Informe de la practica 03

Tema: Heap

Nota

Estudiante	Escuela	Asignatura
Javier Coronado Peña Diego Nina Suyo	Escuela Profesional de Ingeniería de Sistemas	Estructura de Datos y Algoritmos Semestre: III Código: 20231001

Practica	Tema	Duración
03	Heap	—

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 16 Junio 2023	Al 17 Junio 2023

1. Consideraciones para la entrega

- Ademas, deben de subir al aula virtual el archivo con el codigo realizado.
- El trabajo sera desarrollado en parejas, durante las horas de practica en aula.
- Las soluciones de los ejercicios deberán ser subidos a un repositorio en Github, que deben compartirlo con el profesor. Limite de plazo para cualquier actualización que se realice hasta las 12.00 del medio del sábado 17/06/2023.
- Además, un integrante del grupo debe subir al aula virtual el archivo con el código realizado, y el enlace del repositorio de trabajo. Limite de plazo hasta el término de la sesión del viernes 16 de junio de 2023.

2. Tarea

- 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:

- 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.
- 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.
- Front() : solo devuelve el elemento de mayor priorioridad.
- Back(): sólo devuelve el elemento de menor prioridad.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/Shavicho/Practica_Heaps_grupoA.git`

4. Ejercicio 3

4.1. Creando la clase generica Elemento

- Definimos una clase generica llamada Elemento. La clase elemento implementa la interfaz Comparable lo que permite comparar las instancias de esta clase entre si.
- La clase Elemento tiene 2 private, prioridad un numero entero el cual sera la prioridad del elemento y el elemento el cual es un tipo generico T, su funcion es representar al elemento real.

Listing 1: Elemento.java

```java
package heap;

public class Elemento<T extends Comparable<T>> implements Comparable<Elemento<T>> {

    private int prioridad;
    private T elemento;

    public Elemento(T elemento, int prioridad) {
        this.elemento = elemento;
        this.prioridad = prioridad;
    }

    public int getPrioridad() {
        return prioridad;
    }

    public void setPrioridad(int prioridad) {
        this.prioridad = prioridad;
    }

    public T getElemento() {
        return elemento;
    }

    public void setElemento(T elemento) {
        this.elemento = elemento;
```

```
27      }
28
29      @Override
30      public int compareTo(Elemento<T> t) {
31          return Integer.compare(this.prioridad, t.getPrioridad());
32      }
33
34      @Override
35      public String toString() {
36          return "|" + prioridad+"|";
37      }
38  }
```

- La clase sobrescribe el metodo compareTo de la interfaz Comparable, Su funcion es comparar 2 objetos elementos en funcion de sus prioridades y devuelve el resultado de la comparacion.

4.2. Creando la clase Heap

- La clase Heap implementa la interfaz HeapInterface, que define las operaciones básicas de un montículo. La implementación utiliza un ArrayList llamado heap para almacenar los elementos del Heap. A continuacion los metodos utilizados en esta clase.
- insertar(T elemento): Este metodo recibe un elemento de tipo T y lo agrega al Heap. Luego, se llama al metodo flotar(int indice) para asegurarse de que el elemento llegue hasta su posicion correcta dentro del Heap.
- eliminarMaximo(): Este metodo elimina y devuelve el elemento máximo del Heap, que es el elemento en la posicion raíz (indice 0). Primero, verifica si el Heap está vacio y, si es asi, lanza una excepción. Luego, guarda el elemento máximo en una variable maximo y lo elimina del ArrayList heap. Si el Heap no está vacío, coloca un nuevo elemento en la posición raiz y llama al metodo hundir(int indice) para asegurarse de que el elemento hunda hasta su posicion correcta dentro del monticulo. Finalmente, se devuelve el elemento máximo guardado en la variable maximo.
- minimo(): Este método devuelve el elemento minimo del Heap. Para hacer esto busca y devuelve el elemento minimo en el Heap. Si el montículo está vacío, se lanza una excepcion.
- flotar(int indice): Este metodo toma un indice como parametro y hace que el elemento en ese indice suba en el Heap hasta alcanzar la posicion correcta. El metodo compara el elemento con su padre y, si es mayor, los intercambia. Luego, actualiza el indice y continua comparando con el padre hasta que se alcance la posición correcta.
- hundir(int indice): Este metodo toma un indice como parametro y hace que el elemento baje el Heap hasta alcanzar la posición correcta. El metodo compara el elemento con sus hijos izquierdo y derecho y, si alguno de los hijos es mayor, intercambia el elemento con el hijo mayor. Luego, actualiza el índice y continua comparando con los hijos hasta que se alcance la posición correcta.

Listing 2: Heap.java

```
1   package heap;
2
3   import java.util.*;
4
5   public class Heap<T extends Comparable<T>> implements HeapInterface<T> {
6
```

```
7     private ArrayList<T> heap;
8
9     public Heap() {
10        heap = new ArrayList<T>();
11    }
12
13    @Override
14    public void insertar(T elemento) {
15        this.heap.add(elemento);
16        flotar(this.heap.size() - 1); //Este metodo debe flotar el ultimo elemento
17    }
18
19    @Override
20    public T eliminarMaximo() throws Exception {
21        if (this.heap.isEmpty()) {
22            throw new Exception("El heap esta vacio");
23        }
24        T maximo = this.heap.get(0);
25        T ultimoElemento = this.heap.remove(this.heap.size() - 1);
26
27        if (!this.heap.isEmpty()) {
28            this.heap.set(0, ultimoElemento);
29            hundir(0); // debe undir el primer elemento
30        }
31        return maximo;
32    }
33
34    @Override
35    public T obtenerMaximo() throws Exception {
36        if (this.heap.isEmpty()) {
37            throw new Exception("El heap esta vacio");
38        }
39        return this.heap.get(0);
40    }
41
42    @Override
43    public T obtenerMinimo() throws Exception {
44        if (this.heap.isEmpty()) {
45            throw new Exception("El heap esta vacio");
46        }
47        return this.minimo();
48    }
49
50    private int altura() {
51        return (int) Math.floor(Math.log(this.heap.size()) / Math.log(2));
52    }
53
54   private T minimo() {
55        if (this.heap.isEmpty()) {
56            throw new IllegalStateException("El heap esta vacio");
57        }
58
59        T minimo = this.heap.get(0);
60
61        for (int i = 1; i < this.heap.size(); i++) {
62            T elemento = this.heap.get(i);
```

```
63              if (elemento.compareTo(minimo) < 0) {
64                  minimo = elemento;
65              }
66          }
67
68          return minimo;
69      }
70
71      private int size() {
72          return this.heap.size();
73      }
74
75      @Override
76      public boolean isEmpty() {
77          return this.heap.isEmpty();
78      }
79
80      private void flotar(int indice) {
81          T elemento = this.heap.get(indice);
82          int indicePadre = getIndicePadre(indice);//devolver el indice padre
83
84          while (indice > 0 && elemento.compareTo(this.heap.get(indicePadre)) > 0) {
85              this.heap.set(indice, this.heap.get(indicePadre));
86              indice = indicePadre;
87              indicePadre = getIndicePadre(indice);//devolver el indice padre
88          }
89          this.heap.set(indice, elemento);
90      }
91
92      private void hundir(int indice) {
93          T elemento = this.heap.get(indice);
94          int indiceHijoIzquierdo = getIndiceHijoIzquierdo(indice);//devolver el indice de hijo
                izquierdp
95
96          while (indiceHijoIzquierdo < this.heap.size()) {
97              if (indiceHijoIzquierdo + 1 < this.heap.size() &&
                    this.heap.get(indiceHijoIzquierdo +
                    1).compareTo(this.heap.get(indiceHijoIzquierdo)) > 0) {
98                  indiceHijoIzquierdo++;
99              }
100
101             if (elemento.compareTo(this.heap.get(indiceHijoIzquierdo)) >= 0) {
102                 break;
103             }
104
105             this.heap.set(indice, this.heap.get(indiceHijoIzquierdo));
106             indice = indiceHijoIzquierdo;
107             indiceHijoIzquierdo = getIndiceHijoIzquierdo(indice);//devolver el indice de hijo
                    izquierdp
108         }
109
110         this.heap.set(indice, elemento);
111     }
112
113     private int getIndicePadre(int indice) {
114         return (indice - 1) / 2;
```

```
115     }
116
117     private int getIndiceHijoIzquierdo(int indice) {
118         return 2 * indice + 1;
119     }
120
121     private int getIndiceHijoDerecho(int indice) {
122         return 2 * indice + 2;
123     }
124
125     @Override
126     public String toString() {
127         String content = "{";
128         for (T element : heap) {
129             content += (element );
130         }
131         return content + "}";
132     }
133
134 }
```

4.3. Creando la clase PriorityQueueHeap

- La clase PriorityQueueHeap es una implementación de una cola de prioridad utilizando un Heap genérico. Esta clase implementa la interfaz PriQueHeaInt.
- Cada elemento en el montículo es un objeto Elemento, que contiene el elemento real y su prioridad,
- Metodos utilizados en esta clase.
- enqueue(T elemento, int prioridad): Este método recibe un elemento de tipo T y su prioridad, y lo inserta en el montículo como un objeto Elemento. Utiliza el método insertar() del montículo para realizar la inserción.
- dequeue(): Este método elimina y devuelve el elemento con la máxima prioridad de la cola. Utiliza el método eliminarMaximo() del montículo para obtener el elemento máximo y luego llama al método getElemento() en el elemento para obtener el elemento real.
- front(): Este método devuelve el elemento con la máxima prioridad de la cola sin eliminarlo. Utiliza el método obtenerMaximo() del montículo para obtener el elemento máximo y luego llama al método getElemento() en el elemento para obtener el elemento real.
- back(): Este método devuelve el elemento con la mínima prioridad de la cola. Utiliza el método obtenerMinimo() del montículo para obtener el elemento mínimo y luego llama al método getElemento() en el elemento para obtener el elemento real.

Listing 3: PriorityQueueHeap.java

```
1 package heap;
2
3 public class PriorityQueueHeap<T extends Comparable<T>> implements PriQueHeaInt<T> {
4
5     private Heap<Elemento<T>> heap;
6
```

```
    public PriorityQueueHeap() {
        this.heap = new Heap<Elemento<T>>();
    }

    @Override
    public void enqueue(T elemento, int prioridad) {
        this.heap.insertar(new Elemento<>(elemento, prioridad));
    }

    @Override
    public T dequeue() throws Exception {
        return this.heap.eliminarMaximo().getElemento();
    }

    @Override
    public T front() throws Exception {
        return this.heap.obtenerMaximo().getElemento();
    }

    @Override
    public T back() throws Exception {
        return this.heap.obtenerMinimo().getElemento();
    }

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

4.4. Main

- En el main se ejecuta el programa, ingresando datos;

Listing 4: HeapMain.java

```
package heap;

public class HeapMain {

    public static void main(String[] args) throws Exception{
        PriorityQueueHeap<String> colaPrioridad = new PriorityQueueHeap<>();
        colaPrioridad.enqueue("Descripcion 5", 5);
        colaPrioridad.enqueue("Descripcion 2", 2);
        colaPrioridad.enqueue("Descripcion 4", 4);
        colaPrioridad.enqueue("Descripcion 12", 12);
        colaPrioridad.enqueue("Descripcion 6", 6);
        colaPrioridad.enqueue("Descripcion 3", 3);
        colaPrioridad.enqueue("Descripcion 8",8 );
        colaPrioridad.enqueue("Descripcion 9", 9);
        colaPrioridad.enqueue("Descripcion 10", 10);
        colaPrioridad.enqueue("Descripcion 7", 7);


        System.out.println("hola");
```

```
20          System.out.println("|--Imprimiendo el Heap--|");
21          System.out.println(colaPrioridad);
22          System.out.println("|--Elemento de mayor prioridad--|");
23          System.out.println(colaPrioridad.front());
24          System.out.println("|--Elemento de menor prioridad--|");
25          System.out.println(colaPrioridad.back());
26          System.out.println("|--Eliminando el elemento de mayor priridad--|");
27          System.out.println(colaPrioridad.dequeue());
28          System.out.println("|--Mostrando el nuevo elemento de mayor prioridad--|");
29          System.out.println(colaPrioridad.front());
30          System.out.println("|--Mostrando el nuevo Heap modificado--|");
31          System.out.println(colaPrioridad);
32
33          colaPrioridad.enqueue("Descripcion 1", 1);
34          System.out.println("|--Agregando un nuevo elemento de menor prioridad y lo
                mostramos--|");
35          System.out.println(colaPrioridad.back());
36      }
37  }
```

- Ejecucion del programa, mostrando los metodos solicitados por el ejercicio.

```
run:
|--Imprimiendo el Heap--|
{|12||10||8||9||7||3||4||2||6||5|}
|--Elemento de mayor prioridad--|
Descripcion 12
|--Elemento de menor prioridad--|
Descripcion 2
|--Eliminando el elemento de mayor priridad--|
Descripcion 12
|--Mostrando el nuevo elemento de mayor prioridad--|
Descripcion 10
|--Mostrando el nuevo Heap modificado--|
{|10||9||8||6||7||3||4||2||5|}
|--Agregando un nuevo elemento de menor prioridad y lo mostramos--|
Descripcion 1
BUILD SUCCESSFUL (total time: 0 seconds)
```