	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación

salas A y B

Profesor : Edgar Tista García

Asignatura: EDA 2

Grupo: 4

No. Práctica : 4

Integrante: Chagoya Gonzalez Leonardo

No. Equipo de Cómputo: N/A

No. de Lista o Brigada: # Lista 08

Semestre: 2022-2

Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____

Objetivo: El estudiante identificará el comportamiento y características de los principales algoritmos de búsqueda por comparación de llaves.

Objetivo de clase: El alumno aplicará la búsqueda por comparación de llaves mediante la implementación de listas de tipos de datos primitivos y de tipos de datos abstractos

Ejercicio 1

Add es un método que se encuentra sobrecargado ya que si a este método se le pasa un parámetro insertará dicho elemento al final de la lista, por el contrario si se le pasa como parámetro la posición donde se encontrará el elemento lo insertará en dicha posicion y además se encargará de recorrer los demás elementos.

Por el contrario Set recibe dos parámetros: la posición donde se colocará el elemento y dicho elemento, pero su funcionamiento es sobreescribir la información en la posición indicada de modo que la información previa ubicada en dicha posición se perderá.

El método “subList” recibe como parámetros las posiciones del rango desde donde se va a partir el objeto lista su notación es la siguiente objeto.subList(indice1,indice2) la lista se divide desde el indice 1 hasta el (indice2 - 1) -es decir no toma en cuenta el elemento del indice2-, una vez realizado este proceso devuelve la sublista. Cabe aclarar que no realiza ninguna modificación sobre la lista original.

Para borrar elementos se encuentra el método remove() al cual se le pasa como argumento la posición del elemento que se encuentra dentro de la lista. En la página de Oracle encontré más métodos sin embargo al intentar utilizarlos en el código me generaban un error. Si se quisiera eliminar todos los elementos de la lista utilizamos el método clear().

Para saber si la lista está vacía podemos utilizar el método size el cual devolverá el número de elementos de una lista si dicho número es igual a 0 indicará que la lista está vacía.

Para buscar algún elemento podemos verificar su existencia dentro de la lista mediante el método contains() el cual recibe como parámetro el elemento a buscar y devolverá True si el valor se encuentra o False si el valor buscado no se entra dentro de la lista. De igual manera está el método indexOf() el cual recibe como parámetro el elemento a buscar y devolverá el índice de la primera aparición de dicho elemento -en caso de que este se encuentre- y devolverá -1 si esta lista no contiene el elemento.

Para probar dichos métodos se realizarán operaciones sobre lista1

```
Estado 3
15
14
16
10
500
700
69
18
66
80
 ***
```

Borrando Elementos mediante remove

```
Estado 4 (borrando elementos de la lista 1)
15
16
10
500
69
18
66
80
```

```
listal.remove(5);
listal.remove(1);
```

Se extrajo de la lista los elementos en la posición 5 y en la posición 1 en nuestro caso extrae primero el número 700 y recorre todo lo que hay frente a él para que la lista siga estando enlazada, posteriormente extrae el número 14 y realiza el mismo procedimiento -importante recordar que los elementos de una lista tienen por índices del 0 hasta (n-1)-.

Búsqueda de un Elemento

Se decidió buscar el elemento 18 que se encuentra en la lista, esperando que el método arrojase un true

```
Estado 5 buscar elementos dentro de una lista 2
Existencia del elemento: true
15
16
10
500
69
18
66
80
```

```
listal.contains(18)
```

Comprobando si la lista está vacía

```
 ***
Estado 6 comprobar si una lista esta vacia
La lista se encuentra vacia
 ***
```

Para realizar esta comprobación de lista 1 se eliminaron todos sus elementos mediante el método clear y posteriormente se verificó que el método size aplicándolo a lista1 arroje un 0

```
System.out.println("Estado 6 comprobar si una lista esta vacia ");
listal.clear(); //se borran los elementos de la lista
if( listal.size()==0){
    System.out.println("La lista se encuentra vacia");
}
imprimirLista(listal);
System.out.println(" *** ");
```

Ejercicio 2

Para realizar la búsqueda lineal sin necesidad de utilizar los métodos de java para recorrer la lista nos apoyamos del bucle for each - (Integer elem: lista) - de tal forma que la variable elem va ir tomando cada uno de los elementos de la lista

Para la búsqueda lineal retornando true o false se declaró el método *Existencia* el cual recibe como parámetros una lista y una clave del tipo entero , además retorna un valor booleano - True en caso de que el valor exista, false en caso de que el valor no se encuentre en la lista-. Dentro de la función se recorre la lista mediante el bucle for each y se compara cada elemento con la clave de tal forma que con solo un elemento que coincida el método retorna inmediatamente un true, si el ciclo termina quiere decir que el elemento no se encuentra de la lista y retorna un false

Para búsqueda lineal retornando posición se declaró el método *Posición* el cual recibe la lista y una clave, retornando un valor entero el cual será la posición en donde se encuentra el elemento a buscar . Dentro de la función se recorre la lista comparando cada elemento con la clave y se apoya de una variable auxiliar *i* que nos servirá como contador de posición, una vez que se encuentra el elemento en la lista retorna el valor en turno de *i* , si el bucle termina el elemento a buscar no se encuentra en la lista y se retorna el valor -1 -se decidió retornar dicho valor ya

que este se encuentra fuera del índice de las listas - en caso de que el elemento a buscar en la lista esté repetido sólo se retorna la posición del primero de estos elementos (este punto será importante para la búsqueda binaria retornando número de apariciones) .

Para la búsqueda lineal por retorno de número de apariciones dentro del método *AparicionElemento* se recorre la lista verificando cada elemento de ella con el elemento a buscar, teniendo como variable auxiliar un contador que indica cuántas veces aparece dicho elemento,al final será devuelto el valor de contador

Pruebas de Ejecución

```
La lista al momento es:
15
14
16
10
500
700
69
18
66
80
15
10
10
*****
Buscando el numero 10
Existencia: true
Posicion en donde se encuentra el elemento :3
Numero de veces que aparece: 3
```

```
La lista al momento es:
15
14
16
10
500
700
69
18
66
80
15
10
10
*****
Buscando el numero 700
Existencia: true
Posicion en donde se encuentra el elemento
Numero de veces que aparece: 1
```

```
La lista al momento es:
15
14
16
10
500
700
69
18
66
80
15
10
10
*****
Buscando el numero 900
Existencia: false
Posicion en donde se encuentra el elemento :-1
Numero de veces que aparece: 0
```

Ejercicio 3

Para la búsqueda binaria se decidió implementar antes un método de ordenamiento hacia la lista donde se va a realizar la búsqueda, dicho método es el ordenamiento burbuja con la implementación de mejora -una vez ordenada la lista no realizar más iteraciones-,dicho ordenamiento es ascendente .

Debido a la restricción de ocupar métodos de Java se me presentó la dificultad de cómo implementar el algoritmo de búsqueda binaria, ya que este requiere saber las posiciones de los elementos, para solucionar este problema se decidió guardar los elementos de la lista en un arreglo pues es muy fácil acceder a los elementos en las posiciones de este mismo. Dicho guardado se realiza apoyándonos de un bucle for each y una variable *i* que nos indicará el índice en turno para guardar el elemento.

```
int[]array = new int [n];
int i=0;

for(Integer elem:lista){ //pasara los elementos de la lista a un arreglo mediante ciclo for each
    array[i] = elem;
    i++;
}
```

Para la búsqueda binaria indicando la existencia de un elemento en la lista se creó el método Existencia este método recibe como parámetros la lista, una clave del tipo int y el número de elementos de la lista, este último parámetro nos ayudará al crear la instancia del arreglo mencionado previamente. Posteriormente con el arreglo mediante un bucle while se busca el elemento que se encuentra en medio y se compara con la clave en caso de que la clave sea igual el elemento se ha encontrado y se retorna un true, en caso de que la clave sea mayor al elemento medio busca un elemento intermedio hacia la derecha, en caso contrario busca un valor intermedio hacia la izquierda, una vez que ya no hay más formas de encontrar más elementos medios sale del bucle indicando que no existe un elemento en la lista y se retorna un false.

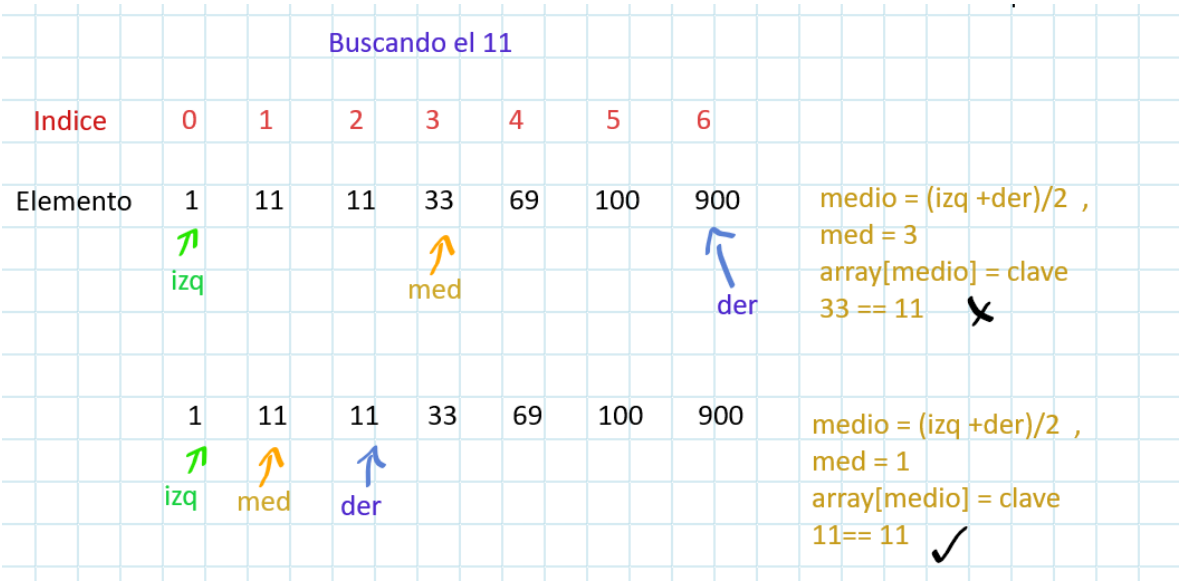
Para búsqueda binaria por retorno de número de apariciones se creó el método *AparicionElemento* el cual recibe como parámetro una lista ordenada, clave y el número de elementos de la lista . Se hace la copia de elementos de la lista a un arreglo, creamos una variable auxiliar llamada cont -nos ayudará a realizar el conteo de apariciones del elemento en la lista- y se realiza el mismo procedimiento en búsqueda binaria Existencia hasta encontrar el elemento.
Una vez que se encuentra es donde se presenta otra dificultad ya que el algoritmo no puede terminar con solo encontrar el elemento ya que este puede estar repetido, como la lista está ordenada es algo sencillo encontrar dichos elementos

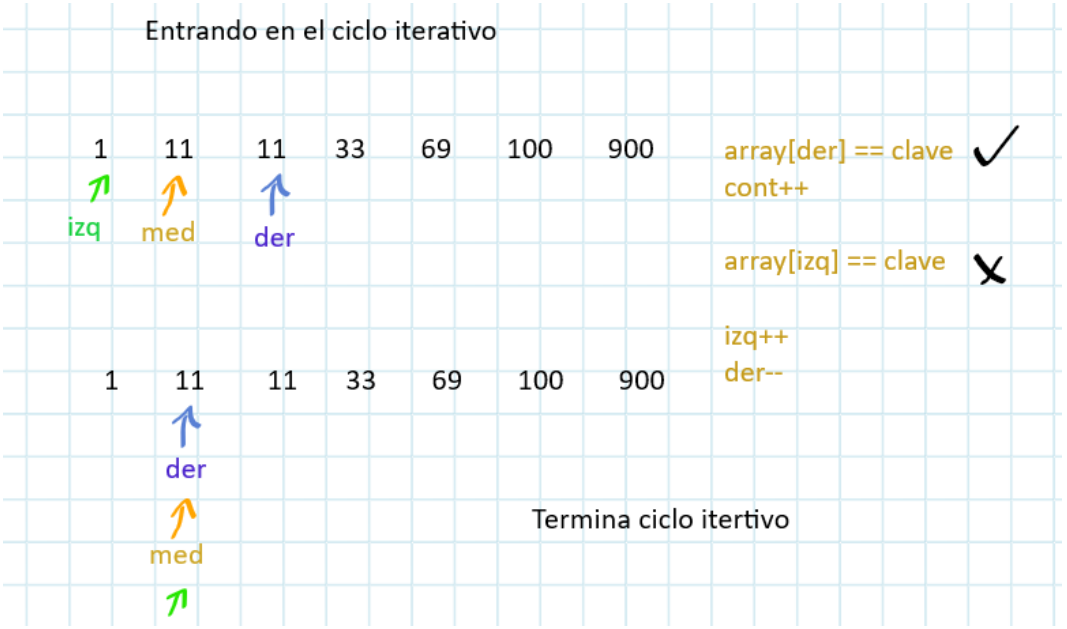
pues iterando con los extremos izq y derecha en turno hacia el indice medio buscamos dichas repeticiones del elemento clave una vez que se termina esta iteración se retorna el valor de cont.

Ciclo repetitivo para contar el número de elementos desde los extremos

```
//una vez encontrado el valor puede que exista mas de uno hacia la izquierda o derecha inmediata
while(izq!=medio && der!=medio){ //se recorre ambos extremos de izq y derecha
    if(array[der]==clave)
        cont++;
    if(array[izq]==clave)
        cont++;
    der--;
    izq++;

}
return cont;
```





```
La lista ordenada es:
10
10
10
14
15
15
16
18
66
69
80
500
700
Buscando el numero 19
Existencia: false
Numero de veces que aparece: 0
```

```
La lista ordenada es:
10
10
10
14
15
15
16
18
66
69
80
500
700
Buscando el numero 15
Existencia: true
Numero de veces que aparece: 2
```

Ejercicio 4

Para este ejercicio se creó la clase ListaAsignaturas la cual creará instancias de objetos Asignaturas, el objeto Asignatura tiene una sobrecarga de métodos constructores dependiendo los parámetros que se le pasen inicializará distintos atributos de dicho objeto, una vez realizada esta instancia de forma inmediata se procede a añadirlos a una lista la cual fue pasada como parámetro previamente.

```
//Metodos
static void Creacion(List<Asignatura> lista){
    lista.add(new Asignatura("Estructura de datos y algortimos",1317,10));
    lista.add(new Asignatura("Probabilidad",111,8,4,true));
    lista.add(new Asignatura("Cultura y Comunicacion",1222,2));
    lista.add(new Asignatura("Estructura de datos y Algortimos",2222,10,6,false));
    lista.add(new Asignatura("Ecuaciones diferenciales",1325,8));
    lista.add(new Asignatura("Calculo vectorial",1321,8,4,true));
    lista.add(new Asignatura("Probabilidad",1436,8));
    lista.add(new Asignatura("Programacion orientada a Objetos",1323,10,4,false));
}
```

Las búsquedas lineales por nombre y número de horas son muy parecidas a las búsquedas lineales del ejercicio 2 ya que se realizan mediante un bucle for each comparando cada objeto de la lista en su respectivo atributo con la clave .

Prueba de ejecución

```
Busqueda por nombre de la asignatura: Probabilidad
La lista de asignaturas es:
La asignarura es: Probabilidad
Clave: 111
Num de creditos 8
*****
La asignarura es: Probabilidad
Clave: 1436
Num de creditos 8
*****

Busqueda por numero de horas a la semana: 4
La lista de asignaturas es:
La asignarura es: Probabilidad
Clave: 111
Num de creditos 8
*****
La asignarura es: Calculo vectorial
Clave: 1321
Num de creditos 8
*****
La asignarura es: Programacion orientada a Objetos
Clave: 1323
Num de creditos 10
*****
```

Conclusiones

Los objetivos de la práctica se cumplieron de manera satisfactoria ya que con la realización de los ejercicios se puso en práctica las búsquedas lineales y binarias con sus respectivas variaciones de acuerdo al tipo de dato que regresan -booleanos, enteros e inclusive hasta listas de objetos- para la realización de dichos ejercicios se necesitaron los conceptos de búsqueda, programación básica y programación orientada a objetos (clases,objetos,atributos,métodos).

La dificultad de esta práctica para mi estuvo en analizar la forma de implementar búsqueda binaria sin ocupar métodos de java pues esta depende mucho de los índices de la lista a buscar, sin embargo esto fue solucionado copiando los elementos de la lista en un arreglo haciendo la manipulación de los índices más fácil y entender de mejor forma el funcionamiento de dicha búsqueda.

Esta práctica aunque sea de EDA 2 debido a los conocimientos de programación orientada a objetos ayuda a reforzar la teoría ,pues la solución de los ejercicios requieren dichos conocimientos haciendo que EDA2 y POO se complementen, hasta la realización de la práctica habían pequeños ejercicios en java lo cual permitía adentrarse a este lenguaje y al paradigma orientado objetos haciendo

que una vez teniendo esta práctica las dificultades se presentaran en el funcionamiento de los algoritmos y no en conceptos de POO o utilización del IDE que en mi caso en Netbeans.

Bibliografía

Oracle. (2021). Class LinkedList. Marzo 3,2022, Sitio web: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/LinkedList.html

Moisset de Esoanes Diego. (2020). Colecciones: LinkedList. Marzo 3,2022, Sitio web: https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=73&codigo=152&inicio=60