CSCE 629 - 602 Analysis of Algorithms

October 5, 2016

Homework IV
Prof. Anxiao (Andrew) Jiang

Rishabh Yadav
UIN: 425009824

1A. 22.2-8 Solution

Idea

The basic idea is to recursively calculate the *diameter* of the child *sub-tree* The diameter of the tree is maximum of the following:

1. Maximum of Diameter of child *sub-trees.*
2. A path which include the *root* node. That is, the longest path between vertices that goes through the root of T (this can be computed from the heights of the sub-trees of T)

The idea is to write an algorithm which recursively calculates the Diameter of sub-trees and return the Diameter of the tree T.

If $D(r)$ is the diameter of the tree T with root node r, and $H(r)$ is the Height of the tree T rooted at r. Then recursive relation is:

$D(r) = max\ \{D(r.child),\, 1 + H(r.child_i) + H(r.child_j)\}$

where two sub-trees with max height are rooted at $child_i$ *and* $child_j$

We just need to look at the max of the above values to return the diameter.

Pseudo Code

```
Diameter(root, heightRef[]):
  1. int childDiameter = []//childDiameter- diameter of child subtree
  2. if(root == NULL)
     (a) heightRef[] = 0
```

```
    (b) return 0 //diameter is also 0
3. for all children of $root$:
    (a) calculate the diameter of each children and keep track of them in an array = Diameter(root.child, heightRef)
    (b) calculate the height of each children (child $i$ here) and keep track of them in an array,
        heightRef[i] = heightRef[i]+1
4. return max(height of $child_i$ + height of $child_j$ + 1, max(childDiameter)) //$child_i$ and $child_j$ are top t heights among all the children, if there is only one child then $child_j$ is 0.
```

Proof of correctness

By the concept of generality we can say that the diameter of a tree can be one of the following.

- Either equal to the diameter of one of its sub-tree. OR,
- It could be a path passing through the root itself.

The second case will be the top 2 maximum height of child sub-tree plus 1 for itself. In the algorithm proposed we cover all the cases and is hence exhaustive. Therefore giving us the correct Diameter.

Time complexity

Since we are traversing each node exactly once the proposed algorithms run in linear time. hence time complexity is $O(V)$, where are V is the number of vertices.

Time complexity = O(V)

2A. 22.3-13 Solution

Idea

The idea is to solve this problem by running Depth first search, (DFS), multiple times by taking every vertex as source.
Run DFS for every vertex in the graph as source and if a visited vertex is encountered again, then the graph is not singly connected. We repeat this operation for every vertex which is not visited in the previous DFS. If the operation executes successfully without any break then we he graph is singly connected.

Pseudo code

```
1. Check Singly Connected ():
   (a) for all vertices $v$ in graph $G(V, E)$
         i. Perform DFS on $v$
        ii. if already visited vertex encountered
              A. return Not-Singly-Connected
       iii. if any vertex, $u$ not visited then perform DFS with vertex, $u$
   (b) the operation was finished successfully hence return Singly-Connected
```

Proof of correctness

The proposed algorithm is essentially checking for forward edges and cross edges in the given graph. We will try here to prove that the graph if has forward or cross edges then we conclude that the graph is not singly connected.

Let us say there is a forward edge (u, v) in G, then there must be a path p separate from (u, v) which has been found in order for (u, v) to be labelled forward. But p and the edge (u, v) are two vertex-disjoint paths from u to v and this contradicts the fact that G is singly connected. Therefore, there are no forward edges in G. Now let us , assume there is a cross edge (u, v), then u and v must share a common ancestor r, the *root* of the DFS tree, so there are two vertex-disjoint paths from r to v, one via u and also another direct through r to v. Thus by contradiction we can say there are no cross edges within the graph for it to be singly connected.

Time complexity

Since we are performing a DFS on every vertex as source vertex, and we know a DFS takes $O(V + E)$ time. Hence time complexity is $O(V(V + E))$, where are V is the number of vertices and E is the total number of edges in graph $G(V, E)$

Time complexity = O(V(V+E))

References

Introduction to Algorithms by T. Cormen, C. Leiserson, R. Rivest, C. Stein