



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

INFORME DE TRABAJO PRÁCTICO

INFORMACIÓN BÁSICA					
ASIGNATURA:	<i>Laboratorio EDA</i>				
TÍTULO DEL TRABAJO:	<i>Lab 06</i>				
NÚMERO DE TRABAJO:	<i>6</i>	AÑO LECTIVO:	<i>2</i>	NRO. SEMESTRE:	<i>3</i>
FECHA DE PRESENTACIÓN	<i>27/06/2024</i>	HORA DE PRESENTACIÓN	<i>23:21</i>		
INTEGRANTE (s) <i>Quispe Huaman, Rodrigo Ferdinand</i> <i>Quispe Maradiaga, Jeferson Jofre</i>				NOTA (0-20)	<i>Nota colocada por el docente</i>
DOCENTE(s): <i>Edson Luque</i>					

INTRODUCCIÓN
<p>Elabore un informe implementando Skip List, Splay Tree</p> <ul style="list-style-type: none"> Revise la tabla de división de trabajo, formando grupos con sus temas respectivos(máximo 2 alumnos). Estudie librerías como Graph Stream para obtener una salida gráfica de su implementación. Siempre y cuando sea viable.
MARCO CONCEPTUAL
<p>SkipList:</p> <p>Un Skip List es una estructura de datos probabilística que permite búsquedas, inserciones y eliminaciones eficientes en $O(\log n)$ tiempo promedio, utilizando múltiples niveles de listas enlazadas con elementos "saltando" sobre varios nodos, lo que mejora el rendimiento en comparación con las listas enlazadas tradicionales.</p> <p>Splay List:</p> <p>Cada vez que se accede a un nodo (ya sea para una búsqueda, inserción), el árbol se reorganiza de manera que el nodo en el que se accede se mueve a la raíz a través de una serie de operaciones de rotación llamadas "splay".</p>
SOLUCIONES Y PRUEBAS

	<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>

Parte 1 - SkipList:

Clase Nodo

Almacena sus nodos siguientes en el array, donde cada posición es el siguiente nodo respecto a cada nivel.

Java

```
public class Nodo<T extends Comparable<T>> {
    T valor;
    Nodo<T>[] next;

    public Nodo(T valor, int nivel) {
        this.valor = valor;
        this.next = new Nodo[nivel + 1];
    }
}
```

Clase SkipList():

Tenemos un atributo que delimita el límite de niveles que puede tener la lista, también tenemos la cabeza de la lista, además un entero que nos indicara el nivel actual que será utilizado en los métodos.

Java

```
public class SkipList<T extends Comparable<T>> {
    private static final int MAX_NIVEL = 6;
    private final Nodo<T> cabeza = new Nodo<>(null, MAX_NIVEL);
    private final Random rand = new Random();
    private int nivelActual = 0;
}
```

Método insertar():

Para la inserción se hace uso de una array para almacenar referencias que tienen que ser actualizadas para conservar la estructura y reglas del skipList, también tenemos un nodo actual que sera el que se mueve por la lista para encontrar el lugar correcto.

Java

```
public void insertar(T valor) {
    Nodo<T>[] actualizacion = new Nodo[MAX_NIVEL + 1];
    Nodo<T> actual = cabeza;
    for (int i = nivelActual; i >= 0; i--) {
```

```
0) {
    while (actual.next[i] != null && actual.next[i].valor.compareTo(valor) <
        actual = actual.next[i];
    }
    actualizacion[i] = actual;
}
int nivelNuevo = nivelAleatorio();
if (nivelNuevo > nivelActual) {
    for (int i = nivelActual + 1; i <= nivelNuevo; i++) {
        actualizacion[i] = cabeza;
    }
    nivelActual = nivelNuevo;
}
Nodo<T> nuevo = new Nodo<>(valor, nivelNuevo);
for (int i = 0; i <= nivelNuevo; i++) {
    nuevo.next[i] = actualizacion[i].next[i];
    actualizacion[i].next[i] = nuevo;
}
}
```

Método buscar():


Se utiliza un nodo actual que recorre los nodos de todos los niveles del SkipList hasta encontrar el valor deseado, se utiliza un for para recorrer los niveles, y se utiliza un while para recorrer cada nodo del nivel.

Java

```
public boolean buscar(T valor) {
    Nodo<T> actual = cabeza;
    for (int i = nivelActual; i >= 0; i--) {
        while (actual.next[i] != null && actual.next[i].valor.compareTo(valor) <
0) {
            actual = actual.next[i];
        }
    }
    actual = actual.next[0];
    return actual != null && actual.valor.compareTo(valor) == 0;
}
```

Método eliminar():

Se utiliza un array para almacenar las referencias que se tienen que actualizar después de la eliminación de un nodo para que el SkipList siga cumpliendo sus reglas, primero se recorre la lista buscando el valor a eliminar, y después se actualiza la información de los nodos que estaban relacionados al nodo eliminado.

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 4

Java

```

public void eliminar(T valor) {
    Nodo<T>[] actualizacion = new Nodo[MAX_NIVEL + 1];
    Nodo<T> actual = cabeza;
    for (int i = nivelActual; i >= 0; i--) {
        while (actual.next[i] != null && actual.next[i].valor.compareTo(valor) <
0) {
            actual = actual.next[i];
        }
        actualizacion[i] = actual;
    }
    actual = actual.next[0];
    if (actual != null && actual.valor.compareTo(valor) == 0) {
        for (int i = 0; i <= nivelActual; i++) {
            if (actualizacion[i].next[i] != actual) break;
            actualizacion[i].next[i] = actual.next[i];
        }
        while (nivelActual > 0 && cabeza.next[nivelActual] == null) {
            nivelActual--;
        }
    }
}

```

Parte 2: Splay tree



-Para esta ocasión reutilice código anterior del BST y modifique bastante el avl que me ayudaba con algunos métodos

```

J BST.java
J Node.java
J SplayTree.java
J Test.java

```

- Node.java
 - public class Node<T> {
 - protected T data;
 - protected Node<T> left;
 - protected Node<T> right;
 - protected Node<T> parent;
 -
 - public Node(T data){
 - this.data = data;
 - this.left = null;
 - this.right = null;

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">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: 5</p>

```

○      this.parent = null;
○      }
○
○      public Node<T> getLeft () {return this.left ;}
○      public Node<T> getRight(){return this.right;}
○      public T  getData () {return this.data ;}
○
○      public void setLeft (Node<T> left ) {this.left = left ;}
○      public void setRight(Node<T> right){this.right = right;}
○      public void setData ( T data ) {this.data = data ;}
○
○      public String toString(){
○          return data.toString();
○      }
○      public Node<T> getParent() {
○          return parent;
○      }
○
○      public void setParent(Node<T> parent) {
○          this.parent = parent;
○      }
○  }
○

```

estos son mis metodos para la clase nodo aumentando la direccion parent para que sea mas sencillo subir

- SplayTree.java

Mis principales metodos son:

- protected NodeSplay splay(NodeSplay node):
 - Este método mueve el nodo accedido recientemente a la raíz del árbol mediante una serie de rotaciones (zig, zig-zig, zig-zag)
 - private void splay(NodeSplay node) {
 - while (node != null && node != this.getHead()) {
 - NodeSplay parent = (NodeSplay) parentSplay(node);
 - NodeSplay grandParent = (NodeSplay) (parent != null ? parentSplay(parent) : null);
 -
 - if (parent == null) {
 - return;
 - }
 -
 - if (grandParent == null) {
 - if (node == parent.getLeft()) {
 - rotateSR(parent);

```

    •         } else {
    •             rotateSL(parent);
    •         }
    •     } else if (node == parent.getLeft() && parent == grandParent.getLeft()) {
    •         rotateSR(grandParent);
    •         rotateSR(parent);
    •     } else if (node == parent.getRight() && parent == grandParent.getRight()) {
    •         rotateSL(grandParent);
    •         rotateSL(parent);
    •     } else if (node == parent.getRight() && parent == grandParent.getLeft()) {
    •         rotateSL(parent);
    •         rotateSR(grandParent);
    •     } else {
    •         rotateSR(parent);
    •         rotateSL(grandParent);
    •     }
    • }
    • }
    • }
    • }

○ protected NodeSplay rotateDR(NodeSplay node);
○ protected NodeSplay rotateSR(NodeSplay node);
    • private NodeSplay rotateSL(NodeSplay node) { // izquierda
    •     NodeSplay son = (NodeSplay) node.getRight();
    •     node.setRight(son.getLeft());
    •     if (son.getLeft() != null) {
    •         son.getLeft().setParent(node);
    •     }
    •     son.setLeft(node);
    •     son.setParent(node.getParent());
    •     if (node.getParent() == null) {
    •         this.setHead(son);
    •     } else if (node == node.getParent().getLeft()) {
    •         node.getParent().setLeft(son);
    •     } else {
    •         node.getParent().setRight(son);
    •     }
    •     node.setParent(son);
    •     return son;
    • }

    • private NodeSplay rotateSR(NodeSplay node) { // derecha
    •     NodeSplay son = (NodeSplay) node.getLeft();
    •     node.setLeft(son.getRight());
    •     if (son.getRight() != null) {
    •         son.getRight().setParent(node);
    •     }
    •     son.setRight(node);

```

```
● son.setParent(node.getParent());  
● if (node.getParent() == null) {  
●     this.setHead(son);  
● } else if (node == node.getParent().getLeft()) {  
●     node.getParent().setLeft(son);  
● } else {  
●     node.getParent().setRight(son);  
● }  
● node.setParent(son);  
● return son;  
● }
```

Realiza las operaciones de rotación y splay necesarias para que al nodo que accedamos suba a la cabeza.

Luego estan los demas

- public void insert(T data);
 - Verificando primero si esta vacio el arbol
 - luego crea dos nodos actual y padre con actual nos movemos a la posicion en el que se insertara y luego de encontrar donde se inserta con parent ingresamos el nodo y al final hacemos splay al nodo para que este en la cabeza
- public void insert(E x) {
○ NodeSplay newNode = new NodeSplay(x);
○ if (this.isEmpty()) {
○ this.setHead(newNode);
○ } else {
○ NodeSplay current = (NodeSplay) this.getHead();
○ NodeSplay parent = null;
○
○ while (current != null) {
○ parent = current;
○ if (x.compareTo(current.getData()) < 0) {
○ current = (NodeSplay) current.getLeft();
○ } else {
○ current = (NodeSplay) current.getRight();
○ }
○ }
○
○ newNode.setParent(parent);
○ if (x.compareTo(parent.getData()) < 0) {
○ parent.setLeft(newNode);
○ } else {
○ parent.setRight(newNode);
○ }
○
○ splay(newNode);
○ }
○ }

- public T search(T data);
 - public NodeSplay searchSplay(E x) {
 - NodeSplay current = (NodeSplay) this.getHead();
 - while (current != null) {
 - int cmp = x.compareTo(current.getData());
 - if (cmp == 0) {
 - splay(current);
 - return current;
 - } else if (cmp < 0) {
 - current = (NodeSplay) current.getLeft();
 - } else {
 - current = (NodeSplay) current.getRight();
 - }
 - }
 - return null;
 - }
- public void inOrden();
 - imprimira de manera inorden el arbol ascendente
 - public void inOrden() {
 - if (this.isEmpty())
 - System.out.println("Arbol esta vacio");
 - else
 - inOrden((NodeSplay) this.getHead());
 - System.out.println();
 - }

LECCIONES APRENDIDAS Y CONCLUSIONES

Aprendí a usar el Graph Stream para visualizar los árboles de forma gráfica.

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: 9</p>

<https://graphstream-project.org/download/>

<https://github.com/eluqm/LabEDA-SIS/tree/main/Lab04>