Laboratory practice No. 4: Hash Tables and Trees

Andres Felipe Agudelo Ortega
Universidad EAFIT
Medellín, Colombia
afagudeloo@eafit.edu.co

Juan Camilo Gutiérrez Urrego
Universidad EAFIT
Medellín, Colombia
jcgutierru@eafit.edu.co

April 25, 2021

1) Killer Bees

For this point, we programmed an Octree algorithm (see 1), in which we divided the entire set of point into smaller groups with a certain tolerance depending on the total bee size. For example, for a 1500 size bee set we could take a tolerance of 20 bees for each "boundary" of the Octree, so in each smaller group there would be only a maximum of 20 bees. All this elaborated process has a reason to be done; we want to solve a really big problem, which has an $O(N^2)$ complexity to compare each bee with all the others and the distance between them so if we took the entire set of bees, it would take an unacceptable amount of time. That's where Octrees come, we are making smaller sets of bees, which we can compare and get N for the $O(N^2)$ way smaller.

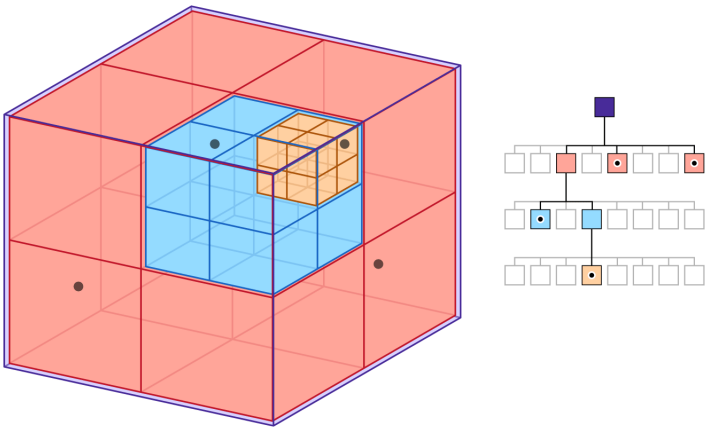

Figure 1: Octree Example

2)

2.1. Solve the problem using binary trees

This point is solved in Python, and the code is attached on its respective GitHub folder

3)

3.1. Explain what data structures you used to calculate bee coalitions, why did you choose it and what complexity does this algorithm have

The chosen data structure for this problem was the Octree, as we needed to make smaller the sets of bees we wanted to compare. At first we were studying the Quadtree, but we needed the extra dimension for all three bees coordinates. The complexity for this data structure for searching and inserting is $O(NlogN)$, so we can first divide our large sets using this method, and then executing a comparison between the points in each boundary.

3.4. Calculate point 2.1 complexity

At first, for this point we inserted every number in the binary tree, which complexity in the worst case tended to be $O(log_2N)$, but then, we needed to print every number in the "pos-orden", so we needed to recursively go through every node, so we obtained a $O(N)$ worst complexity which actually is the worst complexity of our algorithm.

4) Practice for midterms

4.1.

4.1.1 b

4.1.2 d

4.2.

4.2.1 It returns the node where it first finds the value of n1 or n2

4.2.2 T(n) = 2*T(n/2) + C que es O(n)

4.2.3 We could make the mistery algorithm compare n1 and n2 with the value of each root, and only send it to the right or left, depending on if it's less or greater than the root.key. So it does not have to compare n1 and n2 with ALL the "children", but only with half of them

Professor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co

4.3.

4.3.1 Return True

4.3.2 The complexity would be O(N), because there are two separated cycles, and each one is inserting values in a dictionary, which is O(1). Note: As the dictionary can only have one value per key, there shouldn't be a O(N) inserting complexity

4.4.

4.4.1 c

4.4.2 a

4.4.3 d

4.4.4 b

4.5.

4.5.1 $toInsert == null$

4.5.2 $p.key < toInsert$

Professor Mauricio Toro Bermudez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co