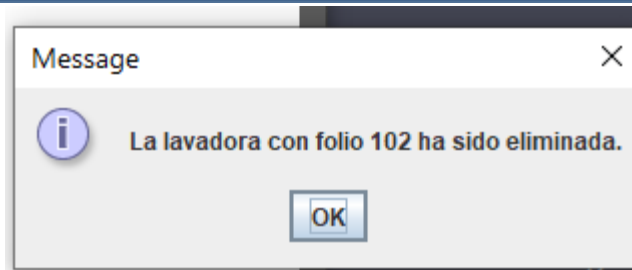
Lectura Int

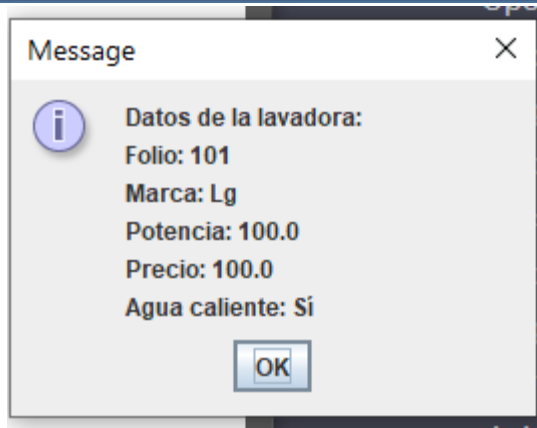
?

Ingresa el folio de la lavadora a eliminar:

102

OK Cancel





CONCLUSIÓN

En conclusión, los métodos de búsqueda desempeñan un papel crucial en la eficiencia y rendimiento de los sistemas de recuperación de información. A través de técnicas como la búsqueda lineal, la búsqueda binaria, los árboles de búsqueda, el hashing y otros métodos avanzados, es posible localizar rápidamente elementos específicos dentro de grandes conjuntos de datos.

La elección del método de búsqueda adecuado depende del tipo de estructura de datos, su organización y las características del problema. La búsqueda lineal es simple pero puede volverse ineficiente en conjuntos de datos grandes. Por otro lado, la búsqueda binaria es altamente eficiente en conjuntos de datos ordenados. Los árboles de búsqueda ofrecen una estructura jerárquica que permite búsquedas eficientes, mientras que el hashing proporciona un acceso de tiempo constante en promedio.

Es importante considerar la complejidad algorítmica de cada método y evaluar las necesidades específicas de cada situación para seleccionar el método de búsqueda más apropiado. Además, es fundamental tener en cuenta la necesidad de mantener la integridad de los datos y realizar un equilibrio entre la velocidad de búsqueda y el consumo de recursos, como el espacio en memoria.

El conocimiento y la comprensión de los métodos de búsqueda permiten a los desarrolladores y profesionales de la informática tomar decisiones informadas al diseñar y desarrollar sistemas de búsqueda eficientes. La elección correcta del método de búsqueda puede marcar una gran diferencia en la eficiencia y el rendimiento de las operaciones de búsqueda, especialmente en aplicaciones donde la recuperación de información es un componente crítico.

BIBLIOGRAFÍA

Métodos de búsqueda. (s/f). Webcindario.com. Recuperado el 22 de mayo de 2023, de <https://paginadec.webcindario.com/old/busqueda-metodos.html>

BUSQUEDAS ESTRUCTURA DE DATOS. (s/f). Applesana.es. Recuperado el 22 de mayo de 2023, de https://applesana.es/formated/aprendeprogramando.es/cursos-online/estructura_de_datos/busquedas/

Algoritmos de Búsqueda. (s/f). Edu.co. Recuperado el 22 de mayo de 2023, de <http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap02.htm>