Laboratory practice No. X: Complete the title of the laboratory practice

Sebastian Herrera Beleño
Universidad Eafit
Medellín, Colombia
sherrerab1@eafit.edu.co
hijueputa

Jairo Miguel Marulanda Giraldo
Universidad Eafit
Medellín, Colombia
jmmarulang@eafit.edu.co

bobo

3) Practice for final project defense presentation

3.1

A Spatial Hashing data structure was used to store the data. This structure was chosen because of its simple implementation and its fast search capability. In a nutshell, the structure saves in the same memory space points located close to each other using a simple hash function, which generates its keys by dividing each point parameter by the desired radius, that in our case is 100 meters. This can be conceptualized as locating points in blocks inside a 3D space, so we can first look for such blocks instead of the point directly, reducing time complexity.
However, some problems were present when formulating the search function.
Our first attempt, the retrieveCell function, which simply returned a linked list with all the points inside a given block, would be prone to error due to many points being in different blocks, despite having less than 100 meters between them. This function had a worst case complexity of O(n).
The solution that was finally implemented was the function getFriends, which looks inside not only the block where a given point is located, but inside any neighboring cell that touches a circumference with the given radius and that originates in the given point. Calculating the complexity for this function proved much defiant, since its dependent in many factors, we estimate O(n*m), with n being the number of keys or blocks inside the search radius and m being the average number of points stored inside each block.

3.2

An Octree data structure could have proven more efficient, since by the very formulation of its structure it is organized so that one has not to look inside so many blocks. On the same note, part of why our function proved inefficient was that the keys inside the dictionary implemented inside the location hash data structure were not organized in any particular manner other than insertion order. If we could some how organize this keys we could directly look inside the ones closest to the point in question, reducing search time.

3.3

3.4

To insert the complexity in average case is O (log (h)), in the worst case is O (N). For p reorder and postorder algorithm the complexity is O (N).

3.5

3.6

4) Practice for midterms

4.1 *b, d*

4.2 1.The function returns the node with the largest deepness whose branches both contain either n1 or n2. If such node does not exist, then it returns the node with lowest deepness which contains either n1 or n2. If such node does not exist either, then it returns a null value.
2. The complexity is O(n) with n being the number of nodes.
3. For each recursion call, the algorithm could check if n1 and/or n2 are either smaller or bigger than the value stored in the current node. Then it could be able to determine which path to follow, for it could guarantee that the data would not be stored in any other path.

4.3 1. return True
2. O(n^2)

4.4 We suspect a mistake in the structure for this exercise. However, by discard:
1. c
2. a
3. d
4. a

4.5 a. p.data == toInsert
b. p.data < toInsert

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473