Examen Febrero 2022 – Suma de mayores y eRank

Instrucciones

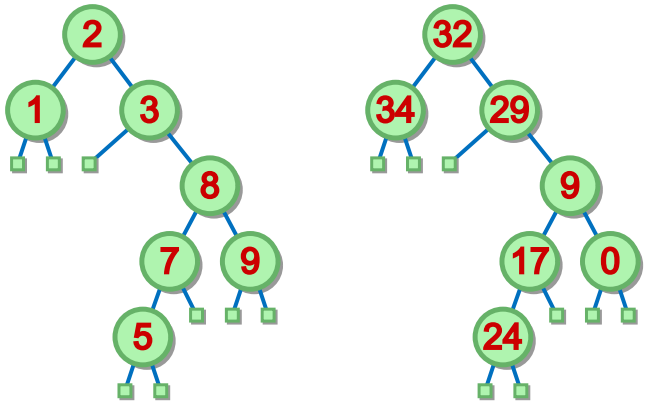
General Sum Tree

En base a un árbol de búsqueda binaria, el objetivo será el de **sustituir** cada **nodo por** la **suma** de los **nodos** con **mayor valor** que este.

Se puede conseguir de una forma más eficiente con el siguiente procedimiento y ayudándonos de una variable acumuladora:

- Se realiza la función con el hijo derecho del nodo.
- Se hace la sustitución y actualización de la variable acumuladora con el valor inicial del nodo.
- Se realiza la función con el hijo izquierdo.

Ejemplo:



eRank

Se deberá seguir el siguiente algoritmo con el objetivo de evaluar la conectividad de un vértice con el resto. Dado un valor a distribuir en un vértice:

- Si se supera el límite (threshold):
  - El valor del vértice se **incrementa** por la **mitad** del valor.
  - Se **distribuye** la otra **mitad** del valor entre los **sucesores** del vértice de forma **equitativa**.
- Si no se supera, no se hace nada.

Cómo proceder:

1. Para empezar, deberás crear un diccionario vacío con todos los vértices como claves y 0 de valor.
2. A continuación, desarrollarás un método distribuir que realizará el algoritmo anterior.
3. Finalmente, una función que distribuya un valor para cada de los vértices del grafo.

Código en Haskell

BinarySearchTree

```haskell
greaterSum :: BST Int -> BST Int
greaterSum x = fst (greateraux x 0)

greateraux Empty acc = (Empty, acc)
greateraux (Node i left right) acc = ((Node ac1 left' right'), ac3)
  where
    (right', ac1) = greateraux right acc
    (left', ac3) = greateraux left (i+ac1)
```

eRank

```haskell
import qualified DataStructures.Dictionary.AVLDictionary  as D
import qualified DataStructures.Graph.Graph  as G
import Data.Maybe

createDict :: (Ord a) => G.Graph a -> D.Dictionary a Double
createDict g = foldr (`D.insert` 0) D.empty (G.vertices g)

threshold :: Double
threshold = 0.00001

g1 :: G.Graph Char
g1 = G.mkGraphEdges ['A', 'B', 'C'] [('A','B'), ('B','C'), ('C','A')]

distribute :: Ord a => Double -> a -> G.Graph a -> D.Dictionary a Double
distribute i vertex graph = distributeaux i vertex graph (createDict graph)

distributeaux :: Ord a => Double -> a -> G.Graph a -> D.Dictionary a Double ->
D.Dictionary a Double
distributeaux i vertex graph dict
   | i > threshold = distributelist (G.successors graph vertex) evenly graph (D.insert vertex (val+half) dict)
   | otherwise = dict
    where
        half = i*0.5
        val = fromJust (D.valueOf vertex dict)
        evenly = half / (fromIntegral (G.degree graph vertex))

distributelist :: Ord a => [a] -> Double -> G.Graph a -> D.Dictionary a Double ->
D.Dictionary a Double
distributelist [] i g d = d
distributelist (x:xs) i g d = distributeaux i x g (distributelist xs i g d)

erank :: Ord a => Double -> G.Graph a -> D.Dictionary a Double
erank i graph = distributelist (G.vertices graph) i graph (createDict graph)
```

Código en Java

BinarySearchTree

```
import dataStructures.searchTree.BST;

public class BinarySearchTree {
    Node root;
    int size;

    private class Node {
        Node left;
        Node right;
        int value;
        private Node (Node l, Node r, int val){
            left=l;
            right=r;
            value =val;
        }
    }

   /** Based on José Enrique Gallardo BST's implementation from Data Structures, LCC, UMA **/
    public BinarySearchTree(){
        root=null;
        size=0;
    }

    public boolean isEmpty() {
        return root == null;
    }

    public void insert(int x) {
        root = insertRec(root, x);
    }

    public void insertList (int [] arr){
    for (int x: arr) {
        root= insertRec(root, x);
    }
}

    // returns modified tree
    private Node insertRec(Node node, int x) {
        if (node == null) {
            node = new Node(null, null, x);
            size++;
        } else if (x < node.value)
            node.left = insertRec(node.left, x);
        else if (x > node.value)
            node.right = insertRec(node.right, x);
        else
            node.value = x;
        return node;
    }

    private static String toStringRec(Node node) {
        return node == null ? "null" : "Node<" + toStringRec(node.left)
                + "," + node.value + "," + toStringRec(node.right) + ">";
    }

    @Override public String toString() {
        String className = getClass().getSimpleName();
        return className+"("+toStringRec(this.root)+")";
    }

     /***/

     public BinarySearchTree(Node root) {
         this.root = root;
     }
```

```
    public static void generalSum (BinarySearchTree A){
        generalSum(A.root, 0);
    }

    private static int generalSum(Node tree, int acu) {
        if (tree!=null){
            int aux = tree.value;
            if (tree.right!=null){
                acu = generalSum(tree.right, acu);
            }
            tree.value =acu;
            acu+=aux;
            if (tree.left!=null){
                acu = generalSum(tree.left, acu);
            }
            return acu;
        } else {
            throw new IllegalArgumentException("Árbol Nulo");
        }
    }


    public static void main(String[] args) {
        BinarySearchTree t = new BinarySearchTree();
        int [] aux = {5, 9, 7 ,8, 3, 1, 2};
        t.insertList(aux);
        System.out.println(t.toString());
        generalSum(t);
        System.out.println(t.toString());
    }
}
```

eRank

```
import dataStructures.dictionary.Dictionary;
import dataStructures.dictionary.HashDictionary;
import dataStructures.graph.DiGraph;
import dataStructures.graph.DictionaryDiGraph;
import dataStructures.graph.DictionaryGraph;
import dataStructures.graph.Graph;

public class eGraph<V>  {
    private Graph<V> graph;
    private Dictionary<V, Double> dict;
    private final double THRESHOLD=0.00001;

    public eGraph(Graph<V>  g){
        graph=(Graph<V>) g.clone();
        dict=new HashDictionary<>();
        for (V vertix:graph.vertices()) {
            dict.insert(vertix, 0.0);
        }
    }

    public void  distribute(V vertix, double rank){
        if (rank>THRESHOLD){
            double half = rank*0.5;
            dict.insert(vertix, (half+dict.valueOf(vertix)));
            double evenly = half/graph.degree(vertix);
            for (V succ: graph.successors(vertix)){
                distribute(succ, evenly);
```

```
            }
        }
    }

    public void eRank(double rank){
        for (V succ: graph.vertices()){
            distribute(succ, rank);
        }
    }

    public static void main(String[] args) {
        Graph<Character> g = new DictionaryGraph<Character>();
        g.addVertex('A');
        g.addVertex('B');
        g.addVertex('C');
        g.addEdge('A', 'B');
        g.addEdge('B', 'C');
        g.addEdge('C', 'A');
        System.out.println(g);
        eGraph e = new eGraph(g);
        e.eRank(1);
        System.out.println(e.dict);
    }
}
```

Y recordad, niños, revisad la entrega, no vaya a ser que tengáis el ejercicio bien, y saquéis un 4.9…

Bibliografía

Examen de 2022 de febrero de Estructuras de Datos, LCC UMA.