UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERIA DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Curso: Estructura de Datos y Algoritmos
Tema: HEAPS

Integrantes:
Condori Pinto Juan Jose
Orbegozo Lescano Joshua Gilbert

Docente: Karim Guevara Puente De la Vega

Repositorio: https://github.com/joshorbles/EDA_PRA_03.git

Arequipa - Perú
Junio 2023

Ejercicio:

ENUNCIADOS

EJERCICIO 5: Construya una cola de prioridad que utilice un heap como estructura de datos. Para esto realice lo siguiente:

- Implemente el TAD Heap genérico que este almacenado sobre un ArrayList con las operaciones de inserción y eliminación. Este TAD debe de ser un heap maximo.
- Implemente la clase PriorityQueueHeap generica que utilice como estructura de datos el heap desarrollado en el punto anterior. Esta clase debe tener las operaciones de una cola tales como:
    a. Enqueue (x, p) : inserta un elemento a la cola 'x' de prioridad 'p' a la cola. Como la cola esta sobre un heap, este deberá ser insertado en el heap-max y reubicado de acuerdo a su prioridad.
    b. Dequeue() : elimina el elemento de la mayor prioridad y lo devuelve. Nuevamente como la cola está sobre un heap-max, el elemento que debe ser eliminado es la raíz, por tanto, deberá sustituir este elemento por algún otro de modo que se cumpla las propiedades del heap-max.
    c. Front() : solo devuelve el elemento de mayor priorioridad.
    d. Back(): sólo devuelve el elemento de menor prioridad.

NOTA: tenga cuidado en no romper el encapsulamiento en el acceso a los atributos de las clases correspondientes.

Resolución:

```
import java.util.ArrayList;

public class TDAHeap<E extends Comparable<E>> {
    private ArrayList<E> array;

    public TDAHeap(int n) {
        this.array = new ArrayList<>(n);
    }

     public int getSize() {
        return this.array.size();
    }

    public ArrayList<E> getArray() {
        return this.array;
    }

    public void insert(E data) {
        this.array.add(data);
        int i = this.array.size();
        while (i > 1 && this.array.get(i -
1).compareTo(this.array.get(i / 2 - 1)) > 0) {
            E temp = this.array.get(i - 1);
            this.array.set(i - 1, this.array.get(i / 2 - 1));
```

```
            this.array.set(i / 2 - 1, temp);
            i /= 2;
        }
    }

    public E delete() {
        if (!this.array.isEmpty()) {
            int n = this.array.size();
            E m = this.array.get(0);
            this.array.set(0, this.array.get(n - 1));
            this.array.remove(--n);
            int j = 0;
            while (2 * j + 1 < n) {
                int k = 2 * j + 1;
                if (k + 1 < n && this.array.get(k +
1).compareTo(this.array.get(k)) > 0)
                    k++;
                if (this.array.get(j).compareTo(this.array.get(k)) > 0)
                    break;

                E temp = this.array.get(j);
                this.array.set(j, this.array.get(k));
                this.array.set(k, temp);
                j = k;
            }
            return m;
        }
        return null;
    }

    @Override
    public String toString() {
        return this.array.toString();
    }
}
```

El código proporcionado es la implementación de una estructura de datos llamada "Heap" o "Montículo" en Java. Un montículo es una estructura de datos en forma de árbol binario completo, donde cada nodo tiene un valor asociado. En este caso, el montículo se implementa utilizando un `ArrayList` llamado `array` para almacenar los elementos.

A continuación, se explica cada método de la clase TDAHeap:

- El constructor TDAHeap(int n) inicializa el ArrayList array con una capacidad inicial especificada por el parámetro n.
- El método getSize() devuelve el tamaño actual del montículo, es decir, la cantidad de elementos almacenados en el ArrayList.
- El método getArray() devuelve una referencia al ArrayList array que contiene los elementos del montículo.
- El método insert(E data) se utiliza para insertar un elemento en el montículo. El elemento se agrega al final del ArrayList array, y luego se realiza un ajuste ascendente para mantener la propiedad de montículo. El ajuste ascendente se realiza comparando el elemento insertado con su padre y, si es necesario, intercambiándolos hasta que se cumpla la propiedad de montículo.
- El método delete() se utiliza para eliminar y devolver el elemento máximo del montículo, que es el primer elemento en el ArrayList array. Primero, se intercambia el primer elemento con el último elemento en el ArrayList, luego se elimina el último elemento y se realiza un ajuste descendente para mantener la propiedad de montículo. El ajuste descendente se realiza comparando el elemento actual con sus hijos y, si es necesario, intercambiándolos hasta que se cumpla la propiedad de montículo.
- El método toString() devuelve una representación en forma de cadena del ArrayList array que contiene los elementos del montículo.

En resumen, la clase TDAHeap proporciona una implementación de un montículo utilizando un ArrayList en Java. Permite insertar elementos en el montículo y eliminar el elemento máximo.

```
public class PriorityQueueHeap<E extends Comparable<E>> {



    private TDAHeap<PriorityQueueElement<E>> queue;



    public PriorityQueueHeap(int n) {

        this.queue = new TDAHeap<>(n);

    }



    private class PriorityQueueElement<T> implements
Comparable<PriorityQueueElement<T>> {

        private T element;
```

```
        private int priority;


        public PriorityQueueElement(T element, int priority) {
            this.element = element;
            this.priority = priority;
        }


        @Override
        public int compareTo(PriorityQueueElement<T> other) {
            return Integer.compare(this.priority, other.priority);
        }


        public T getElement() {
            return element;
        }
    }


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


    public void enqueue(E element, int priority) {
```

```
        PriorityQueueElement<E> priorityElement = new
PriorityQueueElement<>(element, priority);

        this.queue.insert(priorityElement);

    }



    public E deQueue() {

        if (!this.isEmpty()) {

            PriorityQueueElement<E> elem = this.queue.delete();

            return elem != null ? elem.getElement() : null;

        }

        return null;

    }



     public E front() {

        if (!isEmpty()) {

            return this.queue.getArray().get(0).getElement();

        }

        return null;

    }



    public E back() {

        if (!this.isEmpty()) {
```

```
            return this.queue.getArray().get(this.queue.getSize() -
1).getElement();

        }

        return null;

    }



}
```

El código proporcionado es la implementación de una cola de prioridad utilizando un montículo (TDAHeap) en Java. La clase PriorityQueueHeap tiene una clase interna PriorityQueueElement que se utiliza para representar los elementos de la cola de prioridad. Cada elemento tiene un valor (element) y una prioridad asociada (priority).
A continuación se explica cada método de la clase PriorityQueueHeap:

- El constructor PriorityQueueHeap(int n) crea una instancia de la clase TDAHeap con una capacidad inicial especificada por el parámetro n. El TDAHeap se utiliza como base para implementar la cola de prioridad.
- El método isEmpty() verifica si la cola de prioridad está vacía. Devuelve true si no hay elementos en la cola de prioridad, y false en caso contrario.
- El método enqueue(E element, int priority) se utiliza para insertar un elemento en la cola de prioridad. Crea un objeto PriorityQueueElement con el elemento y la prioridad especificados, y luego lo inserta en el montículo (TDAHeap) utilizando su método insert().
- El método deQueue() se utiliza para eliminar y devolver el elemento con la mayor prioridad en la cola de prioridad. Utiliza el método delete() del montículo (TDAHeap) para eliminar el elemento máximo y luego devuelve el elemento almacenado en el PriorityQueueElement.
- El método front() devuelve el elemento con la mayor prioridad en la cola de prioridad sin eliminarlo. Obtiene el primer elemento del ArrayList subyacente del montículo utilizando el método get().
- El método back() devuelve el último elemento de la cola de prioridad, es decir, el elemento con la menor prioridad. Obtiene el último elemento del ArrayList subyacente del montículo utilizando el método get().

La implementación de esta cola de prioridad utiliza un montículo para garantizar que los elementos estén ordenados según su prioridad. Los elementos con mayor prioridad se almacenan en posiciones más altas del montículo, lo que permite un acceso rápido al elemento con la máxima prioridad.

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

        TDAHeap<Integer> heap = new TDAHeap<>(10);

        heap.insert(5);
        heap.insert(10);
        heap.insert(3);
        heap.insert(8);
        heap.insert(13);
        heap.insert(7);
        heap.insert(5);
        heap.insert(19);



        System.out.println("TDAHeap:");
        System.out.println("Array: " + heap.getArray());
        System.out.println("Tamaño: " + heap.getSize());

        Integer deletedElement = heap.delete();
        System.out.println("Elemento eliminado: " + deletedElement);
        System.out.println("Array después de la eliminación: " +
heap.getArray());
        System.out.println("Tamaño después de la eliminación: " +
heap.getSize());

        System.out.println();

        PriorityQueueHeap<String> priorityQueue = new
PriorityQueueHeap<>(10);

        priorityQueue.enqueue("Elemento 1", 3);
        priorityQueue.enqueue("Elemento 2", 1);
        priorityQueue.enqueue("Elemento 3", 2);
        priorityQueue.enqueue("Elemento 4", 4);

        System.out.println("PriorityQueueHeap:");
        System.out.println("Elemento de mayor prioridad: " +
priorityQueue.front());
        System.out.println("Elemento en el fondo: " +
priorityQueue.back());
        System.out.println("¿La cola de prioridad está vacía? " +
priorityQueue.isEmpty());
```

```
        String dequeuedElement = priorityQueue.deQueue();
        System.out.println("Elemento eliminado: " + dequeuedElement);
        System.out.println("Elemento de mayor prioridad después de la
eliminación: " + priorityQueue.front());
        System.out.println("¿La cola de prioridad está vacía? " +
priorityQueue.isEmpty());

    }
}
```

El código proporcionado es un ejemplo de cómo utilizar las clases TDAHeap y PriorityQueueHeap en el método main de una clase llamada main.
En este ejemplo, se crean instancias de TDAHeap<Integer> y PriorityQueueHeap<String>. A continuación, se realizan las siguientes operaciones:
Para el TDAHeap:

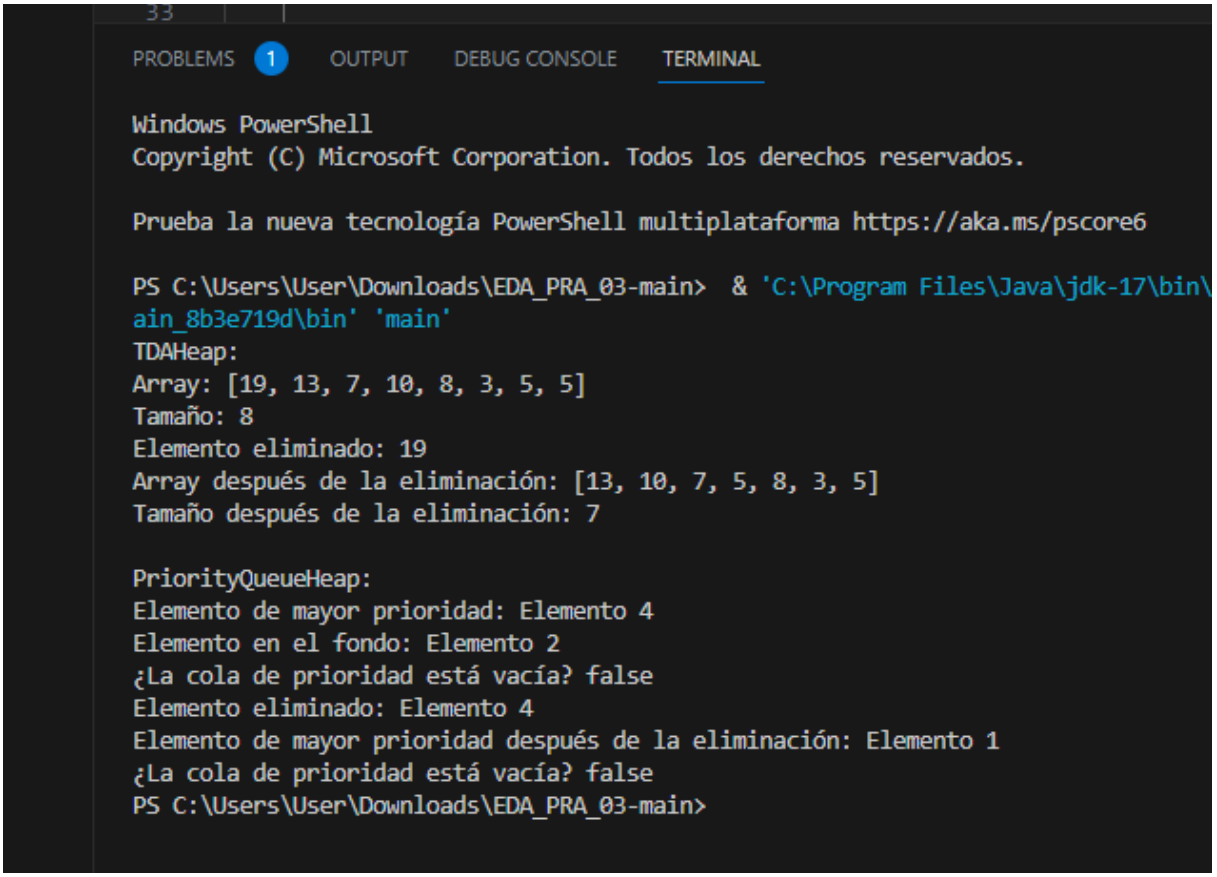
- Se insertan varios números enteros en el montículo utilizando el método insert().
- Se muestra el contenido del montículo utilizando el método getArray() y se muestra el tamaño utilizando el método getSize().
- Se elimina el elemento máximo del montículo utilizando el método delete() y se muestra el elemento eliminado.
- Se muestra el contenido del montículo después de la eliminación y se muestra el nuevo tamaño.

Para el PriorityQueueHeap:

- Se encolan varias cadenas de texto con sus respectivas prioridades utilizando el método enqueue().
- Se muestra el elemento de mayor prioridad utilizando el método front().
- Se muestra el elemento en el fondo de la cola de prioridad utilizando el método back().
- Se verifica si la cola de prioridad está vacía utilizando el método isEmpty().
- Se desencola un elemento utilizando el método deQueue() y se muestra el elemento eliminado.
- Se muestra el nuevo elemento de mayor prioridad después de la eliminación y se verifica si la cola de prioridad está vacía.

El código de ejemplo muestra cómo utilizar las clases TDAHeap y PriorityQueueHeap para trabajar con montículos y colas de prioridad en Java.

Ejecución:


33
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\User\Downloads\EDA_PRA_03-main> & 'C:\Program Files\Java\jdk-17\bin\
ain_8b3e719d\bin' 'main'
TDAHeap:
Array: [19, 13, 7, 10, 8, 3, 5, 5]
Tamaño: 8
Elemento eliminado: 19
Array después de la eliminación: [13, 10, 7, 5, 8, 3, 5]
Tamaño después de la eliminación: 7
PriorityQueueHeap:
Elemento de mayor prioridad: Elemento 4
Elemento en el fondo: Elemento 2
¿La cola de prioridad está vacía? false
Elemento eliminado: Elemento 4
Elemento de mayor prioridad después de la eliminación: Elemento 1
¿La cola de prioridad está vacía? false
PS C:\Users\User\Downloads\EDA_PRA_03-main>