

Taller 2

Estructuras de datos
Profesor: Yoan Pinzón

Hecho por:
Fredy Méndez
Jesús Monroy
Alejandro Salgado

1. Introducción

El presente documento tiene el fin de familiarizarnos con código Java un poco avanzado, específicamente de extender e implementar las clases vista: `LinkedBinaryTree<T>`, `MaxPriorityQueue<T>`, `MaxHeap<T>` explicadas previamente en la clase *Estructuras de datos*, donde crearemos algunos métodos adecuados al planteamiento del problema, utilizando enteros, strings. Para entender este documento se debe tener un conocimiento mínimo visto en clase de programación de Estructuras de datos.

Más adelante se define el problema que se va a estudiar, listando y comprendiendo el código más importante del trabajo, después se anexan imágenes del código en ejecución para verificar la utilidad y el funcionamiento correcto de este. También se mencionan las dificultades que tuvimos en desarrollar el código y la solución que encontramos. Por último se mencionara lo aprendido elaborando el trabajo.

2. Definición del problema

El taller se divide en tres partes.

La primera parte es Extender la funcionalidad de `LinkedBinaryTree<T>`, se crearan métodos que son: `brother (T x)` retorna el hermano del elemento x, sino existe retorna null; `shuffle()` que cambia el orden de los nodos, pero conservando la misma estructura inicial; `int diameter()` que retorna el diámetro del árbol binario, que es la longitud que separan dos nodos.

La segunda parte consta de crear la estructura `LinearListPriorityQueue<T>` que implementa la interface `MaxPriorityQueue<T>` donde utiliza una estructura tipo `LinearList<T>`, luego se compara la nueva estructura creada con `MaxHeap` para mirar cual es la más rápida en ejecución tanto para adicionar elementos, como para ejecutar métodos como `removeMax()` + `getMax`.

Por último

El problema 3 consistía en recibir dos arreglos ordenados y retornar las K-esima suma Mayor de estos dos arreglos y estos arreglos debían tener el mismo tamaño y contaba con ciertas restricciones tales que no se podía hacer todas las combinaciones de las sumas y también a medida que se va encontrando las parejas se debe buscar que estas no estén repetidas.

3. Código importante

```
//recorrido en preorden del arbol
void preorden(Arbol *A){
Pila *p=NULL;
printf("\n\nRecorrido en preorden: ");
```

```

do{
while(A){
printf("%i ",A->d);
push(&p,A);
A=A->izq;
}
A=pop(&p);
A=A->der;
}while(p || A);
}

```

```

//recorre el arbol en inorden
void inorden(Arbol *A){
Pila *p=NULL;
printf("\nRecorrido en inorden: ");
do{
while(A){
push(&p,A);
A=A->izq;
}
A=pop(&p);
printf("%i ",A->d);
A=A->der;
}while(p || A);
}

```

```

//recorre un arbol en postorden
void postorden(Arbol *A){
Pila *p=NULL;
printf("\nRecorrido en postorden: ");
push(&p,NULL);
while(p){
while(A){
push(&p,A);
if(A->der){
push(&p,A->der);
push(&p,NULL);
}
A=A->izq;
}
A=pop(&p);
while(A){
printf("%i ",A->d);
A=pop(&p);
}
}
}

```

```

}
if(p) if(!A) A=pop(&p);
}
}

```

Este código es importa por su complejidad y lógica ya que no podíamos resolver tan fácil y nos salía un error.

4. Pantallazos de ejecución

Se puede observar en la siguiente imagen la ejecución de los métodos propuestos de la primera parte del problema.

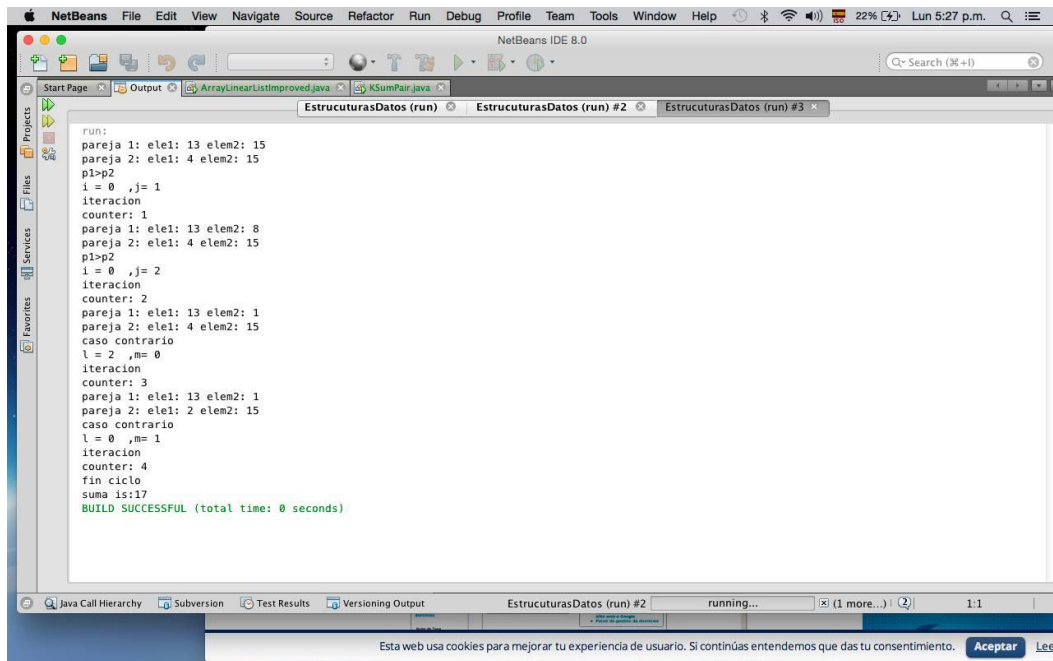
```

C:\Users\Jesus Monroy\Desktop\escritorio\estruc\codigo>javac unal\datastructures/MyLBT.java
Note: unal\datastructures\MyLBT.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\Jesus Monroy\Desktop\escritorio\estruc\codigo>java unal\datastructures/MyLBT
F R E D Y G E R A R D O M E N D E Z G O N Z A L E Z
brother the O is: M
brother the N is: E
A E E Z D Y R R N D D F M O E O G Z R A L Z G E N E
DBEAFc
ABDFc

C:\Users\Jesus Monroy\Desktop\escritorio\estruc\codigo>

```



```
run:
pareja 1: ele1: 13 elem2: 15
pareja 2: ele1: 4 elem2: 15
p1>p2
i = 0 ,j= 1
iteracion
counter: 1
pareja 1: ele1: 13 elem2: 8
pareja 2: ele1: 4 elem2: 15
p1>p2
i = 0 ,j= 2
iteracion
counter: 2
pareja 1: ele1: 13 elem2: 1
pareja 2: ele1: 4 elem2: 15
caso contrario
l = 2 ,m= 0
iteracion
counter: 3
pareja 1: ele1: 13 elem2: 1
pareja 2: ele1: 2 elem2: 15
caso contrario
l = 0 ,m= 1
iteracion
counter: 4
fin ciclo
suma 15:17
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Dificultades

Tuvimos dificultades en el inorden aunque es fácil llegar al código fue difícil pensar cómo desarrollarlo.

Entender el problema mentalmente, ya que tuve que hacer varios ejercicios en papel, para poder plantear una solución, pero ya en la parte de codificación fue más sencillo, también fue complicado el encontrar un método para agregar al Heap solo las K primeras parejas.

6. Que aprendimos

Aprendimos a implementar interface.

Es más fácil utilizar recursividad, lo iterable consume más tiempo y recursos.

Hay que hacer los talleres con tiempo.

A utilizar las ventajas que ofrece la clase MaxHeap para lograr completar ciertos tipos de problemas de una manera más eficiente.

También se aprendió acerca de implementar Comparable en la clase que va a ser usada por MaxHeap

Nota: Nos faltó terminar una estructura, podemos entregarla después cuando nos funcione sobre menos calificación de 10 puntos obviamente?