	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURAS DE DATOS Y ALGORITMOS II

Grupo: 5

No de Práctica(s): 03

Integrante(s): GÓMEZ LUNA ALEJANDRO

No. de Equipo de cómputo empleado: 52

No. de Lista o Brigada: 16

Semestre: 2020-1

Fecha de entrega: 8/SEPTIEMBRE/2019

Observaciones:

CALIFICACIÓN: _____

Objetivo de la práctica

El ESTUDIANTE IDENTIFICARÁ EL COMPORTAMIENTO Y CARCATERÍSTICAS DE LOS PRINCIPALES ALGORITMOS DE BÚSQUEDA POR COMPARACIÓN DE LLAVES.

Desarrollo

1. Ejercicios de la guía: En esta práctica, la guía nos habla de la búsqueda lineal y la búsqueda binaria, las cuales se realizan mediante la comparación de claves, en otras palabras, comparando los valores de los elementos en la lista.
   Se nos muestran los pseudocódigos y las implementaciones en Python.
   Para búsqueda lineal se nos ofrecen tres funciones que la implementan y se nos pide realizar una de tipo recursiva.
   a) busquedaLineal: Esta función va recorriendo todos los elementos de la lista mediante un ciclo de repetición for. En caso de encontrar el elemento, guarda el valor de su índice en la variable encontrado. Cabe destacar que, a pesar de encontrar el valor, seguirá buscando en la lista, por lo que, si el elemento está repetido en la lista, esta función regresará el índice de la última aparición del elemento. En caso de no estar el elemento en la lista, se retorna como valor un -1, con lo que se da a entender que no se encontró el elemento, pues es una posición inválida en una lista.
   b) busquedaLinealMejorada: Esta función es bastante parecida a la anterior, con la particularidad de que, en caso de encontrar el elemento, esta función dejará de ejecutarse y retorna el índice donde se encontró elemento, por lo que, si el elemento está repetido en la lista, esta función regresará el índice de la primera aparición del elemento.
   c) busquedaLinealCentinela: Esta función realiza la búsqueda lineal de una manera distinta. Se guarda el elemento al final de la lista en una variable temporal para después ser reemplazado por el elemento que se está buscando. Una vez hecho esto, se empezará a recorrer la lista desde el primer elemento hasta encontrar el elemento buscado. Mientras se recorre la lista, habrá una variable llamada k, la cual nos dirá el índice actual. Cuando se encuentra el elemento buscado, se imprime en pantalla el valor del índice. El último elemento vuelve a tener su valor original y se compara si la variable k tiene un valor menor al índice del último elemento o si el valor original del último elemento es el elemento buscado. En caso de cumplirse se regresa k, que es el índice donde está el elemento. En caso contrario, se regresa -1, debido a que, si k vale el índice del último elemento y el valor original del último elemento no es el buscado, quiere decir que se recorrió toda la lista sin encontrar el elemento buscado. Al último elemento solo se cambia momentáneamente su valor como frontera hasta donde se realizará la búsqueda. En esta función existía un error, pues al final se regresaba la variable encontrado, sin embargo, esta nunca fue definida con anterioridad, además de que nunca se llegará a esta sentencia, pues la función siempre retornará un valor con anterioridad.

d) busquedaLinealRecur: Esta función realiza lo mismo que la segunda función, solamente que haciendo uso de la recursión. Para realizar esta función, no se siguió el pseudocódigo de la práctica, por lo que su implementación en Python resultó de esta manera:

```
def busquedaLinealRecur(A,n,x):
    if(n==-1):
        return -1
    if(A[n]==x):
        return n
    else:
        return busquedaRecur(A,n-1,x)
```

En cuanto a la cuenta del número de iteraciones, se usaba una variable contadora que se iba incrementando en uno cada que se repetía un ciclo de repetición. Lo que resultaba en las siguiente salida en pantalla:

```
l=[10,2,5,7,9,23,57,32,17,5,2,9]
print(busquedaLineal(l,12,9))
print(busquedaLinealMejorada(l,12,9))
print(busquedaLinealCentinela(l,11,99))
print(busquedaLinealRecur(l,11,23))
```

```
12
11
5
4
11
11
-1
5
```

Para búsqueda binaria se nos ofrecen dos funciones que la implementan:

a) BusquedaBinIter: Esta función es de tipo iterativo, solamente utiliza ciclos de repetición para realizarla. El ciclo que se utiliza es el mientras, el cual solamente se ejecutará siempre y cuando el valor de izquierda sea menor o igual al de derecha, pues esto significa que todavía se puede seguir partiendo a la lista en dos. Dentro de este mientras, se guarda en la variable medio el valor del índice de la mitad de la sub-lista en la que nos encontramos. Con este valor, se compara el elemento que se encuentra en esa posición, si es el elemento buscado, se retorna este índice, en caso contrario, se compara si el elemento buscado es menor a este elemento, en caso de serlo, se empezará a buscar el elemento en la parte izquierda de la sub-lista, en caso contrario se empezará a buscar el elemento en la parte derecha.

Esto es debido a que, como la lista se encuentra ordenada, los elementos a la derecha del elemento en el medio son mayores y los elementos a la izquierda del elemento en el medio son menores. Todo el proceso anterior se repite hasta que se realice la partición y en dicha partición solamente exista un elemento. Si este no es el elemento buscado, entonces se retorna un -1.

b) BusquedaBinRecursiva: Esta función es de tipo recursiva, sigue el mismo principio que la función anteriormente descrita. Su caso base es cuando el valor de izquierda es mayor al de derecha, pues se llegará a una sub-lista que ya no se puede particionar y a partir de aquí, empezará el retroceso recursivo, sin embargo, si se llegó a este punto, entonces todas las funciones pendientes retornarán -1, puesto que en ninguna de las particiones se encontró el elemento buscado. La función se llama a sí misma siempre en un caso más simple, donde se ha particionado la lista o sub-lista a la mitad y se le pasa como argumento una de las mitades, dependiendo de el valor del elemento buscado y el valor del elemento a la mitad de dicha lista o sub-lista. Cabe destacar que, esta función va imprimiendo siempre el valor de la mitad de la lista o sub-lista donde se está realizando la función en ese momento.

La captura de implementación es la siguiente:

```
l=[2,2,5,7,9,9,10,17,23,57,68,78]
print(BusquedaBinIter(l,78,0,11))
print(BusquedaBinRecursiva(l,10,0,11))
print(BusquedaBinIter(l,10,0,11))
print(BusquedaBinRecursiva(l,60,0,11))
```

```
11
5
8
6
6
6
5
8
10
9
-1
```

2. Ejercicios propuestos:
   a) Ejercicio 1: La diferencia entre los métodos set y add radica en que, set modifica un valor que ya se encuentra en la lista, mientras que add añade un nuevo elemento a la lista. El método add tiene sobrecarga, en donde si solo se pasa como argumento el elemento a añadir, este se añadirá al final de la lista, mientras que si se le pasa una posición y el elemento, insertará al elemento en dicha posición de la lista. El método sublist nos devuelve una porción de una lista, que en este caso, es una porción de la lista1. Esta porción se obtiene al pasarle como argumentos al método los valores frontera de dicha sublista.
   Para las siguientes operaciones, existen estos métodos en Java:
      - Borrar un elemento de la lista: remove() se le especifica el índice del elemento a borrar y retorna el elemento eliminado.
      - Para saber si es vacía: isEmpty() no recibe parámetros y retorna un boolean true si la lista esta vacía o false si no lo está.
      - Para buscar algún elemento: contains() recibe el elemento a buscar como parámetro y retorna un boolean true si el objeto está en la lista o false si no lo está.

        Existen otros métodos que se pueden implementar para eliminar elementos en una lista, sin embargo, estos métodos solo se emplean cuando la lista está siendo usada para modelar una cola o una pila, o cuando la lista es creada desde un inicio como una lista ligada.

```
run:
Estado punto 1
15
25
35
45
55
65
80
 ***
Estado punto 2
15
300
25
500
35
700
45
55
65
80
 ***
```

```
 Estado punto 3
4
300
6
500
35
700
45
8
65
80
 ***
500
35
700
 ***
false
Elemento eliminado: 6
La lista esta vacia: false
Buscando el elemento 65: true
Buscando el elemento 10: false
```

b) Ejercicio 2: Para este ejercicio se pedía implementar la búsqueda lineal de diversas maneras.

   a. Para este caso, se hizo una función que regresara un valor boolean. Se utiliza un ciclo for para ir recorriendo todos los elementos de la lista. En cada iteración, se usa el método get() de lista y se le pasa como parámetro el número de iteración para obtener el elemento actual. Una vez obtenido esto, se compara con el elemento que se está buscando. Si son iguales, entonces la función retorna true. En caso contrario, el ciclo for se continúa repitiendo. Si el ciclo ha concluido, entonces se retorna un false, pues eso quiere decir que ningún elemento en la lista coincide con el elemento buscado.

   b. Para este caso, se hizo una función que regrese un valor int. Se usa la misma dinámica que en la función anterior, con la variación de que, en cuanto encuentre el elemento buscado, regresará su posición. En caso de que concluya el ciclo, se retornará un -1, pues es una posición inválida de la lista, haciendo referencia a que no se encontró el elemento buscado.

   c. Para este caso, se hizo una función que también regresa un valor int. En este caso, a diferencia de las demás funciones, en cuanto encuentra el elemento buscado, la función no retorna ningún valor, si no que aumenta en uno una variable contadora declarada con anterioridad. Es así como, al finalizar el ciclo for, se habrán contado todas las apariciones del elemento buscado. Si no aparece ninguna vez, entonces se retornará 0, pues no apareció en ninguna ocasión en la lista.

Cabe destacar que todos los métodos fueron declarados como static, con la finalidad

```
4
4
500
35
700
45
4
65
80
Comprobando los metodos creados para busqueda lineal
Se encuentra el elemento 500: true
Se encuentra el elemento 100: false
Indice del elemento 80: 8
Indice del elemento 70: -1
Cantidad de veces que esta el elemento 4: 3
Cantidad de veces que esta el elemento 5: 0
```

de no tener que crear una instancia de la clase BusquedaLineal para poder utilizarlos.

c) Ejercicio 3: Para este ejercicio se pedía implementar la búsqueda binaria de diversas maneras.

   a. Para este caso, se creó una función que regresara un valor boolean. En esta función, además de pasarle como parámetro el valor a buscar y la lista donde lo hará, se le pasa el valor del primer índice y del último índice. La función empieza por revisar si el valor de primer es mayor que el de último, pues como la función es de tipo recursivo, esto nos indicaría que ya se ha recorrido toda la lista y no se ha encontrado el valor buscado, por lo que se retorna un false. Después de esto, se obtiene el valor del centro de la lista y el elemnto en dicha posición. Se compara ese elemento con el buscado, en caso de ser iguales se retorna true. En caso contrario, se verifica si el elemento buscado es menor al elemento del centro. Si es así, se vuelve a llamar a la misma función, pasándole como valores frontera los de la sub-lista a la izquierda del centro. En caso contrario, se llama a la misma función y se le pasa como valores frontera los de la sub-lista a la derecha del centro.

   b. Para este caso, se creó una función que regresa un valor int. Esta función sigue el mismo principio que la otra función, con la particularidad de que, cuando encuentre el elemento buscado, se aumenta en uno la vairable contadora apariciones y después se.empezará a buscar en las posiciones aledañas a la posición donde se encontró el elemento buscado. Esto es debido a que puede haber más apariciones del mismo elemento y, como la lista está ordenada, todos los elementos con el mismo valor estarán juntos. Para lorgarlo, se consideraron tres casos:

      i. El primer caso es cuando el elemento se encuentra al inicio de la lista. Para este caso, se seguirá buscando el elemento a la derecha, pues a la izquierda ya no existen elementos.

      ii. El segundo caso es cuando el elemento se encuentra al final de la lista. Para este caso, se seguirá buscando el elemento a la izquierda, pues a la derecha ya no existen elementos.

      iii. El tercer caso es cuando el elemento se encuentra en cualquier lugar de la lista. Para este caso, se buscará a la derecha y a la izquierda del elemento.

   En los tres casos, si se volvía a encontrar el elemento, se aumentaba en uno la variable contadora de apariciones. Al final se retorna el valor final de apariciones.

```
4
4
4
35
45
65
80
500
700
Comprobando los metodos creados para busqueda binaria
Se encuentra el elemento 700: true
Se encuentra el elemento 800: false
Cantidad de veces que esta el elemento 4: 3
Cantidad de veces que esta el elemento 9: 0
Cantidad de veces que esta el elemento 700: 1
```

d) Ejercicio 4: Para este ejercicio se pedía implementar los mismos métodos de búsqueda lineal y de búsqueda binaria, pero con el cambio de que, en vez de buscar un entero en una lista de enteros, se buscaría el nombre de un país o el nombre de la capital de un país en una lista de objetos de tipo país.

   Primero se definió la clase Pais, la cual tiene como atributos el nombre del pais, nombre de la capital y el tamaño. Como métodos tiene uno que muestra el nombre del país y su tamaño. El otro método es de tipo constructor, en donde se inicializan los atributos.

   Se creó una lista de objetos de tipo Pais, en donde para cada elemento de la lista, se agregaba una referencia «fantasma» de un objeto de tipo Pais. Se le denomina como referencia fantasma pues no tiene un identificador para poder acceder a el, sin embargo, no se necesita para este caso, pues se pueden acceder a ellos a través de la lista. Cada que se creaba una nueva instancia de Pais, se le pasaba como argumentos al constructor el nombre del pais, nombre de la capital y tamaño respectivamente, con la finalidad de poder buscar por nombre y por capital. Es importante resaltar que desde un inicio los elementos se agregaron en orden algabético, pues el método sort no funciona para este tipo de colección.

   Por último, los métodos de búsqueda lineal y búsqueda binaria solo cambian en que el elemento a buscar es una variable de tipo String para el nombre del pais o la capital. Además, para cada elemento de la lista, se accede a su atributo nombre o capital para poder compararlo con el elemento que estamos buscando. Para evitar errores, todas las cadenas se pasaron a mayúsculas mediante el método toUpperCase() dentro de String.

   En el caso de búsqueda binaria, como no se puede comparar como tal si una cadena es mayor que la otra, solamente se comparo el primer carácter para poder saber en que lado de la lista seguir la búsqueda binaria.

```
Creando una lista de paises con su capital y dimension...
ALEMANIA BERLIN 2030.8
ARGENTINA BUENOS AIRES 2010.8
CHILE SANTIAGO 2020.8
EUA WASHINGTON DC 2000.8
MEXICO CIUDAD DE MEXICO 1990.8
MEXICO CIUDAD DE MEXICO 1990.8
Comprobando los metodos creados para busqueda lineal en la lista de paises
Se encuentra el pais Argentina: true
Se encuentra el pais Holanda: false
Indice de la capital Berlin: 0
Indice de la capital Belice: -1
Cantidad de veces que esta el pais Mexico: 2
Cantidad de veces que esta el pais Canada: 0
ALEMANIA BERLIN 2030.8
ARGENTINA BUENOS AIRES 2010.8
MEXICO CIUDAD DE MEXICO 1990.8
MEXICO CIUDAD DE MEXICO 1990.8
CHILE SANTIAGO 2020.8
EUA WASHINGTON DC 2000.8
Comprobando los metodos creados para busqueda binaria en la lista de paises
Se encuentra la capital Berlin: true
Se encuentra la capital Montevideo: false
Cantidad de veces que esta la capital Ciudad de Mexico: 2
Cantidad de veces que esta la capital Madrid: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusiones

En esta práctica se logró identificar la estructura y forma en como trabaja la búsqueda lineal, así como la búsqueda binaria, además de analizar las variantes que puede presentar cada una, como es el hecho de regresar si está presente o no un elemento en la lista, el índice de la primera aparición del elemento y la cantidad de veces que se encuentra el elemento en la lista.

Además de las implementaciones de las búsquedas, se analizaron algunos métodos presentes en la colección List del lenguaje de programación de Java. Estos métodos permitían eliminar elementos, agregar nuevos elementos, modificar elementos ya existentes, verificar si la lista estaba vacía, ordenar la lista y realizar una búsqueda sobre la lista. Con respecto al ordenamiento de la lista, Java implementa un algoritmo de ordenamiento Quick Sort con doble pivote.

Las búsquedas son bastante importantes en la vida diaria, pues siempre se quiere encontrar un elemento o un conjunto de elementos para distintas finalidades. La búsqueda lineal tiene la ventaja de que se puede realizar sobre cualquier lista, esté o no ordenada, sin embargo, tomará más tiempo en realizarse en comparación a una búsqueda binaria, pues la búsqueda lineal tiene un tiempo de complejidad lineal, mientras que la búsqueda binaria tiene un tiempo de complejidad logarítmico base 2. La desventaja de búsqueda binaria es que solamente se puede realizar sobre una lista ya ordenada.

Para los ejercicios propuestos de la práctica, se pudieron resolver, pero, en ciertos casos, como en búsqueda binaria, las funciones realizadas, aunque encontraban el elemento buscado, eran un poco extensas, por lo que, es probable que se haya podido reutilizar código de alguna manera, o poder realizar dichas funciones de una manera más eficiente.

La práctica nos permite entender las variantes de la búsqueda lineal y binaria, así como su estructura básica, además de que ofrece un análisis al uso de listas en Java, así como el empleo de diversos métodos con los que cuenta, por tanto, la práctica cumple con el objetivo propuesto y añade un extra en el uso del lenguaje Java.