



	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 1</p>

INFORME DE TRABAJO PRÁCTICO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Estructura de datos y Algoritmos				
TÍTULO DEL TRABAJO:	HeapQueuePriority				
NÚMERO DE TRABAJO:	1	AÑO LECTIVO:	2023	NRO. SEMESTRE:	2
FECHA DE PRESENTACIÓN	25/05/2023	HORA DE PRESENTACIÓN			
INTEGRANTE (s) Mauricio Eduardo Zegarra Puma Christian Henry Casso Quispe				NOTA (0-20)	Nota colocada por el docente
DOCENTE(s): Karim Guevara Puente de la Vega					

INTRODUCCIÓN
<p>Realizamos este ejercicio con el objetivo de entender y comprender con mayor detalle todo lo visto hasta ahora en clases. Conceptos tales como Técnicas y Diseños de Algoritmos, clase genéricas. Usando como metodología el árbol binario(Heap).</p>
MARCO CONCEPTUAL
<p>Definir los conceptos utilizados o información referencial que hayan tomado en como base para resolver los problemas. Todo debe estar debidamente citado y referenciado a los documentos fuentes</p>
SOLUCIONES Y PRUEBAS
<p>EJERCICIO 5: Se construye una cola de prioridad que utilice heap como estructura de datos. LA CLASE NODE: Creamos la clase genérica node con sus parámetros E, T tal que el Key es "E" y el data "T". Para modificar sus atributos utilizamos los métodos "getKey" y "getData" -> para devolver los valores y "setKey" y "setData" para guardar nuevos valores.</p> <pre>public class Node <E, T> { private E key; private T data;</pre>

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 2</p>

```

public Node(E key, T data) {
    this.key = key;
    this.data = data;
}

public E getKey() {
    return key;
}

public void setKey(E key) {
    this.key = key;
}

public T getData() {
    return data;
}

public void setData(T data) {
    this.data = data;
}
}

```

AHORA HAREMOS LA ESTRUCTURA DE NUESTRO HEAP:

La clase "HeapQueue" implementa una cola de prioridad utilizando un hap binario.

la clase "HeapQueue" tiene los siguiente atributos:

- "queue": es un objeto de tipo "ArrayList<Node<E, T>>" que almacena los nodos en la cola de prioridad.

Métodos utilizados:

enqueue(): Agrega un nuevo nodo a la cola de prioridad. El nodo se crea utilizando los parámetros 'key' y 'data'; agregando al final del 'ArrayList'. Se realiza el proceso de reordenamiento ascendente para mantener la prioridad del heap.

dequeue(): Extrae el primer elemento de la cola de prioridad y reordena los nodos restantes para mantener la propiedad de la cola de prioridad.

front(): devolver el nodo de máxima prioridad.

back(): devolver el nodo de mínima prioridad.

toString(): devolver la representación de la cola de prioridad.

```

import java.util.ArrayList;

public class HeapQueue <E extends Comparable<E>, T>{

```

```
private ArrayList<Node<E, T>> queue;

public HeapQueue(){
    this.queue = new ArrayList<Node<E, T>>();
}

public boolean isEmpty() {
    return this.queue.size() == 0;
}

// para insertar un elemento y comparar con el padre con el método heap-max
public void enqueue (E key, T data) {
    Node<E, T> aux = new Node<E, T>(key, data);
    this.queue.add(aux);

    int size = this.queue.size();

    if (size > 1) {
        int pos = size - 1;
        int father = (pos - 1)/2;

        while
        (this.queue.get(pos).getKey().compareTo(this.queue.get(father).getKey()) > 0) {
            Node<E, T> aux2 = new Node<E,
T>(this.queue.get(father).getKey(), this.queue.get(father).getData());

            this.queue.set(father, this.queue.get(pos));
            this.queue.set(pos, aux2);

            pos = father;
            father = (pos - 1)/2;
        }
    }
}

// devuelve el nodo con la maxima prioridad
public Node<E, T> Front() {
    return this.queue.get(0);
}
```

```
// devuelve el nodo con la minima prioridad
public Node<E, T> Back() {

    Node<E, T> menor = this.queue.get(0);

    for (int i = 1; i < this.queue.size(); i++) {
        Node<E, T> actual = this.queue.get(i);

        if (menor.getKey().compareTo(actual.getKey()) > 0) {
            menor = actual;
        }
    }

    return menor;
}

// elimina el primer elemento y lo reemplaza con el último
public void dequeue() {
    this.queue.set(0, this.queue.get(this.queue.size() - 1));
    this.queue.remove(this.queue.size() - 1);

    for (int i = 0; i < this.queue.size()/2; ) {
        int left = 2*i + 1;
        int right = 2*i + 2;
        int max = left;

        if (left < this.queue.size() && right < this.queue.size()) {
            if
(this.queue.get(left).getKey().compareTo(this.queue.get(right).getKey()) < 0) {
                max = right;
            }
        }

        Node<E,T> aux = new Node<E, T> (this.queue.get(max).getKey(),
this.queue.get(max).getData());

        if (this.queue.get(i).getKey().compareTo(aux.getKey()) < 0) {
            this.queue.set(max, this.queue.get(i));
            this.queue.set(i, aux);
        }
    }
}
```

```
    }

    i = 0;
    i += max;
}

}

/* devuelve una cadena de la cola de prioridad, mostrando las claves de los
nodos en orden*/
public String toString() {
    String str = "";

    for (Node<E, T> aux : queue) {
        str += aux.getKey() + ", ";
    }



    return str;
}
}
```

LA PRUEBA DE TEST:

Un ejemplo de prueba donde agregamos elementos a nuestro ArrayList y pedimos su máxima y mínima prioridad. Aplicamos los métodos enqueue() y dequeue().

```
public class test {
    public static void main(String[] args) {
        HeapQueue<Integer, Integer> cola = new HeapQueue<Integer, Integer>();

        cola.enqueue(4,5);
        cola.enqueue(1,7);
        cola.enqueue(7,6);
        cola.enqueue(5,37);
        cola.enqueue(11,8);
        cola.enqueue(6,100);
        cola.enqueue(-4,96);
        cola.enqueue(0,3);
        cola.enqueue(-1,41);
        cola.enqueue(6,59);
    }
}
```

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 6</p>

```

cola.enqueue(3,11);
cola.enqueue(8,1);

System.out.println(cola);

cola.dequeue();

System.out.println(cola);

cola.dequeue();

System.out.println(cola);

cola.dequeue();

System.out.println(cola);

System.out.println("La mayor prioridad es de: " + cola.Front().getKey()
+ " y pertenece al elemento con la informacion: " + cola.Front().getData());

System.out.println("La menor prioridad es de: " + cola.Back().getKey()
+ " y pertenece al elemento con la informacion: " + cola.Back().getData());

}
}

```

LECCIONES APRENDIDAS Y CONCLUSIONES

En el desarrollo de este trabajo se han adquirido conocimientos valiosos sobre estructuras de datos y uso de clases genéricas. Además, se ha utilizado con éxito técnicas como el uso importante de los árboles binarios(HEAP), lo cual ha contribuido a mejorar la eficiencia y rendimiento de los algoritmos implementados.

REFERENCIAS Y BIBLIOGRAFÍA

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 7</p>

LINK DEL REPOSITORIO:

https://github.com/MauricioZegarra/EDA_PriorityQueueHeap.git