INTRODUCCIÓN

En este trabajo se trabajaran con diferentes estructuras y solucionaran diferentes problemas. Se extenderán e implementaran nuevos métodos para LinkedBinaryTree, además de esto se sobre escribirán métodos ya existentes. Se creara una estructura LinearListPriorityQueue que implementara una interfaz MaxPriorityQueue en el cual se comparara los tiempos en que las dos estructuras MaxHeap y LinearListPriorityQueue, realizan las mismas operaciones. Se creara una nueva clase que encuetra la k-ésima suma de dos listas ordenadas.

Se encontrarán en el mismo las diversas soluciones y métodos de éstas junto con evidencia documental de su implementación y ejecución.

DEFINICIÓN DEL PROBLEMA

1. Extender la funcionalidad LinkedBinaryTree<T> para que maneje las siguientes funcionalidades:

    - brother(T x):

      Este método retorna el hermano del nodo que se pase como parámetro, y en dado caso de no tener hermano retornara nulo.

    - shuffle():

      Este método cambia los nodos pero mantiene la estructura del árbol

    - preOrderOutput():

      El método muestra el recorrido pre-order de manera iterativa

    - inOrderOutput():

      El método muestra el recorrido in-order de manera iterativa

    - postOrderOutput():

      El método muestra el recorrido post-order de manera iterativa

2. **LinearListPriorityQueue**

   La intención es comparar de forma experimental las estructuras MaxHeap y LinearListPriorityQueue para determinar mediante funcionalidades de medición de tiempo, la eficiencia y eficacia de cada una de ellas.

3. **KsumPair**

   Sobre dos listas distintas de números, indicar el índice de suma de números entre las dos listas en donde k sumará y enseñará la mayor de las sumas sobre los mismos resultados.

CÓDIGOS IMPORTANTES

1. Códigos importantes implementados en MyLBT

- inOrderOutput()

```
public void inOrderOutput(){
     ArrayStack<BinaryTreeNode<T>> s = new ArrayStack<>(),
           visitado=new ArrayStack<>();
     BinaryTreeNode<T> aux=root;
     s.push(aux);
     visitado.push(null);
     while(!s.isEmpty()&&aux!=null){
         aux=s.peek();
         if(s.peek()==visitado.peek()){
              System.out.print(s.pop()+" ");
              visitado.pop();
              if(aux.rightChild!=null)
                  s.push(aux.rightChild);
         }
         else{
              if(aux.leftChild!=null){
                  s.push(aux.leftChild);
                  visitado.push(aux);
              }
              else if(aux.rightChild!=null){
                  System.out.print(s.pop()+" ");
                  s.push(aux.rightChild);
              }
              else{
                  System.out.print(s.pop()+" ");
              }
          }
     }
}
```

- diámetro()

```
 public int diametro(){
     return diameter(root);
 }
 public int diameter(BinaryTreeNode<T> aux){
     if(aux==null)
           return 0;
     int lheight=height(aux.leftChild);
     int rheight=height(aux.rightChild);
      int ldiameter = diameter(aux.leftChild);
     int rdiameter = diameter(aux.rightChild);
      return max(lheight + rheight + 1, max(ldiameter,
rdiameter));
 }
```

```
int height(BinaryTreeNode<T> aux){
     if(aux == null)
          return 0;
     return 1 + max(height(aux.leftChild),
            height(aux.rightChild));
}
int max(int a, int b){
  return (a >= b)? a: b;
}
```

2. LinearListPriorityQueue

- Main()

```
public static void main (String[] args){

        LinearListPriorityQueue<Vehicle> car1 = new
LinearListPriorityQueue<>();
        MaxHeap<Vehicle> car2 = new MaxHeap<>();
        long time =     System.currentTimeMillis();
        for (int i = 0; i < 100000; i++)
                car1.put(getRandom());
        time = System.currentTimeMillis() - time;
        System.out.println("Fase 1: ");
        System.out.println("    El tiempo en nanosegundos de
car1: " + time);
        time = System.currentTimeMillis();
        for (int i = 0; i < 100000; i++)
                car2.put(getRandom());
        time = System.currentTimeMillis() - time;
        System.out.println("    El tiempo en nanosegundos de
car2: " + time);
        System.out.println("Fase 2: ");
        time = System.currentTimeMillis();
        for (int i = 0; i < 50000; i++){
                car1.removeMax();
                car1.getMax();
        }
        System.out.println("    El tiempo en nanosegundos de
car1: " + time);
        time = System.currentTimeMillis();
        for (int i = 0; i < 50000; i++){
                car2.removeMax();
                car2.getMax();
        }
        time = System.currentTimeMillis() - time;
        System.out.println("    El tiempo en nanosegundos de
car2: " + time);
```

```
        }
        public static Vehicle getRandom(){
                Random r = new Random();
                int valorDado = r.nextInt(5000000);
                /*Creando placas aleatorias*/
                String p = getChar() + getChar() + getChar() + "-" +
getNumChar() + getNumChar() + getNumChar();
                Vehicle v = new Vehicle(valorDado, p);
                return v;
        }
        public static String getChar(){
                Random l = new Random();
                /*creando numero entre 65 y 90 que es el codigo ascci del
abecedario en mayusculas*/
                int aux = l.nextInt(25)+66;
                String letra = (char)(aux) + "";
                return letra;
        }
        public static String getNumChar(){
                Random l = new Random();
                /*creando numero entre 48 y 57 que es el codigo ascci de
los numeros del 0 al 9*/
                int aux = l.nextInt(9)+49;
                String num = (char)(aux) + "";
                return num;
        }
}
```

3. KSumPair

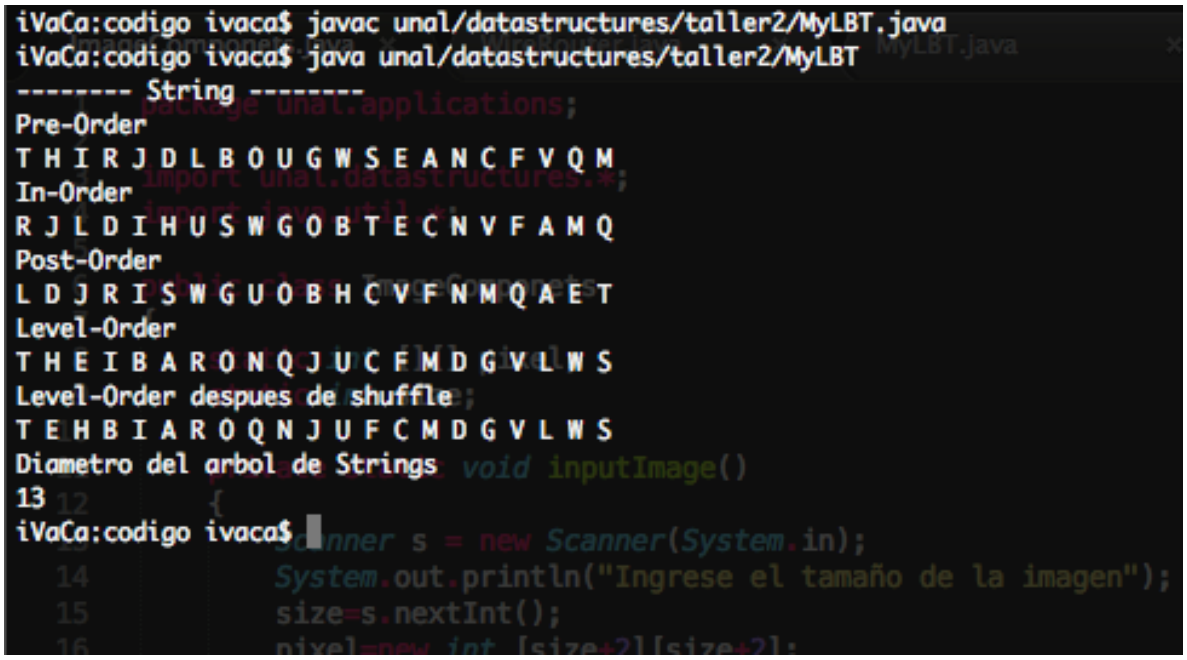
```
int max1 = 0;//m1.removeMax();
   int max2 = 0;//m2.removeMax();
   int theSlove = max1 + max2;
   for(int i = 0; i <=k; i++){
                System.out.println(theSlove);
        if (m1.isEmpty()) {
                m1 = mm1;
                mm1 = new MaxHeap<>();
                //max1 = m1.removeMax();
        }
        if (m2.isEmpty()) {
                m2 = mm2;
                mm2 = new MaxHeap<>();
                //max2 = m2.removeMax();
        }
        max1 = m1.removeMax();
        max2 = m2.removeMax();
        theSlove = max1 + max2;
        while((!m1.isEmpty() || !m2.isEmpty()) && i<k){
                System.out.println(theSlove);
                if(m1.isEmpty() && !m2.isEmpty()){
                        theSlove = max1 + m2.getMax();
                        mm2.put(m2.removeMax());
                }
```

```
                    if(m2.isEmpty() && !m1.isEmpty()){
                            theSlove = max2 + m1.getMax();
                            mm1.put(m1.removeMax());
                    }
                    if (m1.isEmpty() && m2.isEmpty()){
                            i++;
                            break;
                    }
                    int next1 = max1 + m2.getMax();
                    int next2 = max2 + m1.getMax();
                    if ( next1>next2 ){
                            theSlove = next1;
                            int aux = m2.removeMax();
                            mm2.put(aux);
                    }else{
                            theSlove = next2;
                            int aux = m1.removeMax();
                            mm1.put(aux);
                    }
                    i++;
            }
    }
    return theSlove;
```

PANTALLAZOS DE EJECUCIÓN

1. MyLBT

    - Ejecución de los métodos con tipo de dato String



    - Ejecución de los métodos con tipo de dato Integer

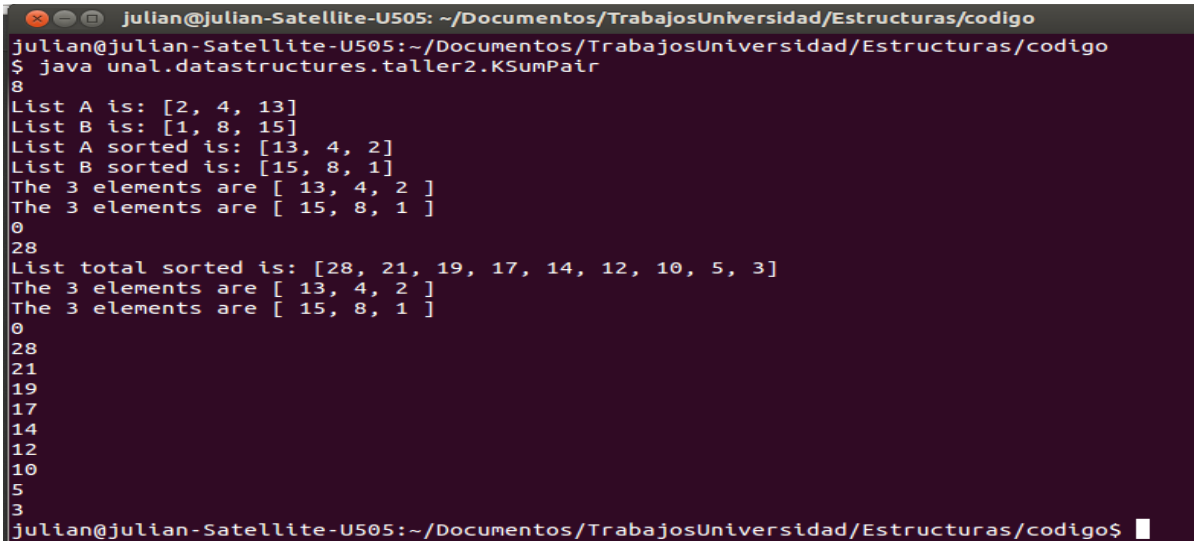
```
iVaCa:codigo ivaca$ javac unal/datastructures/taller2/MyLBT.java
iVaCa:codigo ivaca$ java unal/datastructures/taller2/MyLBT
-------- Integer --------
Pre-Order
10 3 5 25 24 89 69 22 51 6 2 55 39 15 4 7 80 90 30 14 18
In-Order
24 25 89 5 22 69 51 3 55 2 39 6 15 10 80 7 90 4 14 30 18
Post-Order
24 89 25 22 51 69 5 55 39 2 15 6 3 80 90 7 14 18 30 4 10
Level-Order
10 3 4 5 6 7 30 25 69 2 15 80 90 14 18 24 89 22 51 55 39
Level-Order despues de shuffle
10 4 3 6 5 30 7 69 25 15 2 90 80 18 14 89 24 51 22 39 55
Diametro del arbol de Strings
8
iVaCa:codigo ivaca$
```

2. LinearListPriorityQueue

```
julian@julian-Satellite-U505: ~/Documentos/TrabajosUniversidad/Estructuras/codigo
ulian@julian-Satellite-U505:~/Documentos/TrabajosUniversidad/Estructuras/codig
 javac unal/datastructures/taller2/LinearListPriorityQueue.java
ulian@julian-Satellite-U505:~/Documentos/TrabajosUniversidad/Estructuras/codig
 java unal.datastructures.taller2.LinearListPriorityQueue
ase 1:
       El tiempo en nanosegundos de car1: 52478
       El tiempo en nanosegundos de car2: 999
ase 2:
       El tiempo en nanosegundos de car1: 1415043130467
       El tiempo en nanosegundos de car2: 70
ulian@julian-Satellite-U505:~/Documentos/TrabajosUniversidad/Estructuras/codig
```

Estructura/Fase	MaxHeap	LinearListPriorityQueue	Total
Fase 1	999	53.630	54.629
Fase 2	73	1''415.050'509.398	1''415.050'509.471
Total	1.072	1''415.050'563.028	1''415.050'564.100

3. KsumPair (Se debe ingresar la posición a consultar)



DIFICULTADES

Al desarrollar el taller tuvimos diferentes dificultades, como en los métodos de pre-ordenOutput, inordenOutput y postOrdenOutput no teníamos muy claro como hacer para almacenar los nodos que se iban visitando para no volver a evaluar si tocaba visitarlo o no, para esto usamos la estructura ArrayStack en esta estructura almacenábamos los nodos ya visitados.

En la implementación de solución del problema número 3, la lógica del mismo fue un poco engorrosa y se dificultó hallar la solución eficiente del mismo; así pues, se ofrece la solución de forma en que se crean dos árboles alternos a los dos principales en donde se van almacenando en forma de MaxHeap los números cuyos la suma es más alta que el próximo.

QUÉ APRENDIMOS

Aprendimos que para solucionar los diferentes puntos podemos hacer uso de algunas estructuras ya vistas anteriormente en clase con lo cual se nos facilito el desarrollo del taller. Es éste el caso de las estructuras MaxHeap, LinkedBinaryTree y LinearListPriorityQueue principalmente y nos enseñó que el uso de las formas distintas de las estructuras de datos son eficazmente implementables a diversos ámbitos comunes y que además tienden a ser obligatoriamente entendibles para el diseñador.