Michael Rodrigues
rodrigues.mi@husky.neu.edu
November 24, 2014

CS4800: Programming Assignment

Table of Contents

Mathematical Formulation of the Problem

A company is comprised of a hierarchy which can be described as an undirected, acyclic graph. The source node of the graph is the CEO, and all other nodes are employees. Any node that contains edges leaving this node are managers, or bosses. The nodes that do not contain any outgoing edges are the lowest level employees.

Suppose an employee wants to spread propaganda in a company. This employee only has enough time to talk to a certain number of employees. When this employee influences another employee, the employee that was influenced will also spread the propaganda to his or her superior (boss), who in turn influences his or her superior (boss) all the way until the propaganda percolates back up to the CEO at the root of the hierarchy. If a superior (boss) has two of his subordinates (employees) approach him with the propaganda, he can only influence his superior (boss) once, thus his value cannot be duplicated or multiplied. Further, spreading propaganda to one employee twice only effectively influences them once., thus they cannot be re-influenced.

Given a number of employees n and the number of employees that should be influenced j (where $j \leq n$), this algorithm must determine the optimal solution, which is the maximum influence value possible in the company given the assigned values to each employee and the hierarchy of the company. More specifically, given a company hierarchy tree T and the number of employees to influence j, this algorithm must determine some subset of m employees from the company hierarchy which shall maximum the sum of the individual influence values of each employee in this subset. The output of the algorithm is the sum of the influenced employee's individual influence values.

$$MAX(\sum_{i=1}^{m} i.value)$$

High-Level English Description of the Algorithm

The algorithm that I used to solve this problem basically traverses the tree starting from the CEO (root) of the hierarchy. Initially, as algorithm walks down the tree, it sets a value for each employee which is the value for the node that is currently visiting plus the value computed for the current employee's boss. This basically sets the tree up as though influencing an employee would also influence that employee's superiors, all the way back up to the CEO (root of the hierarchy). Although this may not truly be the case since an employee may have both of his subordinates influence him or her, it is vital to finding the most influential employees. As the algorithm starts to walk back up the tree, each manager (non-leaf/non-lowest-level) employee, keeps track of the maximum value that any of its employees have reported back to it. It then loops through all non-maximum branches and recursively fixes the values by treating this branch as a another instance of the initial problem. In other words, it resets the employee values to account for the fact that this employee's superior has already been influenced and thus decreases the overall residual effect of influencing this employee.

After all of these computations have been performed, we are able to sort the list of residual effect values in decreasing order. From this, we can then select the given number of employees with the highest residual influence from the list and compute the sum of these influence values. This is ultimately the result that the algorithm prints to stdout and finally exits.

Pseudo Code Description

The algorithm can be expressed in Pseudo Code as:

```
1. Read the number of employees and the number of employees to influence
2. values = [] # Value of each employee (non-cumulative)
3. effects = [] # Cumulative value of influencing this employee and all superiors
4. bossIds = [] # Array of boss ids
5. subordinates = [] # Array of arrays of employee ids (subordinates list)
6. residualEffect = [0] * (num_employees+1) # Array of residual value for each employee
7. For i=1 to number of employees:
8.   Read a line from stdin and parse the values into id, boss id, and value
9.   Set values[i] to value
10.  Set residualEffect[i] to (value + residualEffect[boss id]) # Cumulative effect
11.  Set bossIds[i] to boss id
12.  Add id to subordinates[boss id]'s array
13.EndFor
14.# id: the id of the employee to look at, value: value to pass
15.FixTree(id, value):
16.  Set residualEffect of this employee to value + this employee's value
17.  If employee with id == id has no subordinates:
18.    return (id, residualEffect[id], residualEffect[id])
19.  Let branchMax = residualEffect[id], and branchMaxId = id
20.  For each subordinate of employee with id == id:
21.    Call FixTree passing the subordinate id, and the current residualEffect
22.    Retrieve the subId, subEffect, and subBranchMax from the return value
23.    If the subBranchMax value > branchMax: Update the branchMaxId and branchMax values
24.  Set the residualEffect of this employee to 0 # better to influence a subordinate
25.  For all subordinates that are not the maximum of this employee:
26.    Call FixTree passing the subordinate id and a residualEffect of 0
27.  Return (id, residualEffect[id], branchMax)
28.# id: the id of the employee to look at, value: value to pass
29.ScanTree(id, value):
30.  If employee with id == id has no subordinates:
31.    return (id, residualEffect[id], residualEffect[id])
32.  Let branchMax = residualEffect[id], and branchMaxId = id
33.  For each subordinate of employee with id == id:
34.    Call ScanTree passing the subordinate id, and the current residualEffect
35.    Retrieve the subId, subEffect, and subBranchMax from the return value
36.    If the subBranchMax value > branchMax: Update the branchMaxId and branchMax values
37.  Set the residualEffect of this employee to 0 # better to influence a subordinate
38.  For all subordinates that are not the maximum of this employee:
39.    Call FixTree passing the subordinate id and a residualEffect of 0
40.  Return (id, residualEffect[id], branchMax)
41.Call scanTree(1, 0)
42.Sort the residualEffect array in increasing order
43.Print the sum of the first number to influence array entries.
```

Proof of Correctness

This algorithm is correct and outputs the correct optimal value given the hierarchy and the number of employees to be influenced. Suppose this algorithm outputs a value *S* which is not the optimal solution, and instead *S'* is the optimal solution. This means that there is some subset *s'* which contains a higher sum of values than the subset *s* that is constructed by my algorithm. Since this algorithm corrects the values in the hierarchy at each point when we determine a branch maximum, the values the algorithm assigns to each node accurately portrays the residual influence value taking into account the managers that have already been influenced, then the optimal solution *S'* either counted managers influence values multiple times which violates the problem requirements yielding a contradiction or influenced the same employee multiple times which also violates the problem and thus yields a contradiction.

This algorithm also works correctly on a company hierarchy of any size and with any number of employees that can be influenced. Suppose the hierarchy contains a single employee, the CEO, and we can influence no employees. Since we cannot influence any employees, the maximum influence value is zero. Suppose we keep the same hierarchy, but can influence one employee. Then, we will influence the CEO and thus we have achieved the optimal solution. Then, suppose we have two employees, the CEO with id 1 and one subordinate with id 2. If we cannot influence any employees, then the optimal solution will again be. If we can influence one employee, then the algorithm walks over the hierarchy and assigns values to each node. The algorithm will then pick the one with the highest residual value which will be the maximum of value(1) and value(2) + value(1), thus yielding the optimal value. If we increase the number of employees that can be influenced to two, then we can influence all employees, and thus value(2) + value(1) will be optimal. This algorithm can then also determine the case where there are $n+1$ employees and j employees that can be influenced, where $j \leq n+1$ as it will traverse the hierarchy, computing and assigning residual values to each employee which will then be used to determine the optimal solution.

Run-Time Analysis

First, the Python implementation of this algorithm reads a line of input from stdin which contains the number of employees n in the hierarchy and the number of employees that can be influenced j in O(1) time. It then loops O(n) times to read in all of the employees in the hierarchy. For each employee, it appends the retrieved information to different arrays each in O(1) time. Next, the algorithm calls scanTree(1, 0) which walks over the hierarchy starting from the CEO (root node).

This function retrieves the subordinate list in O(1) time of the current employee, and then recursively calls scanTree on each subordinate until we reach the bottom of the tree. After each scanTree call on subordinates, a O(1) time amount of work is performed to update the maximum branch value seen so far, if appropriate. After all subordinates have returned, the algorithm updates this manager node's residual value to 0, and then calls another function, fixTree, which corrects the values of non-maximum subordinates. The implementation of fixTree is almost identical with the exception of one additional O(1) operation. Since each node is initially visited once, but then may be visited again to correct any values that may have been computed, the algorithm may, in the worst case, visit some nodes O(log n) times, thus O(n log n) time is required to traverse the entire tree.

Once all nodes have received a residual value from the traversal of scanTree, the implementation then sorts the array of residual values in O(n log n) time. Lastly, the residual values array is then sliced from the beginning to the appropriate index such that only the given number of employees to influence *j* are computed. According to the Python Wiki TimeComplexity page, this requires O(k) time where k is the number of indices that are being sliced, thus O(j) in my case. Then, a sum operation is performed on this slice which requires O(j) time as well.

The overall run time required by this algorithm is O(n + n log n + n log n + j + j). Therefore this algorithm runs in O(n log n) time which is dominated by both the time required to traverse the tree as well as the time required to sort the array of residual values. The run time of some Python function implementations was confirmed via the Python Wiki TimeComplexity page (Python TimeComplexity).

Python Run-Time Sources

Python Wiki TimeComplexity: https://wiki.python.org/moin/TimeComplexity