ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 4
Tablas de hash y Arboles binarios

Santiago Ochoa Castaño
Universidad Eafit
Medellín, Colombia
sochoac1@eafit.edu.co

Miguel Ángel Zapata Jiménez
Universidad Eafit
Medellín, Colombia
mazapataj@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1.

A modo de ubicar cada abeja en un lugar para determinar si existe o no riesgo de colisión con otra, se implementó la estructura de datos conocida como Octree. Esta estructura consiste en un árbol en donde cada uno de sus nodos se puede dividir en ocho nuevos nodos. La idea principal, se basa en partir un espacio tridimensional, dividiéndolo recursivamente en ocho cuadrantes. De esta manera, es posible asignar a cada abeja un cuadrante específico del árbol, representado por un arreglo dinámico de 8 espacios, y se guardara dentro de una lista enlazada. Cada vez que haya más de una abeja en un mismo cuadrante, este se dividirá en nuevos Octrees repitiendo el proceso hasta que quede una sola abeja por cuadrante. La manera de decidir el sector que le corresponde a cada abeja se hará mediante un método hash. Una vez en que todas las abejas se ubiquen, primero pasará a preguntarse si las diagonales de cada cuadrante son menores a 100 metros. En caso de que si, se imprimirán las abejas con riesgo de colisión. Si no, se divide cada sector formando nuevos Octrees para posicionar las abejas, repitiendo el proceso de manera recursiva.

La complejidad que tiene el algoritmo en caso de que no se presenten colisiones y se posicione una abeja por cuadrante dentro del Octree, se determina por la ecuación T(n) = 8*T(n/8) + C. Esto se debe a que cada nodo se partirá en 8 divisiones y así de forma sucesiva.

3.4 Calcular la complejidad del ejercicio 2.1

Complejidad Asintótica

```
import java.util.ArrayList;
public class Point2 {
    public Node root;
    public void buildingTree (int [] preOrder) { Complejidad de este método: T(n) = T(n(n/2)) + C
    //Si lo pasamos a notación O grande seria O(nLog(n))
        for(int i = 0; i < preOrder.length; i++){ (C1)n
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
            insertar(preOrder[i]); n(n/2)C2
        }
    }

    public void insertar(int n) { Complejidad de este método: T(n) = T(n/2) + C
    //Si lo pasamos a notación O grande seria O(Log(n))
        if (root == null) { T(n) = C3
            root = new Node(n); T(n) = C4
        } else { T(n) = C5
            insertarAux(root, n); T(n) = (n/2) +C
        }
    }

    private void insertarAux(Node node, int n) { Complejidad de este método:  T(n) =  T(n/2) + C
    //Si lo pasamos a notación O grande será O(Log(n))
        if (node == null) { T(n) = C7
            node = new Node(n); T(n) = C8
        }
        if (n > node.data) { T(n) = C9
            if (node.right == null) { T(n) = C10
                node.right = new Node(n); T(n) = C11
            }else{ T(n) = C12
                insertarAux(node.right, n); T(n) = (n/2) + C13
            }
        }

        if (n < node.data) { T(n) = C14
            if (node.left == null) { T(n) = C15
                node.left = new Node(n); T(n) = C16
            }else{ T(n) = C17
                insertarAux(node.left, n); T(n) = (n/2) + C18
            }
        }
    }

    public void preOrder(Node node) { Complejidad de este método: T(n) = 2*T(n/2) + C
    //Si lo pasamos a notación O grande obtendríamos que sería O(Log(n))
        if(node == null){ T(n) = C19
            return; T(n) = C20
        }else{
            System.out.println(node.data); T(n) = C21
            posOrder(node.left); T(n) = T(n/2)
            posOrder(node.right); T(n) = T(n/2)
        }
    }

    public void posOrder(Node node) { Complejidad de este método: T(n) = 2*T(n/2) + C
    //Si lo pasamos a notación O grande quedaría O(Log(n))
       if(node == null){ T(n) = C22
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

```
            return; T(n) = C23
        }else{
            posOrder(node.left); T(n) = T(n/2)
            posOrder(node.right); T(n) = T(n/2)
            System.out.println(node.data);  T(n) = C24
        }
    }

    public void exercise21 (int [] input) {
        buildingTree(input);
        System.out.println("PosOrder");
        posOrder(root);
        System.out.println();
    }

    public static void main (String [] args) {
        int [] test = {50,30,24,5,28,45,98,52,60};
        Point2 testing = new Point2();
        testing.exercise21(test);
    }
}
```

3.5 Explique con sus palabras las variables presentes en el cálculo de complejidad del punto 2.1

Cuando observamos el código vemos que cada método tiene como parámetro principal el nodo raíz, estando esté ligado a todos los demás nodos presentes en el árbol. Cuando se realiza la ecuación de complejidad asintótica las variables n seria la cantidad de nodos presentes en el árbol binario.

4) Simulacro de Parcial

4.1 *b)* ***4.1.2*** *d)*

4.3 *Línea 3: false*
Línea 5: suma == a.dato
Línea 7: sumaElCamino(a.der, suma-a.dato)
Línea 8: sumaElCamino(a.izq, suma-a.dato)

4.4.1 c)

4.4.2 c)

4.4.3 d)

4.4.4 a)

4.9 a)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

4.13.1 Línea 10: suma[raiz.id] = suma[e.id] + suma[raiz.id]

4.13.2 d)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473