Laboratorio Nro. 4
Tablas de Hash y Árboles

Sebastian Castaño Orozco
Universidad Eafit
Medellín, Colombia
scasta31@eafit.edu.co

Dennis Castrillón Sepúlveda
Universidad Eafit
Medellín, Colombia
dcastri9@eafit.edu.co

1) Simulacro de proyecto

1.1 Se entrega en la carpeta código el ejercicio propuesto para las abejas. A continuación, se muestra el código.

```
"""
Estructura de datos y algoritmos 1
Laboratorio 4
Punto 1.1
Sebastian Castaño Orozco 201610054014
Dennis Castrillón Sepúlveda 201610035014

"""

import math
import pandas as pd
import numpy as np

data=pd.read_csv("PruebaLab4.txt",sep=',')
data=np.asarray(data)

def distance(a,b):
    lat1=data[a,0]
    lon1=data[a,1]
    lat2=data[b,0]
    lon2=data[b,1]
    alt1=data[a,2]
    alt2=data[b,2]
    rad=math.pi/180
    dlat=lat2-lat1
    dlon=lon2-lon1
    R=6372.795477598
    a=(math.sin(rad*dlat/2))**2 + math.cos(rad*lat1)*math.cos(rad*lat2)*(math.sin(rad*dlon/2))**2
    distancia_xy=2*R*math.asin(math.sqrt(a))
    distancia_xy=distancia_xy*1000
    dalt=abs(alt1-alt2)
```

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

```
    distancia=math.sqrt((distancia_xy**2)+(dalt**2))
    return distancia

distances=[]
close_bees=[]
for i in range (0,len(data)):
    for n in range (i,len(data)):
        distances.append(distance(i,n))
        if i!=n and distance(n,i)<100:
            close_bees.append(data[n])
            close_bees.append(data[i])

close_bees=np.asarray(close_bees)
```

2) Simulacro de maratón de programación

2.1 Se entrega el código de solución del problema de pasar de preorder a postorder mediante árboles binarios en la carpeta ejerciciosEnLinea. A continuación, se muestra este código.

```
"""
Estructura de datos y algoritmos 1
Laboratorio 4
Punto 2.1
Sebastian Castaño Orozco 201610054014
Dennis Castrillón Sepúlveda 201610036014
"""

class Node(object):
    def _init_(self, value):
        self.left = None
        self.right = None
        self.value = value

class BinarySearchTree(object):
    def insert(self, root, node):
        if root is None:
            return node
        if root.value < node.value:
            root.right = self.insert(root.right, node)
        else:
            root.left = self.insert(root.left, node)
        return root


    def post_order_place(self, root):
        if not root:
```

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 None
        else:
            self.post_order_place(root.left)
            self.post_order_place(root.right)
            print (root.value)

def post_order(node_order):
  List = Node(node_order[0])
  node = BinarySearchTree()
  for nd in node_order:
      node.insert(List, Node(nd))
  print ("------Post order ---------")
  node.post_order_place(List)


print(post_order([50,30,24,5,28,45,98,52,60]))
```

2.2 Se entrega el código de solución del problema de TreeSumming en la carpeta ejerciciosEnLinea, en este caso se realizar para el cálculo de la suma entre los nodos de un árbol dado de acuerdo a la estructura de datos de Binary Search Tree realizada previamente en los talleres. A continuación se muestra el código.

```
"""
Estructura de datos y algoritmos 1
Laboratorio 4
Punto 2.2
Sebastian Castaño Orozco 201610054014
Dennis Castrillón Sepúlveda 201610035014

"""

class BinarySearchTree:

    def __init__(self):
        self.root = None

    def __iter__(self):
        return self.root.__iter__()

    def insert_aux(self, value, node):
        if node == None:
            node = TreeNode(value)
            return node
        if value < node.value:
            node.left = BinarySearchTree.insert_aux(self, value, node.left)
        elif value > node.value:
            node.right = BinarySearchTree.insert_aux(self, value, node.right)
```

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 node

    def insert(self, value):
        if self.root == None:
            self.root = TreeNode(value)
        else:
            BinarySearchTree.insert_aux(self, value, self.root)

    def search_aux(self, value, node):
        if node == None:
            print("Not found")
            return False
        elif node.value == value:
            print("Found")
            return True

        if node.value > value:
            return BinarySearchTree.search_aux(self, value, node.left)
        elif node.value < value:
            return BinarySearchTree.search_aux(self, value, node.right)
        else:
            return False

    def search(self, value):
        BinarySearchTree.search_aux(self, value, self.root)

    def sumaElCamino(self,a,suma):
        a=TreeNode(a)
        if (a == None):
            return False
        if (a.left == None and a.right == None):
            return suma == a
        else:
            return(BinarySearchTree.sumaElCamino(self,a.left,suma-a)or
BinarySearchTree.sumaElCamino(self,a.right,suma-a))

class TreeNode:

    def __init__(self, value, left = None, right = None):
        self.value = value
        self.left = None
        self.right = None


arbol = BinarySearchTree()
arbol.insert(5)
arbol.insert(4)
arbol.insert(8)

print(arbol.sumaElCamino(arbol.root,9))
```

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

3) Simulacro de preguntas de sustentación de Proyectos

3.1 Expliquen qué estructura de datos utilizaron para calcular las colisiones entre abejas, por qué la eligieron y qué complejidad tiene el algoritmo que, utilizando dicha estructura de datos, calcula las colisiones.

Para el cálculo de las colisiones entre abejas se desarrolló un algoritmo analítico basado en las ecuaciones aprendidas en la carrera de Ingeniería Mecánica durante la carrera. Para este caso, se calcula la distancia entre cada par de abejas haciendo uso del teorema de Pitágoras y teniendo en cuenta el radio de curvatura de la tierra. Con ello, mediante la latitud, longitud y altura sobre el nivel del mar de cada abeja, se calcula para un par de abejas la distancia entre estas. Luego, se recorre todo el archivo de datos y se calculan las distancias, clasificando así las abejas que se encuentran a 100 metros o menos de otra.

A continuación, se presenta el cálculo de complejidad para este algoritmo:

```
distances=[]
close_bees=[]
for i in range (0,len(data)):
    for n in range (i,len(data)):
        distances.append(distance(i,n))
        if i!=n and distance(n,i)<100:
            close_bees.append(data[n])
            close_bees.append(data[i])
```

En el peor de los casos, realiza n^2 veces el proceso. Por tanto, el algoritmo es $O(n^2)$.

Sin embargo, la estructura de datos apropiada para realizar el código de colisión entre abejas es a través de Octrees, ya que para este caso, no se compararían las distancias entre todas las abejas sino que se revisarían las distancias entre abejas que estén cerca.

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

Este algoritmo subdivide el espacio tridimensional dependiendo de qué tan cerca estén las abejas y entrega las abejas que se encuentren a menos de 100 metros de distancia mediante estas mismas subdivisiones. Para este caso, la complejidad del algoritmo sería O(n) ya que en el peor de los casos subdivide n veces el espacio, siendo n el numero de abejas en cuestión, entendiendo que todas estuvieran cerca.

3.2 Se puede implementar más eficientemente un árbol genealógico para que la búsqueda e inserción se puedan hacer en tiempo logarítmico? ¿O no se puede? ¿Por qué?

No se puede ya que la complejidad del algoritmo de colisión entre abejas depende directamente de la cantidad n de abejas que se estén analizando, por ende, al incrementar el número de abejas que se tienen, la grafica de n vs tiempo incrementará, por ende, la recta de complejidad siempre tendrá una pendiente y no tenderá a estabilizarse en el tiempo luego de cierto valor de n como sucede con un algoritmo que posea complejidad logarítmica.

3.3 Expliquen con sus propias palabras cómo funciona la implementación del ejercicio 2.1 y el 2.2

3.3.1 Ejercicio 2.1

En este caso, el algoritmo reordena los valores de salida (print) para el árbol que se esté analizando. Por tanto, se recibe el árbol en pre-orden (raiz-izquierda-derecha), se construye este mismo mediante la clase nodo definida previamente y se vuelve a recorrer pero en post-orden (izquierda-derecha-raiz).

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

3.3.2 Ejercicio 2.2

Este algoritmo, crea un árbol binario de decisión en donde cada nodo tiene las propiedades definidas previamente, y para la función interna llamada SumarelCamino, se recorre cada camino posible desde la raíz hasta las hojas y se evalúa si la suma de los términos de ese camino es igual o diferente al valor requerido como input para esta misma función, que será el valor esperado de suma.

En este caso, el algoritmo hace recursión, recorriendo cada uno de los subárboles. Finalmente, devuelve True si para algún camino la suma de los nodos hasta la hoja es igual al valor suma y False en caso contrario.

3.4 Calculen la complejidad del ejercicio realizado en el numeral 2.1 y 2.2, y agregarla al informe PDF

3.4.1 Ejercicio 2.1

Para el peor de los casos, la complejidad del algoritmo es O(n) ya que el código deberá recorrer todos los nodos y reorganizarlos.

3.4.2 Ejercicio 2.2

Para el peor de los casos, el algoritmo no encuentra ningún camino que cumpla con la suma de los términos o encuentra un camino que si lo cumple luego de recorrer todos los caminos posibles. Por ende, estaría recorriendo cada mitad del árbol y subárbol, entonces, siendo n el número de nodos del árbol, la complejidad sería O(n log(n)).

3.5 Expliquen con sus palabras las variables (qué es ‘n’, qué es ‘m’, etc.) del cálculo de complejidad del numeral 3.3

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

3.5.1 Ejercicio 2.1

En este caso, n es el número de nodos que posee el árbol. En el peor de los casos el algoritmo debe reorganizar todos los nodos.

3.5.2 Ejercicio 2.2

En este caso, n hace referencia también al número de nodos del árbol.

4) Simulacro de Parcial

4.1

4.1.1 b

4.1.2 d

4.2 c

4.3

a) False

b) a.value

c) a.izq, suma-a.value

d) a.der, suma-a.value

4.4

4.4.1 b

4.4.2 b

4.4.3 d

4.4.4 a

4.6

4.6.1 c

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

4.6.2 return 0;

4.7

4.7.1 a

4.7.2 b

4.8 a

4.9 a

4.11

4.11.1 b

4.11.2 a

4.11.3 a

4.12

4.12.1 i

4.12.2 a

4.12.3 a

4.13

4.13.1 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