Geeks México

BLOG DE PROGRAMACIÓN EN ESPAÑOL SOBRE JAVA, FRAMEWORKS, BASES DE DATOS, CÓMPUTO EN LA NUBE, ETC. EN ESPAÑOL Y EN INGLÉS.

Árboles binarios en Java

HACE 2 SEMANAS DEJA UN COMENTARIO

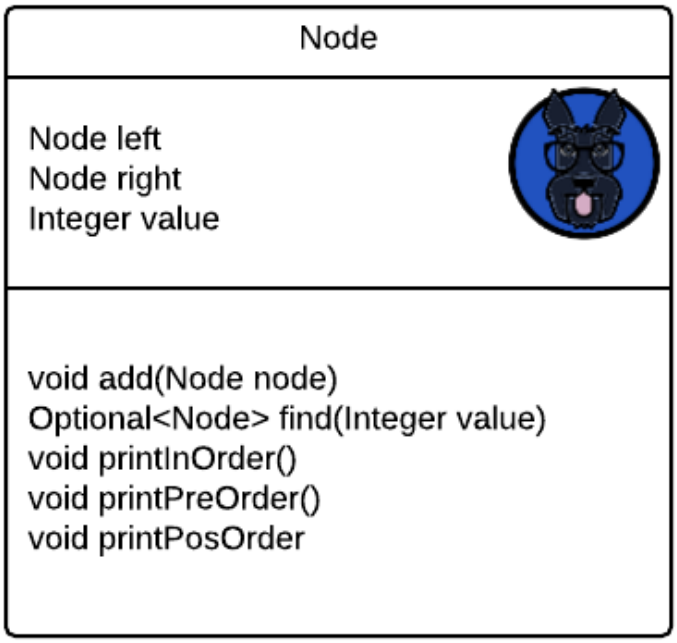
Uno de los temas sobre desarrollo de software que todos los desarrolladores de software deberían conocer y que por lo mismo es cuestionado en una gran parte de las entrevistas es el manejo de arboles binarios, en este post entraremos en detalle en este tema y lo analizaremos paso a paso.

Introducción

En este ejemplo se mostrará como desarrollar un árbol binario balanceado en Java, esto significa que los datos serán ordenados en tiempo de inserción de acuerdo a lo siguiente:

- Si el valor a insertar es menor al valor del nodo se insertará a la izquierda
- Si el valor a insertar es mayor al valor del nodo se insertará a la derecha

La clase a desarrollar será de acuerdo al siguiente diagrama:



Analizando la clase

El siguiente paso será analizar la clase a desarrollar, para esto expliquemos a detalle cada uno de los componentes:

- Atributos
  - value: Cada nodo contendrá un valor almacenado
  - left : Cada nodo tendrá una rama izquierda
  - right : Cada nodo tendrá una rama derecha
- Métodos
  - add : Permite agregar un nuevo valor al árbol binario
  - find : Permite buscar un valor en el árbol binario
  - printInOrder : Permite imprimir el árbol en modo In order
  - printPreOrder : Permite imprimir el árbol en modo Pre Order
  - printPosOrder : Permite imprimir el árbol en modo Pos Order

Creando el árbol

Para crear el árbol solo debemos crear una clase llamada Node (Nodo), veamos el código necesario para definir los atributos a utilizar:

```
1  public class Node {
2      private Integer value;
3      private Node left;
4      private Node right;
5
6      public Node(Integer value) {
7          this.value = value;
8      }
9  //... Getters y setters
10 }
```

Agregar un valor nuevo

Para agregar un valor nuevo al árbol la lógica será la siguiente:

1. Si el valor a agregar es menor al del nodo actual se insertará del lado izquierdo
2. Si el valor a agregar es mayor al del nodo actual se insertará del lado derecho

Recordemos que cada nodo puede tener solo 2 nodos hijos, por lo tanto si el nodo actual contiene un 10 y el valor a insertar es un 7 este se insertará del lado izquierdo, si ya hay un valor insertado del lado izquierdo se ejecutará la misma lógica con el valor insertado hasta llegar a un nodo izquierdo o derecho nulo, veamos el código:

```
1  public class Node {
2  ...// El código anterior
3  public void add(Integer value) {
4      if (value < this.value) {
5          if (left != null) {
6              left.add(value);
7          } else {
8              left = new Node(value);
9          }
10     } else {
11         if (right != null) {
```

```
            right.add(value);
        } else {
            right = new Node(value);
        }
    }
}
}
```

Como se puede observar en el código se utiliza recursividad para agregar los nodos en caso de que ya existan en el árbol.

Buscar un valor

La lógica para buscar un valor en el árbol es muy similar que para insertarlo, en el código que se mostrará se utilizará la clase Optional agregada en Java 8, veamos el código:

```
import java.util.Optional;
public class Node {
...// El código anterior
public Optional<Node> find(Integer value)
    if (value == this.value) {
        return Optional.of(this);
    } else if (value < this.value) {
        if (this.left != null) {
            return this.left.find(value);
        } else {
            return Optional.empty();
        }
    } else {
        if (this.right != null) {
            return this.right.find(value);
        } else {
            return Optional.empty();
        }
    }
}
}
```

Como se puede ver es necesario importar la clase Optional del paquete java.util.

Imprimir los valores del árbol binario

Para imprimir los valores es necesario iterar el árbol utilizando recursividad y existen 3 formas de hacerlo:

1. In order: Imprime el valor del nodo izquierdo, después el del nodo contenedor y al final el del nodo derecho
2. Pre order: Imprime el valor del nodo contenedor, después el del nodo izquierdo y al final el del nodo derecho
3. Pos order: Imprime el valor del nodo izquierdo, después el del nodo derecho y al final el del nodo contenedor

Veámoslo en código:

```
public class Node{
...// El código anterior
        public void printInOrder(){
        if(left!=null){
            left.printInOrder();
        }
        System.out.println(value);
        if(right!=null){
            right.printInOrder();
        }
    }

    public void printPreOrder(){
        System.out.println(value);
        if(left!=null){
            left.printPreOrder();
        }
        if(right!=null){
            right.printPreOrder();
        }
    }

    public void printPosOrder(){
        if(left!=null){
            left.printPreOrder();
        }
        if(right!=null){
            right.printPreOrder();
        }
        System.out.println(value);
    }
}
```

Todo el código junto

Si unimos el código anterior queda del siguiente modo:

```
1  import java.util.Optional;
2
3  /**
4   * @author raidentrance
5   *
6   */
7  public class Node {
8      private Integer value;
9      private Node left;
10     private Node right;
11
12     public Node(Integer value) {
13         this.value = value;
14     }
15
16     public Integer getValue() {
17         return value;
18     }
19
20     public void setValue(Integer value) {
21         this.value = value;
22     }
23
24     public Node getLeft() {
25         return left;
26     }
27
28     public void setLeft(Node left) {
29         this.left = left;
30     }
31
32     public Node getRight() {
33         return right;
34     }
35
36     public void setRight(Node right) {
37         this.right = right;
38     }
39
40     public void add(Integer value) {
41         if (value < this.value) {
42             if (left != null) {
43                 left.add(value);
44             } else {
45                 left = new Node(value);
46             }
47         } else {
48             if (right != null) {
49                 right.add(value);
50             } else {
51                 right = new Node(value);
52             }
53         }
54     }
55
56     public Optional<Node> find(Integer va
57         if (value == this.value) {
58             return Optional.of(this);
59         } else if (value < this.value) {
```

```
60              if (this.left != null) {
61                  return this.left.find(va
62              } else {
63                  return Optional.empty();
64              }
65          } else {
66              if (this.right != null) {
67                  return this.right.find(va
68              } else {
69                  return Optional.empty();
70              }
71          }
72      }
73
74      public void printInOrder() {
75          if (left != null) {
76              left.printInOrder();
77          }
78          System.out.println(value);
79          if (right != null) {
80              right.printInOrder();
81          }
82      }
83
84      public void printPreOrder() {
85          System.out.println(value);
86          if (left != null) {
87              left.printPreOrder();
88          }
89          if (right != null) {
90              right.printPreOrder();
91          }
92      }
93
94      public void printPosOrder() {
95          if (left != null) {
96              left.printPreOrder();
97          }
98          if (right != null) {
99              right.printPreOrder();
100         }
101         System.out.println(value);
102     }
103
104     @Override
105     public String toString() {
106         return "Node [value=" + value + '
107     }
108
109 }
```

Probando el árbol

Para probar el árbol que creamos, utilizaremos una clase nueva llamada TreeTest, veamos el código:

```
1  import java.util.Optional;
2
```

```
/**
 * @author raidentrance
 *
 */
public class TreeTest {
    public static void main(String[] args)
        Node root = new Node(10);
        root.add(5);
        root.add(15);
        root.add(8);

        Optional<Node> result = root.find(
        if (result.isPresent()) {
            System.out.println(result.get(
        } else {
            System.out.println("Value not
        }

        result = root.find(8);
        if (result.isPresent()) {
            System.out.println(result.get(
        } else {
            System.out.println("Value not
        }
        System.out.println("Print in order
        root.printInOrder();
        System.out.println("Print pos orde
        root.printPosOrder();
        System.out.println("Print pre orde
        root.printPreOrder();
    }
}
```

Producirá la siguiente salida:

```
Value not found
Node [value=8, left=null, right=null]
Print in order
5
8
10
15
Print pos order
5
8
15
10
Print pre order
10
5
8
15
```

Si te gusta el contenido compártelo y no olvides seguirnos en nuestras redes sociales https://twitter.com/geeks_mx (https://twitter.com/geeks_mx) y

https://www.facebook.com/geeksJavaMexico/ (https://www.facebook.com/geeksJavaMexico/).

Autor: Alejandro Agapito Bautista

Twitter: @raidentrance

Contacto:raidentrance@gmail.com