ESTRUCTURA DE DATOS 1
Código ST0245

Laboratorio Nro. 4
Tablas de Hash y Árboles

Juan Sebastian Guerra Hernández
Universidad Eafit
Medellín, Colombia
jsguerrah@eafit.edu.co

Jacobo Rave Londoño
Universidad Eafit
Medellín, Colombia
jravel@eafit.edu.co

3.1 explicación, documentación y análisis de complejidad del punto 1.1
La estructura de datos utilizada para calcular las colisiones entre abejas fue la de Octree. Esta estructura se define en forma de árbol, cuyos nodos poseen 8 hijos. Estos hijos representan un espacio resultante después de la división del espacio del nodo padre.
De forma recursiva, el espacio que representa un nodo se va repartiendo entre 8 nodos pequeños congruentes.
Con base en lo anterior, la implementación realizada en Python procede de la siguiente forma:

1. La lectura del archivo se hace en O(n), ya que el uso de la librería pandas se emplea para la construcción de un diccionario de los datos, y con el ciclo en el que se empieza a ingresar las abejas en una lista se reafirma la complejidad.

2. Al comenzar con la iteración del Octree, durante su creación se hace uso de getMins() y get Maxs(), métodos que tienen la intención de definir los posibles valores máximos y mínimos en cada uno de los ejes del espacio, para así definir un espacio cerrado para todas las abejas descritas en la entrada. Al tener que recorrer la mencionadas abejas, hace que su complejidad sea O(n).

3. Luego de definir los marcos, procedemos a encontrar el centro del cubo, que nos será útil para fraccionar el espacio mas adelante. Eso se logra al sumar los bordes y dividir en 2 (operación en O(1))

4. Para finalizar la construcción se le tiene que asignar los 8 hijos al espacio total. Para esto se implementó un método recursivo que busca encontrar eso hijos y, de forma recursiva, ir asignando los hijos de aquellas subdivisiones. Este planteamiento afirma que el caso base sea la existencia de al menos dos abejas o que la distancia más larga posible entre los extremos del hexaedro sea de 100: eso haría que filtre el espacio lo suficiente para saber si dos abejas están lo suficientemente cerca para una colisión.
Este método en primera instancia buscamos ir asignando las abejas a su lista correspondiente basado en su posición, eso se hace en O(Log8 n) ya que cada vez que se va segmentando las abejas, sus cantidad de abejas se va dividiendo en 8. Sin embargo,

como el método busca llenar todos los espacios, va a ser que finalmente se termine recorriendo la totalidad de las abejas, lo que lo vuelve O(n). Ahora bien, el último paso consiste que para cada hijo se le haga la asignación completa de sus hijos. Conociendo que son 8 hijos siempre, hace que el for sea O(8), ahora bien, como adentro se consolida la recursión, y habiendo mencionado las implicaciones con la asignación de sus abejas implica que la complejidad en tiempo sea O(n).

5. Para finalizar, a lo largo de la construcción también fuimos asignando a una lista las veces que llegábamos al caso base y aún había al menos 2 abejas (un choque). En la última parte del main, procedemos a recorrer esos choques, que se haría en O(n) asumiendo que el peor caso sea en el que se choquen todas las abejas.
Para finalizar, al aplicar las propiedades de notación O, podemos concluir que la complejidad en tiempo del algoritmo es de O(n), siendo n el número de abejas.

3.3 Teniendo en cuenta que recibimos un arreglo con el recorrido de un árbol en preorden lo ideal es construir ese árbol con dicho recorrido. Por esta razón implementamos el método buildingTree quien se encarga de recorrer los elementos del recorrido para así de forma recursiva ir asignando nodos dentro de un árbol binario. Luego de esto implementamos una función posorden que busca hacer el recorrido de los nodos del árbol de la forma -izq, der, cabeza-. Este método, al igual que lo hicimos durante la inserción también es recursivo. De esta forma podemos entregar el recorrido por posorden del mismo árbol binario.

3.4 La complejidad del algoritmo es O(n^2), ya que utilizamos el método recursivo *insert* dentro de un ciclo en el cual vamos recorriendo los elementos del *array*

3.5 En la complejidad del ejercicio anterior n es el número de elementos en el arreglo de entrada, es decir el número de nodos que finamente va a tener el árbol.

4) Simulacro de Parcial

4.1.1 B
4.1.2 D
4.2 OPCIONAL C
4.3 Línea 3 : false
Línea 5 : suma == 0
Línea 7 : (a.left, suma-node.data)
Línea 8 : (a.right, suma-node.data)
4.4.1 OPCIONAL C
4.4.2 OPCIONAL A
4.4.3 OPCIONAL D
4.4.4 OPCIONAL A
4.6.1 OPCIONAL A
4.9 A
4.13.1 Línea 10 : raiz.id

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

4.13.2 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