Gray Houston

ghousto

0026483532

Project 4 Analysis Questions

**For all questions, the variable ‘*n*’ refers to the number of vertices and the variable ‘*m*’ refers to the number of edges.**

1. The asymptotic performance of my recursive tree height function is O(*n*), where *n* is the number of nodes in the tree. The algorithm basically works by finding the maximum height of the current children and returning that height plus 1, while watching for the leaf node corner case.

Because of the tree structure, each node can only be accessed by its parent. Therefore, once a parent node has recursively calculated the height of a child (and the algorithm starts moving back down the recursive stack), that child’s height will never be used again and that child node will not ever be accessed again. Proof for this statement:

Because the child’s height is only used to calculate the parent’s height, a child’s height is only needed exactly once. Therefore, once a parent’s height has been determined, it never needs to be accessed again. Furthermore, because the recursive accessing is essentially a DFS traversal of the tree, each node will be accessed once.

For this reason, throughout all of the recursive calls, each node will be accessed exactly once. Said another way, there will be *n* node accesses in the algorithm. Therefore, the asymptotic performance of the algorithm will be O(*n*), where *n* is the number of nodes in the tree.

It should be noted that there will be some overhead because of the recursion. However, as *n* increases and approaches its asymptotic boundary (i.e. as *n* approaches ∞), this recursive overhead becomes insignificant and negligible. Therefore, this overhead shouldn’t be considered for the asymptotic performance.

1. The asymptotic performance of my non-recursive tree height function is O(*n*), where *n* is the number of nodes in the tree.

The algorithm uses as BFS approach to traversing the tree. It basically works by adding the children of all the elements of a certain level (i.e. all vertices with the same height), beginning with the root’s children, to the queue, then removing the current level’s nodes from the queue. Each time a new level is reached (i.e. the queue isn’t empty), the height counter increases. Thereby, the algorithm traverses the tree by moving from one height level to the next.   
  
Excluding the root, each node is accessed twice: first when it is added to the queue, and second when it is removed from the queue (when its children are added to the queue). Because the root is never added to the queue, it is only accessed once. Therefore, the running time for the algorithm is O(2*n* – 1). Therefore, the asymptotic performance (as *n* approaches ∞) for my algorithm is O(*n*), where *n* is the number of vertices in the tree.

1. As stated above, the recursive tree height function essentially implements a Depth-First Search (DFS). The algorithm recursively moves down the levels of the tree (from parent to child) until it hits a leaf. When a leaf is hit, it moves up one level to the parent, and then recursively tries to follow other children, recursively move down tree height levels until a leaf node is hit.

Because this algorithm attempts to travel a single path as deep into the tree as possible before moving back up a level and trying a new path, the algorithm is a DFS traversal. Furthermore, because this function operates like a DFS, it traverses all of the nodes in a pre-order traversal (accesses parent first, then child nodes from left to right).