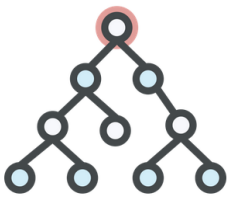
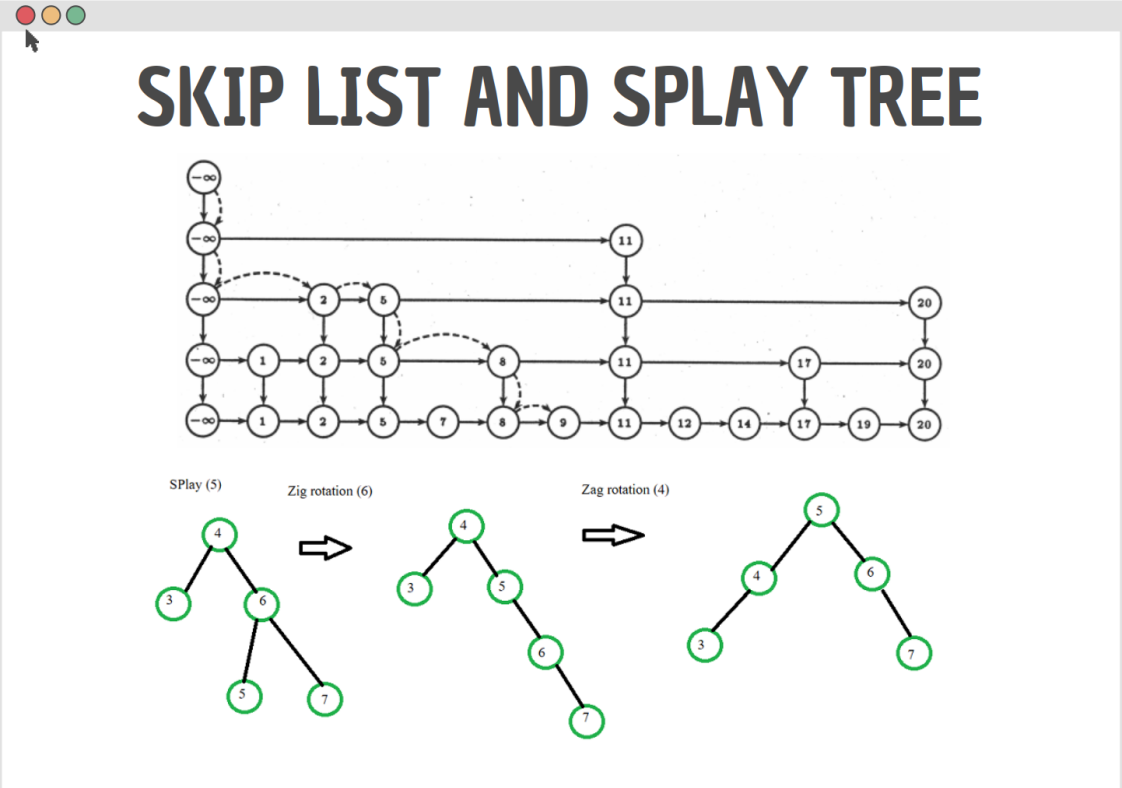


ESTRUCTURAS DE DATOS Y ALGORITMOS con Java



Profesor(a):
Edson Luque Mamani

Estudiante:
Jorge Luis Mamani Huarsaya
Victor Narciso Mamani Anahua

27 de junio, 2024

Lab06 - SkipList:

La clase SkipList

La clase SkipListT en Java es una implementación de una estructura de datos avanzada conocida como SkipList, diseñada para proporcionar operaciones eficientes de búsqueda, inserción y eliminación. Utiliza múltiples niveles de punteros, permitiendo un acceso más rápido en comparación con las listas enlazadas simples. Esta implementación genérica admite cualquier tipo que implemente la interfaz Comparable, asegurando que los elementos puedan ser ordenados y comparados correctamente. A continuación, se describen los componentes y métodos principales de esta implementación.

La clase SkipListT implementa una SkipList genérica que admite elementos que implementan la interfaz Comparable. La clase contiene una constante MAX_LEVEL que define el número máximo de niveles en la lista, un nodo cabeza head de tipo SkipListNodeT con el nivel máximo, y un objeto Random para generar niveles aleatorios. Además, se mantiene un entero level que indica el nivel más alto actualmente en uso. La estructura de datos permite que cada nodo apunte a varios otros nodos a diferentes niveles, proporcionando así una manera rápida de atravesar la lista.

La clase SkipListNodeT es una clase interna estática que representa un nodo en la SkipList. Cada nodo contiene un valor de tipo T y un array de punteros forward que apunta a otros nodos en niveles distintos. El constructor de SkipListNode inicializa estos punteros según el nivel proporcionado, asegurando que cada nodo pueda tener múltiples referencias hacia adelante, lo que permite saltar varios nodos en una sola operación, mejorando así la eficiencia de las búsquedas.

El método contains(T value) verifica si un valor está presente en la lista. Recorre la lista desde el nivel superior hacia abajo, avanzando en cada nivel hasta encontrar el nodo que contiene el valor o determinar que no está presente. Esto permite realizar búsquedas de manera rápida, aprovechando los niveles superiores para saltar grandes segmentos de la lista y reducir el número total de comparaciones necesarias.

El método add(T value) agrega un nuevo valor a la SkipList. Primero, encuentra las posiciones donde se deben actualizar los punteros para mantener la integridad de la lista. Luego, si el valor no está presente, genera un nuevo nivel para el nodo, actualiza los punteros y, si es necesario, incrementa el nivel de la lista. Esta técnica asegura que la lista se mantenga balanceada y que las operaciones de inserción sean eficientes.

El método remove(T value) elimina un valor de la SkipList. Similar a add, primero encuentra las posiciones de los punteros que deben actualizarse. Si encuentra el nodo con el valor, actualiza los punteros para omitir este nodo y ajusta el nivel de la lista si es necesario. Esto permite eliminar elementos de manera eficiente, ajustando dinámicamente los niveles de la lista para mantener el equilibrio y el rendimiento.

El método randomLevel() genera un nivel aleatorio para un nodo nuevo. Incrementa el nivel hasta MAX_LEVEL con una probabilidad del 50% en cada incremento. Esta función es crucial para mantener la naturaleza probabilística de la SkipList, asegurando que la distribución de niveles se mantenga equilibrada a largo plazo y proporcionando la base para la eficiencia de las operaciones de búsqueda e inserción.

La clase implementa la interfaz IterableT y proporciona un iterador que recorre la SkipList de principio a fin. Esto permite utilizar la SkipList en bucles for-each y otras construcciones que requieren un iterador, proporcionando una manera sencilla y natural de recorrer los elementos de la lista.

El método printList() imprime la estructura de la SkipList mostrando los nodos en cada nivel, útil para visualizar el contenido y la estructura de la lista. Este método es particularmente útil para depuración y para entender cómo se distribuyen los nodos en diferentes niveles, proporcionando una representación visual clara del estado de la SkipList.

```java
import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.Random;

public class SkipList<T extends Comparable<? super T>> implements Iterable<T> {
  private static final int MAX_LEVEL = 4;
  private final SkipListNode<T> head = new SkipListNode<>(null, MAX_LEVEL);
  private final Random random = new Random();
  private int level = 0;

  private static class SkipListNode<T> {
    final T value;
    final SkipListNode<T>[] forward;

    @SuppressWarnings("unchecked")
   SkipListNode(T value, int level) {
      this.value = value;
      this.forward = new SkipListNode[level + 1];
    }
  }

  public boolean contains(T value) {
   SkipListNode<T> current = head;
    for (int i = level; i >= 0; i--)
      while (current.forward[i] != null && current.forward[i].value.compareTo(value)
      ↪  < 0)
        current = current.forward[i];

    current = current.forward[0];
    return current != null && current.value.compareTo(value) == 0;
  }

  public void add(T value) {
   SkipListNode<T>[] update = new SkipListNode[MAX_LEVEL + 1];
   SkipListNode<T> current = head;

    for (int i = level; i >= 0; i--) {
      while (current.forward[i] != null && current.forward[i].value.compareTo(value)
      ↪  < 0)
        current = current.forward[i];
      update[i] = current;
    }

    current = current.forward[0];

    if (current == null || !current.value.equals(value)) {
      int newLevel = randomLevel();
      if (newLevel > level) {
        for (int i = level + 1; i <= newLevel; i++)
          update[i] = head;
        level = newLevel;
      }

     SkipListNode<T> newNode = new SkipListNode<>(value, newLevel);
```

```
53        for (int i = 0; i <= newLevel; i++) {
54          newNode.forward[i] = update[i].forward[i];
55          update[i].forward[i] = newNode;
56        }
57      }
58    }
59
60    public boolean remove(T value) {
61     SkipListNode<T>[] update = new SkipListNode[MAX_LEVEL + 1];
62     SkipListNode<T> current = head;
63      for (int i = level; i >= 0; i--) {
64        while (current.forward[i] != null && current.forward[i].value.compareTo(value)
          ↪  < 0)
65          current = current.forward[i];
66        update[i] = current;
67      }
68      current = current.forward[0];
69      if (current != null && current.value.equals(value)) {
70        for (int i = 0; i <= level; i++) {
71          if (update[i].forward[i] != current)
72            break;
73          update[i].forward[i] = current.forward[i];
74        }
75        while (level > 0 && head.forward[level] == null)
76          level--;
77        return true;
78      }
79      return false;
80    }
81
82    private int randomLevel() {
83      int lvl = 0;
84      while (lvl < MAX_LEVEL && random.nextInt(2) == 0)
85        lvl++;
86      return lvl;
87    }
88
89    @Override
90    public Iterator<T> iterator() {
91      return new Iterator<T>() {
92        private SkipListNode<T> current = head.forward[0];
93
94        @Override
95        public boolean hasNext() {
96          return current != null;
97        }
98
99        @Override
100       public T next() {
101         if (current == null)
102           throw new NoSuchElementException();
103         T value = current.value;
104         current = current.forward[0];
105         return value;
```

```
      }
    };
  }

  public void printList() {
    System.out.println("SkipList:");

    for (int i = MAX_LEVEL; i >= 0; i--) {
      SkipListNode current = head.forward[i];
      System.out.print("Level " + i + ": ");

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