Informe de Laboratorio 06

Tema: Skip List y Splay Tree

Nota

Estudiante	Escuela	Asignatura
Victor Gonzalo Maldonado Vilca, Armando Steven Cuno Cahuari	Escuela Profesional de Ingeniería de Sistemas	Estructura de Datos y Algoritmos (EDA) Semestre: III Código: 1702122

Tarea	Tema	Duración
06	Skip List y Splay Tree	2 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 27/06/2024 - 00:00am	Al 27/06/2024 - 23:59pm

1. Introducción

Las estructuras de datos avanzadas juegan un papel crucial en la optimización de algoritmos y la mejora del rendimiento en aplicaciones que gestionan grandes cantidades de datos. Entre estas estructuras, destacan la Skip List y el Splay Tree, cada una ofreciendo ventajas particulares en términos de eficiencia y consumo de memoria. La Skip List es una estructura de datos probabilística que permite realizar operaciones de búsqueda, inserción y eliminación con un tiempo logarítmico esperado. Por otro lado, el Splay Tree es un tipo de árbol binario de búsqueda autoajustable que mejora el acceso reciente mediante rotaciones.
En este informe, se expone la implementación de ambas estructuras de datos, enfatizando sus objetivos, los métodos empleados y las conclusiones derivadas del análisis de su rendimiento y su aplicabilidad en diversos contextos.

2. Objetivos

- Desarrollar una estructura de datos que permita realizar operaciones de búsqueda, inserción y eliminación en tiempo logarítmico esperado.
- Crear un árbol binario de búsqueda autoajustable que optimice el acceso a los elementos recientemente utilizados mediante rotaciones.
- Evaluar y comparar el rendimiento de la Skip List y el Splay Tree en términos de tiempo de ejecución y uso de memoria.

3. Tarea

Elabore un informe implementando Skip List, Splay Tree

- 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.

4. Entregables

- Informe hecho en latex.
- Archivos java.
- URL: Repositorio GitHub.

5. Equipos, materiales y temas utilizados

- Skip List y Splay Tree
- Java Development Kit
- URL: Repositorio GitHub

6. URL de Repositorio Github

- URL del Repositorio GitHub
- `https://github.com/Victor-Gonzalo-Maldonado-Vilca/EDA_Lab06.git`

7. Desarrollo del trabajo

7.1. Ejecuciones de las clases SkipList.java y SplayTree.java

7.1.1. SkipList

- **CMD:**

```
D:\UNSA\EDA\Lab06\Programa>javac -cp .;"lib/gs-core-2.0.jar;lib/gs-algo-2.0.
jar;lib/gs-ui-swing-2.0.jar" SkipList.java

D:\UNSA\EDA\Lab06\Programa>java -cp .;"lib/gs-core-2.0.jar;lib/gs-algo-2.0.j
ar;lib/gs-ui-swing-2.0.jar" SkipList.java
Skip List después de la inserción:
Level 3: 8
Level 2: 5 8 10
Level 1: 4 5 8 10 12
Level 0: 1 4 5 8 10 12
¿Se encuentra 8 en la Skip List? true
Skip List después de eliminar 5:
Level 3: 8
Level 2: 8 10
Level 1: 4 8 10 12
Level 0: 1 4 8 10 12
```

Figura 1: SkipList - CMD

- **Graph Stream:**

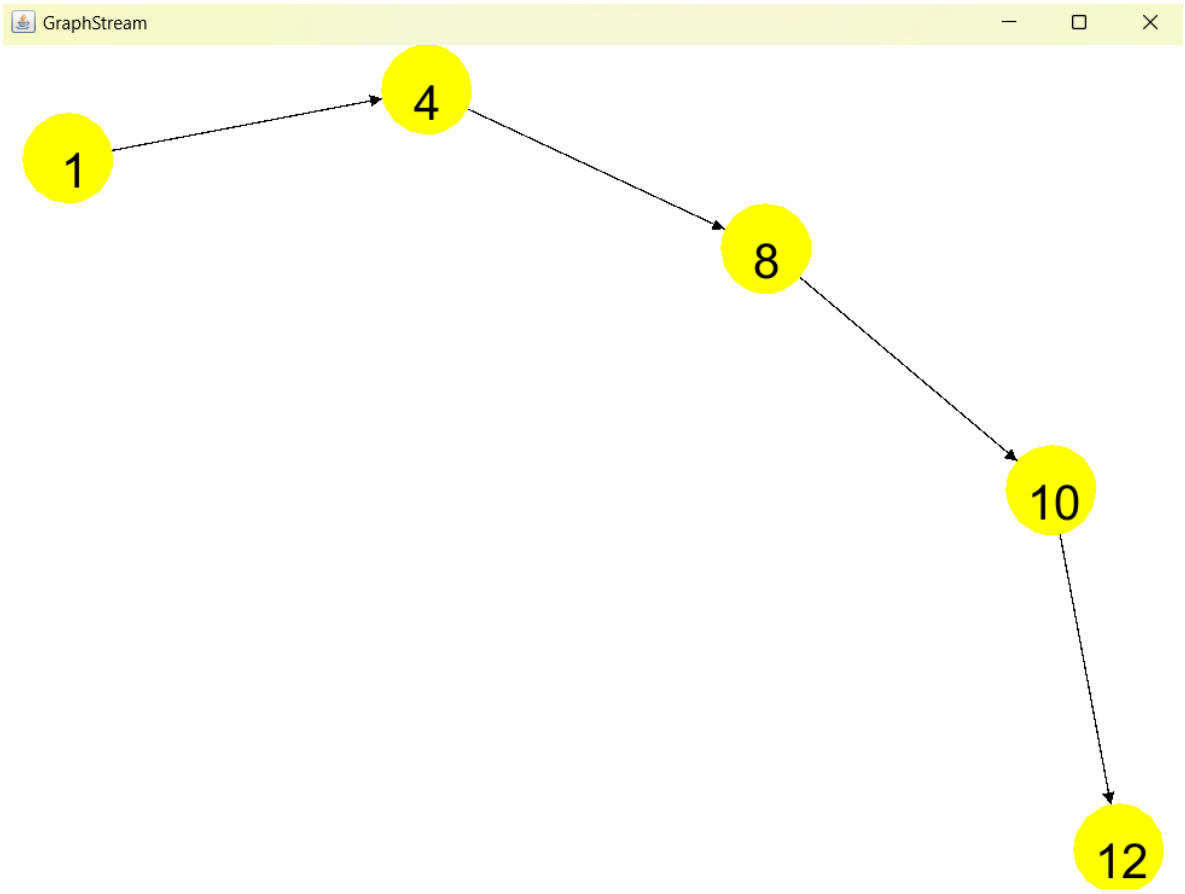

Figura 2: SkipList - Graph Stream

7.1.2. SplayTree

- **CMD:**

```
D:\UNSA\EDA\Lab06\Programa>javac -cp .;"lib/gs-core-2.0.jar;lib/gs-algo-2.0.
jar;lib/gs-ui-swing-2.0.jar" SplayTree.java

D:\UNSA\EDA\Lab06\Programa>java -cp .;"lib/gs-core-2.0.jar;lib/gs-algo-2.0.j
ar;lib/gs-ui-swing-2.0.jar" SplayTree.java
Inorder traversal of the modified tree:
10 20 25 30 40 50
After searching for 20:
10 20 25 30 40 50
After deleting 20:
10 25 30 40 50
```

Figura 3: SplayTree - CMD

- **Graph Stream:**

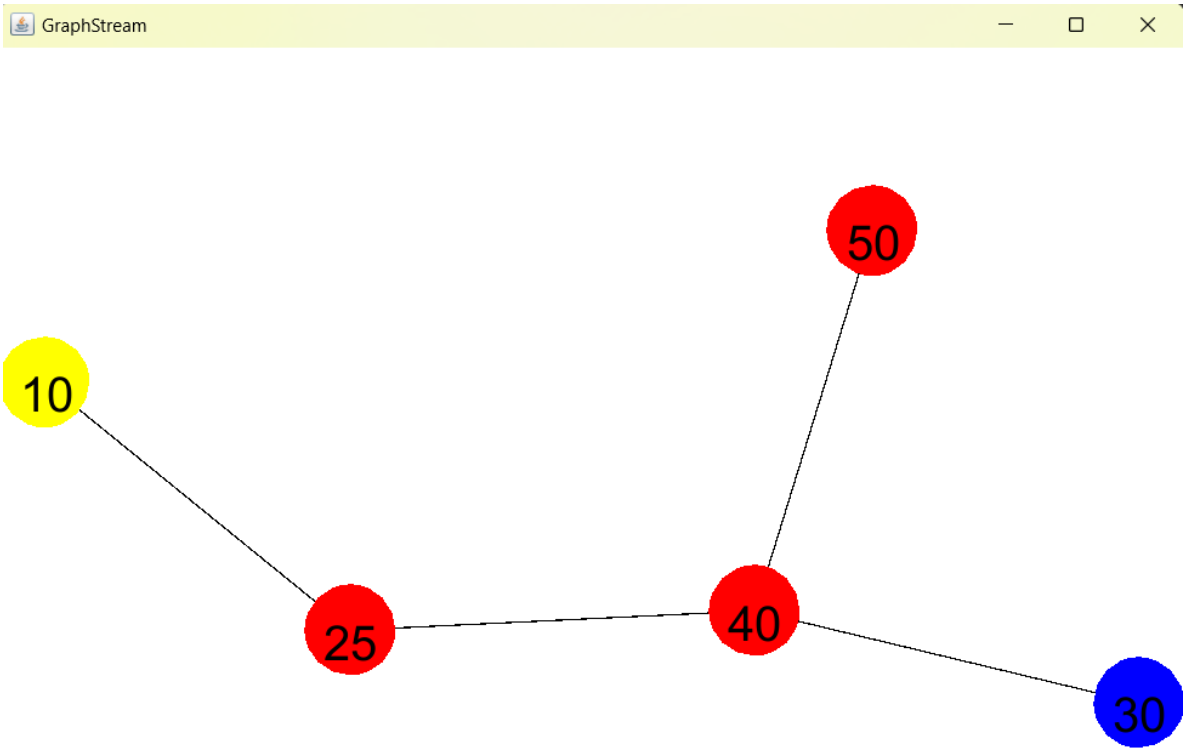

Figura 4: SplayTree - Graph Stream

7.2. SkipList.java

7.2.1. Clase SkipList

- **Descripción:** Esta estructura será capaz de realizar métodos básicos como inserción, eliminación y búsqueda. Para ello, definiremos atributos clave: el nivel máximo permitido en la SkipList, la referencia a "headçomo el nodo inicial que contiene el menor valor posible, el nivel actual de la SkipList, y un valor aleatorio que determinará los niveles de los nuevos valores insertados.
- **Código:**

Listing 1: Clase SkipList

```
public SkipList() {
  this.head = new Node(Integer.MIN_VALUE, MAX_LEVEL);
  this.level = 0;
  this.random = new Random();
  this.graph = new SingleGraph("SkipList");
  this.graph.setStrict(false);
  this.graph.setAutoCreate(true);
}
```

7.2.2. Clase Node

- **Descripción:** Esta será una clase interna privada que estará compuesta por un valor asignado al nodo y un arreglo de referencias hacia nodos de niveles superiores. Además, contará con un constructor que asignará el valor del nodo y el tamaño del arreglo de referencias.
- **Código:**

Listing 2: Clase Node

```
private class Node {
  int value;
  Node[] forward; // Arreglo de referencias hacia adelante en cada nivel

  Node(int value, int level) {
    this.value = value;
    this.forward = new Node[level + 1];
  }
}
```

7.2.3. Método insert

- **Descripción:** Este método se encargará de insertar nodos en la estructura de la SkipList. Primero, crearemos un nuevo nodo con un valor dado y un nivel aleatorio. Luego, actualizaremos las referencias hacia adelante ('forwards') comenzando desde el nivel más alto hasta el nivel 0. Buscaremos el nivel exacto donde colocar el nuevo nodo y actualizaremos las referencias hacia adelante con los nodos correspondientes al insertar el nodo.
- **Código:**

Listing 3: Método insert

```
public void insert(int value) {
  // Crear un nuevo nodo
```

```
    Node newNode = new Node(value, randomLevel());

    // Actualizar los punteros forward en cada nivel
    Node current = head;
    for (int i = level; i >= 0; i--) {
      while (current.forward[i] != null && current.forward[i].value < value) {
        current = current.forward[i];
      }
      if (i <= newNode.forward.length - 1) {
        newNode.forward[i] = current.forward[i];
        current.forward[i] = newNode;
      }
    }

    // Actualizar el nivel maximo de la Skip List si es necesario
    if (newNode.forward.length - 1 > level) {
      level = newNode.forward.length - 1;
    }
    addGraphNode(newNode);
    addGraphEdges();
  }
```

7.2.4. Método search

- **Descripción:** Este método estará encargado de buscar un valor en la estructura de la SkipList. Comenzaremos desde el nivel más alto y descenderemos hasta el nivel 0. En cada nivel, avanzaremos hacia adelante hasta encontrar el siguiente nodo que tenga un valor mayor o igual al buscado. Finalmente, verificaremos si en el nivel 0 se encuentra el valor buscado.
- **Código:**

Listing 4: Método search

```
  public boolean search(int value) {
    Node current = head;
    for (int i = level; i >= 0; i--) {
      while (current.forward[i] != null && current.forward[i].value < value) {
        current = current.forward[i];
      }
    }
    current = current.forward[0];
    return current != null && current.value == value;
  }
```

7.2.5. Método delete

- **Descripción:** Este método estará encargado de eliminar un valor de la estructura de la SkipList. Primero, se buscará el nodo a eliminar y se guardará en una estructura auxiliar llamada üpdate". Luego, se actualizarán las referencias de los nodos. Se actualizarán los nodos hacia adelante ('forwards') de los nodos correspondientes para eliminar las referencias al nodo eliminado. Además, se verificará si es necesario disminuir el nivel de la SkipList después de la eliminación.
- **Código:**

Listing 5: Método delete

```
public void delete(int value) {
  Node[] update = new Node[level + 1];
  Node current = head;
  for (int i = level; i >= 0; i--) {
    while (current.forward[i] != null && current.forward[i].value < value) {
      current = current.forward[i];
    }
    update[i] = current;
  }
  current = current.forward[0];
  if (current != null && current.value == value) {
    for (int i = 0; i <= level; i++) {
      if (update[i].forward[i] != current)
        break;
      update[i].forward[i] = current.forward[i];
    }
    while (level > 0 && head.forward[level] == null) {
      level--;
    }

    removeGraphNode(current);
    addGraphEdges();
  }
}
```

7.2.6. Método randomLevel

- **Descripción:** Este metodo estara encargado de generar de manera aleatorio el nivel de un nuevo nodo.
- **Código:**

Listing 6: Método randomLevel

```
private int randomLevel() {
  int lvl = 0;
  while (lvl < MAX_LEVEL && random.nextDouble() < 0.5) {
    lvl++;
  }
  return lvl;
}
```

7.2.7. Método print

- **Descripción:** Este método estará encargado de imprimir los valores de la SkipList teniendo en cuenta los niveles, recorriendo cada uno de ellos mediante iteraciones.
- **Código:**

Listing 7: Método print

```
public void print() {
  for (int i = level; i >= 0; i--) {
    Node current = head.forward[i];
```

```
        System.out.print("Level " + i + ": ");
        while (current != null) {
          System.out.print(current.value + " ");
          current = current.forward[i];
        }
        System.out.println();
      }
    }
```

7.2.8. Métodos Graph Stream

- **addGraphNode:** Agrega un nodo al gráfico con estilos específicos como círculo amarillo con texto negro y tamaño 60px.

Listing 8: Método addGraphNode

```
    private void addGraphNode(Node node) {
      String nodeId = String.valueOf(node.value);
      org.graphstream.graph.Node graphNode = graph.addNode(nodeId);
      graphNode.setAttribute("ui.label", String.valueOf(node.value));
      String css = "node { size: 60px; shape: circle; fill-color: yellow; text-color:
          black; text-size: 40px; } ";
      this.graph.setAttribute("ui.stylesheet", css);
    }
```

- **removeGraphNode:** Remueve un nodo específico del gráfico si existe.

Listing 9: Método removeGraphNode

```
    private void removeGraphNode(Node node) {
      String nodeId = String.valueOf(node.value);
      if (graph.getNode(nodeId) != null) {
        graph.removeNode(nodeId);
      }
    }
```

- **addGraphEdges:** Agrega los bordes al gráfico, conectando nodos consecutivos en cada nivel del gráfico.

Listing 10: Método addGraphEdges

```
    private void addGraphEdges() {
      graph.clear();
      for (int i = 0; i <= level; i++) {
        Node current = head.forward[i];
        while (current != null) {
          addGraphNode(current);
          String currentId = String.valueOf(current.value);
          String nextId = current.forward[i] == null ? null :
              String.valueOf(current.forward[i].value);
          if (nextId != null) {
            graph.addEdge(currentId + "-" + nextId, currentId, nextId, true);
          }
          current = current.forward[i];
        }
```

```
    }
}
```

7.2.9. Método Main

- **Descripción:** Para probar nuestra estructura de SkipList, realizaremos operaciones básicas como inserción, eliminación y búsqueda. También evaluaremos la impresión de la estructura por niveles para verificar su funcionamiento completo.
- **Código:**

Listing 11: Método Main

```
public static void main(String[] args) {
  // Configurar la propiedad del sistema para el paquete de UI de GraphStream
  System.setProperty("org.graphstream.ui", "swing");

  SkipList skipList = new SkipList();

  skipList.insert(1);
  skipList.insert(4);
  skipList.insert(5);
  skipList.insert(8);
  skipList.insert(10);
  skipList.insert(12);

  System.out.println("Skip List despues de la insercion:");
  skipList.print();

  int searchValue = 8;
  System.out.println("Se encuentra " + searchValue + " en la Skip List? " +
      skipList.search(searchValue));

  int deleteValue = 5;
  skipList.delete(deleteValue);
  System.out.println("Skip List despues de eliminar " + deleteValue + ":");
  skipList.print();

  // Visualizar el grafico
  skipList.graph.display();
}
```

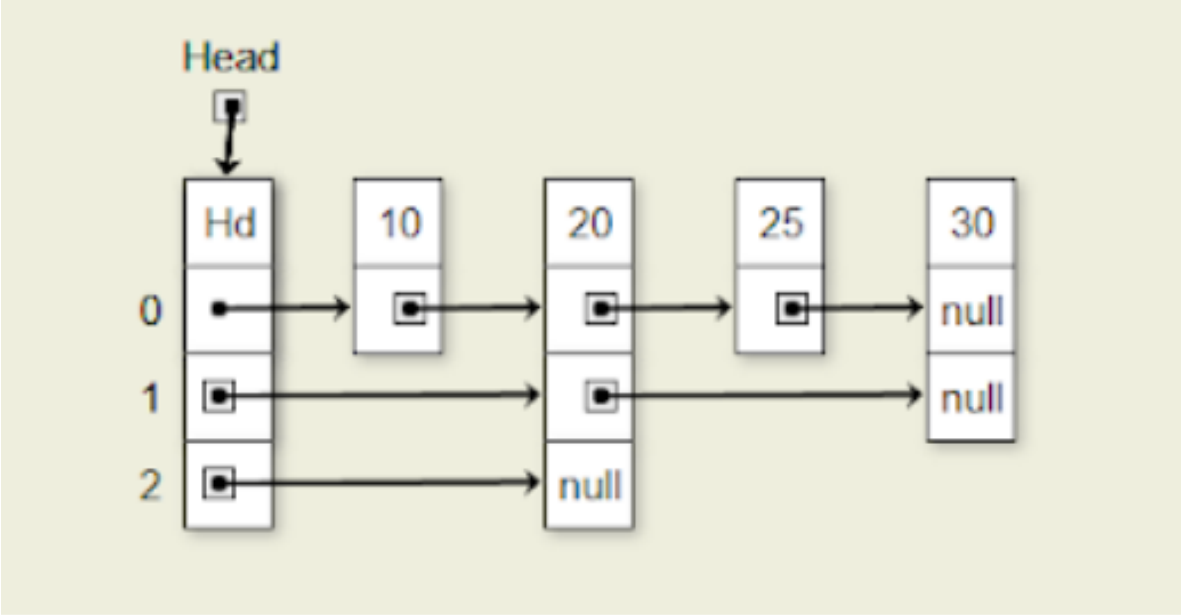

Figura 5: SkipList

7.3. SplayTree.java

7.3.1. Clase Node

- **Descripción:** Definiremos la clase 'Node' como una clave privada interna diseñada para estructurar un 'SplayTree'. Cada nodo contendrá un valor, junto con referencias a sus dos hijos, izquierdo y derecho. En el constructor, inicializaremos las referencias de los hijos como 'null', y el parámetro del constructor se asignará al valor del nodo.
- **Código:**

Listing 12: Clase Node

```
private class Node<T> {
  T key;
  Node<T> left, right;

  Node(T key) {
    this.key = key;
    left = right = null;
  }
}
```

7.3.2. Método rightRotate

- **Descripción:** El método realizará una rotación a la derecha con respecto a un nodo 'x'. Primero, asignaremos a 'y' como el hijo izquierdo de 'x'. Luego, el hijo izquierdo de 'x' será el hijo derecho de 'y', y el hijo derecho de 'y' será ahora 'x'. Finalmente, retornaremos 'y' como la nueva raíz del subárbol.
- **Código:**

Listing 13: Método rightRotate

```
private Node<T> rightRotate(Node<T> x) {
  Node<T> y = x.left;
  x.left = y.right;
  y.right = x;
  return y;
}
```

7.3.3. Método leftRotate

- **Descripción:** El metodo realizara una rotacion a la izquierda con respecto a un nodo 'x'. Primero, asignaremos a 'y' como el hijo derecho de 'x'. Luego, el hijo derecho de 'x' sera el hijo izquierdo de 'y', y el hijo izquierdo de 'y' sera ahora 'x'. Finalmente, retornaremos 'y' como la nueva raiz del subarbol.
- **Código:**

Listing 14: Método leftRotate

```
private Node<T> leftRotate(Node<T> x) {
  Node<T> y = x.right;
  x.right = y.left;
  y.left = x;
  return y;
}
```

7.3.4. Método splay

- **Descripción:** El método realizará un movimiento del nodo con una clave especificada en el árbol. Primero, revisará si el árbol está vacío o si la clave del nodo coincide con la clave buscada; en ese caso, retornará root". Segundo, si la clave es menor que la clave del nodo raíz, ocurrirán varios casos: primero, si el hijo izquierdo de root.[es] nulo, retornará root"; luego, si la clave es menor que la clave del hijo izquierdo de root", realizará una rotación a la derecha de root". Después, si la clave es mayor que la clave del hijo izquierdo de root", realizará dos rotaciones: una a la izquierda y luego otra a la derecha con respecto a root".
  Tercero, si la clave es mayor que el valor de la clave de root", entonces ocurrirán los siguientes casos: primero, si el hijo derecho de root.[es]tá vacío, retornará root". Luego, si la clave es menor que la clave del hijo derecho de root", realizará dos rotaciones: primero una a la derecha y luego otra a la izquierda con respecto a root". Después, si la clave es mayor que la clave del hijo derecho de root", realizará solo una rotación a la izquierda con respecto a root".
- **Código:**

Listing 15: Método leftRotate

```
private Node<T> splay(Node<T> root, T key) {
  if (root == null || root.key.equals(key))
    return root;

  if (root.key.compareTo(key) > 0) {
    if (root.left == null) return root;

    if (root.left.key.compareTo(key) > 0) {
      root.left.left = splay(root.left.left, key);
```

```
        root = rightRotate(root);
      } else if (root.left.key.compareTo(key) < 0) {
        root.left.right = splay(root.left.right, key);
        if (root.left.right != null)
          root.left = leftRotate(root.left);
      }

      return (root.left == null) ? root : rightRotate(root);
    } else {
      if (root.right == null) return root;

      if (root.right.key.compareTo(key) > 0) {
        root.right.left = splay(root.right.left, key);
        if (root.right.left != null)
          root.right = rightRotate(root.right);
      } else if (root.right.key.compareTo(key) < 0) {
        root.right.right = splay(root.right.right, key);
        root = leftRotate(root);
      }
      return (root.right == null) ? root : leftRotate(root);
    }
  }
```

7.3.5. Método insert

- **Descripción:** El método estará encargado de insertar nodos en la estructura. Para ello:

  1. Primero, revisará si el nodo root.está vacío; en ese caso, creará un nuevo nodo con la clave correspondiente y lo asignará a root".
  2. Segundo, realizará un splay con respecto a rootz la clave dada.
  3. Tercero, comparará la clave con la clave del nodo root".
  4. Cuarto, si la clave es idéntica a la clave del nodo root", entonces no realizará ninguna acción.
  5. Quinto, creará un nuevo nodo llamado 'newNode' con la clave dada.
  6. Sexto, si la clave es menor que la clave del nodo root", entonces colocará 'newNode' como hijo izquierdo de root".
  7. Séptimo, si la clave es mayor que la clave del nodo root", entonces colocará 'newNode' como hijo derecho de root".
  8. Finalmente, asignará 'newNode' como el nuevo root".

- **Código:**

Listing 16: Método insert

```
  public void insert(T key) {
    if (root == null) {
      root = new Node<>(key);
      return;
    }
    root = splay(root, key);
    int cmp = key.compareTo(root.key);
    if (cmp == 0) return;
    Node<T> newNode = new Node<>(key);
    if (cmp < 0) {
```

```
        newNode.right = root;
        newNode.left = root.left;
        root.left = null;
      } else {
        newNode.left = root;
        newNode.right = root.right;
        root.right = null;
      }
      root = newNode;
    }
```

7.3.6. Método delete

- **Descripción:** El método delete elimina un nodo con una clave específica en un árbol Splay. Si el nodo raíz es nulo, no hace nada. Luego, utiliza la operación splay para posicionar el nodo con la clave buscada en la raíz. Si encuentra el nodo con la clave, lo elimina del árbol. Si el nodo tiene solo un hijo, conecta ese hijo con el nodo padre; si tiene dos hijos, reestructura el árbol manteniendo la propiedad de Splay.
- **Código:**

Listing 17: Método delete

```
    public void delete(T key) {
      if (root == null) return;
      root = splay(root, key);
      if (!root.key.equals(key)) return;
      if (root.left == null) {
        root = root.right;
      } else {
        Node<T> temp = root.right;
        root = root.left;
        splay(root, key);
        root.right = temp;
      }
    }
```

7.3.7. Método search

- **Descripción:** Este método estará encargado de la búsqueda de ciertas claves. Primero, realizará un splay con respecto a un nodo rootz una clave. Luego, devolverá ‘true‘ si el nodo root”no está vacío, y retornará ‘false‘ en caso contrario.
- **Código:**

Listing 18: Método search

```
    public boolean search(T key) {
      root = splay(root, key);
      return root != null && root.key.equals(key);
    }
```

7.3.8. Método printInOrder

- **Descripción:** Este método imprimirá la estructura en orden. Si el nodo está vacío, no hará nada. Luego, hará una llamada recursiva al hijo izquierdo del nodo, imprimirá el nodo, y finalmente hará una llamada recursiva al hijo derecho del nodo.
- **Código:**

Listing 19: Método printInOrder

```
public void printInOrder(Node<T> node) {
  if (node == null) return;
  printInOrder(node.left);
  System.out.print(node.key + " ");
  printInOrder(node.right);
}
```

7.3.9. Método printTree

- **Descripción:** Este metodo llamara al metodo printInOrder con respecto a root.
- **Código:**

Listing 20: Método printTree

```
public void printTree() {
  printInOrder(root);
  System.out.println();
}
```

7.3.10. Métodos Graph Stream

- **initGraph:** El método initGraph() inicializa un gráfico en GraphStream para visualizar un árbol Splay, estableciendo estilos para los nodos.

Listing 21: Método initGraph

```
private void initGraph() {
  graph = new SingleGraph("SplayTree");
  graph.setAttribute("ui.stylesheet",
  "node { size: 30px; fill-color: black; text-color: white; text-size: 15px; }");
}
```

- **visualizeTree:** El método visualizeTree() visualiza el árbol Splay utilizando GraphStream, inicializando el gráfico y llamando a visualizeSubtree(root) para agregar nodos y aristas.

Listing 22: Método visualizeTree

```
public void visualizeTree() {
  initGraph(); // Initialize the graph
  visualizeSubtree(root);
  Viewer viewer = graph.display();
  viewer.setCloseFramePolicy(Viewer.CloseFramePolicy.HIDE_ONLY);
}
```

- **visualizeSubtree:** El método visualizeSubtree(Node¡T¿node) es recursivo y agrega nodos y aristas al gráfico para representar el subárbol con raíz en el nodo dado.

Listing 23: Método visualizeSubtree

```
private void visualizeSubtree(Node<T> node) {
  if (node != null) {
    String nodeId = node.key.toString(); // Get node value as String

    // Add node to graph only if it doesn't exist
    if (graph.getNode(nodeId) == null) {
      org.graphstream.graph.Node graphNode = graph.addNode(nodeId);
      graphNode.setAttribute("ui.label", nodeId); // Set label with node value
      String css = "size: 60px; shape: circle; fill-color: yellow; text-color:
          black; text-size: 40px;";
      graphNode.setAttribute("ui.style", css);
    }

    // Add edges to children
    if (node.left != null) {
      String leftId = node.left.key.toString();
      if (graph.getNode(leftId) == null) {
        org.graphstream.graph.Node leftNode = graph.addNode(leftId);
        leftNode.setAttribute("ui.label", leftId);
        String css = "size: 60px; shape: circle; fill-color: blue; text-color:
            black; text-size: 40px;";
        leftNode.setAttribute("ui.style", css);
      }
      graph.addEdge(nodeId + "-" + leftId, nodeId, leftId);
      visualizeSubtree(node.left);
    }
    if (node.right != null) {
      String rightId = node.right.key.toString();
      if (graph.getNode(rightId) == null) {
        org.graphstream.graph.Node rightNode = graph.addNode(rightId);
        rightNode.setAttribute("ui.label", rightId);
        String css = "size: 60px; shape: circle; fill-color: red; text-color:
            black; text-size: 40px;";
        rightNode.setAttribute("ui.style", css);
      }
      graph.addEdge(nodeId + "-" + rightId, nodeId, rightId);
      visualizeSubtree(node.right);
    }
  }
}
```

7.3.11. Método Main

- **Descripción:** Una pequeña prueba haciendo uso de esta estructura asimismo de las operaciones básicas que brinda.
- **Código:**

Listing 24: Método Main

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

```
        // Example usage of the SplayTree from its main class

        // Create a SplayTree of integers
        SplayTree<Integer> splayTree = new SplayTree<>();
        System.setProperty("org.graphstream.ui", "swing");

        splayTree.insert(10);
        splayTree.insert(20);
        splayTree.insert(30);
        splayTree.insert(40);
        splayTree.insert(50);
        splayTree.insert(25);

        System.out.println("Inorder traversal of the modified tree:");
        splayTree.printTree();

        splayTree.search(20);
        System.out.println("After searching for 20:");
        splayTree.printTree();

        splayTree.delete(20);
        System.out.println("After deleting 20:");
        splayTree.printTree();

        // Visualize the SplayTree
        splayTree.visualizeTree();
    }
```

8. Conclusiones

- La implementación de la Skip List demuestra que esta estructura es eficiente para operaciones de búsqueda, inserción y eliminación, especialmente en aplicaciones donde la simplicidad y el rendimiento probabilístico son preferidos sobre el rendimiento garantizado.
- La Skip List tiende a ser más simple de implementar y ofrece un buen rendimiento promedio.
- El Splay Tree es beneficioso para escenarios en los que los datos accedidos recientemente necesitan ser recuperados rápidamente.

8.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 2: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		20	

9. Referencias

- https://www.w3schools.com/java/
- https://www.eclipse.org/downloads/packages/release/2022-03/r/eclipse-ide-enterprise-java-and-web
- https://docs.oracle.com/javase/7/docs/api/java/util/List.html
- https://docs.oracle.com/javase/tutorial/java/generics/types.html
- https://www.baeldung.com/java-avl-trees