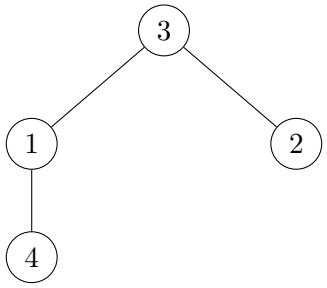
CSCI-GA.1170-001/002 Fundamental Algorithms November 10, 2015

Solutions to Problem 4 of Homework 7 (10 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, November 10*

Given an undirected rooted tree T with n nodes, with possibly negative weights $w(v)$ assigned to its vertices v, the weight of the tree is the sum of the weights of all the nodes in the tree. (The weight of the empty tree is 0.)

A subtree is any connected subgraph of a tree, including an empty tree as a pathological special case. For example, in the tree below the subtrees are $\emptyset$, $\{3\}$, $\{1\}$, $\{2\}$, $\{4\}$, $\{3,1\}$, $\{3,2\}$, $\{1,4\}$, $\{3,1,4\}$, and $\{3,1,2\}$, but not $\{3,4\}$.



The goal of the problem is to design a polynomial time algorithm to find a subtree (possibly empty) with maximum weight, and analyze its running time.

(a) (4 pts) Given a node v, let $T(v)$ be the complete subtree of T rooted at v (so $T = T(T.root)$), and WithRoot(v) to be the maximum weight of a all subtrees of $T(v)$ which must include the node v. For example, if the tree above, WithRoot(1) is the maximum weight of sub-trees $\{1\}$ and $\{1,4\}$ (but not $\{4\}$ or the empty tree). Give the dynamic programming recurrence relation for WithRoot(v) (don't forget the base when v is a leaf in T), and use it to analyze the running time of a dynamic-programming procedure to compute the values WithRoot(v) for all nodes $v \in T$ (including with $T.root$).

Solution:

This problem is similar to the problem of Maximum-Sum-SubArray. In this case, the array is replaced by a tree. Let T be a tree with a root v and let $v_1, \ldots, v_n$ be the children of v. By analogy it is easy to observe that, *WithRoot(v)* is simply the union of all *WithRoot(v_i)* which have weight greater than 0 together with v

$$WithRoot(v) = \begin{cases} v & \text{v is a leaf} \\ \bigcup_{\geq 0} \{WithRoot(v_i)\} \cup v & \text{otherwise} \end{cases}$$

Start from the root node v and make recursive calls to each of it's child and when the recursive call is returned back, store the returned value into the appropriate entry of the array (used for memorization) corresponding to the respective node.

A recursive call is made on every node of the tree exactly once. Therefore the time taken to calculate *WithRoot(v)* for every $v \in T$ is $O(|v|)$, where $|v|$ denotes the number of nodes in the tree T

□

(b) (4 pts) Given a node v, let $T(v)$ be the complete subtree of T rooted at v (so $T = T(T.root)$), and TOTAL(v) to be the maximum weight of a all subtrees of $T(v)$ which must may or may not include the node v. Assume you already solved part (a) and computed all values WITHROOT(v).

Give the dynamic programming recurrence relation for TOTAL(v) (don't forget the base when v is a leaf in T), and use it to analyze the running time of a dynamic-programming procedure to compute the values TOTAL(v) for all nodes $v \in T$ (including with $T.root$, which gives the answer to the original problem). What is the running time of this algorithm?

Solution:

Given that TOTAL(v) may or may not include the the node v. We have already analyzed the case when v is included. Therefore, now analyze the case when v is excluded and then take the maximum of both these cases. When v is excluded it will just be the maximum weight of one of it's children $v_1, \ldots, v_n$

$$WithoutRoot(v) = \begin{cases} \emptyset = 0 & \text{v is a leaf} \\ max\{Total(v_1), \ldots, Total(v_n)\} & \text{otherwise} \end{cases}$$

$$Total(v) = max\{WithRoot(v), WithoutRoot(v)\}$$

The running time of $WithoutRoot(v)$ is similar to that of $WithRoot(v)$. A recursive call is made on every node of the tree exactly once. Hence the running time will be $O(|v|)$.

Calculating $Total(v)$ involves taking maximum of two quantities which takes $O(1)$ time. Therefore, the total running time of this algorithm is $O(|v|)$ □

(c) (2 pts) What will the running-time of the procedures in parts (a) and (b) be if we use standard recursion, and not dynamic programming?

Solution:

Note that the running time of part(a) is similar to that of part(b). Hence the running time of both these procedures will be effected in a similar way if we use standard recursion and not dynamic programming.

Let the total number of nodes in T be n and v be any node in the tree then the time taken to calculate $WithRoot(v)$ and $Total(v)$ is $O(|v|)$. Let $v_1, v_2, \ldots v_n$ be the the nodes in the tree, then the total time taken without memorization is

$$T(n) = \sum_{i=1}^{n} O(|v_i|)$$

In the worst case, the tree takes the structure of a linked list. Then the above sum will evaluate to $O(n^2)$. Therefore if we use recursion without memorization, the running time increases by a factor of n.

However, note that in part(a) and part(b), we used memory to store the values returned by the recursive calls. Therefore, if we are allowed to use memorization in standard recursion, the running time will $O(n)$. So the running time depends on if we're allowed to use memorization or not in the standard recursion

□