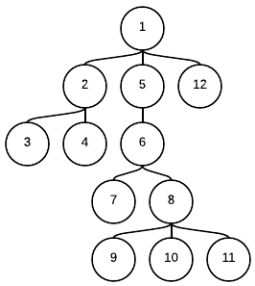
Michael Rodrigues (rodrigues.mi@husky.neu.edu) November 24, 2014

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 manager, who in turn influences his or her manager all the way until the propaganda percolates back up to the CEO at the root of the hierarchy. If a manager has two of his or her subordinates (employees) approach him with the propaganda, he can only influence his or her manager once, thus his or her value cannot be duplicated or multiplied. Further, spreading propaganda to one employee twice only reaps the benefit of influencing them once, thus they cannot be re-influenced.

Suppose we have the following company hierarchy. If employee 4 is influenced, then the value of influencing employee 4 is: value(4) + value(2) + value(1). If we were to instead influence employee 10, then the value of influencing employee 10 is: value(10) + value(8) + value(6) + value(5) + value(1). If employee 4 was influenced, and then employee 10 is influenced, then the value is: value(4) + value(2) + value(1) + value(10) + value(8) + value(6) + value(5). Note that the value of employee 1 is only recognized once.



Given a number of employees n, the number of employees that should be influenced k (where $k \leq n$), and the company hierarchy T consisting of employees $\{e_1, e_2, \ldots, e_n\}$ this algorithm shall determine some subset of k employees $\{e_1', e_2', \ldots, e_k'\}$ to influence such that the sum of the values from the union of Influence(e_i) where $1 \leq i \leq k$ is maximized. The function Influence(e) can simply be defined such that given an employee e, it produces the set of all employees that would be influenced as a result of influencing e. Finally, this algorithm must output the sum of the influence values obtained by influencing these k employees.

$$MAX(\sum Values(Influence(e_1) \cup Influence(e_2) \cup \ldots \cup Influence(e_k)))$$

High-Level English Description of the Algorithm

The algorithm that I used to solve this problem is a mix of a greedy algorithm and a divide and conquer style approach. At a basic level, it traverses the tree starting from the CEO (root) of the hierarchy. Initially, as algorithm walks down the tree, it loops through all subordinates, treating the sub-hierarchy as a new instance of the same type of problem. At each subordinate, the algorithm assigns a value for each employee which is the

value for the node that is currently being visited 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. Although this does not provide the optimal solution since a manager 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, keeps track of the maximum value that any of his or her employees have reported back to it. This value can be defined as the "branch" value, which is the maximum value obtainable by influencing that subordinate (if he or she is a lowest-level employee) or some other subordinate that is further down in the hierarchy in that specific branch. The individual subproblem results are then combined by determining the non-maximum reporting subordinates. Once they have been determined, the algorithm then loops through these subordinates 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, or updates the residual value of the employee, to account for the fact that this employee's superiors have already been influenced and thus decreases the overall residual effect of influencing this employee. The residual value of an employee can be defined as the value obtained by influencing the given employee, taking into account that influencing an employee also causes the employee's manager to be influenced. It also takes into account that an employee's influence value cannot be counted twice.

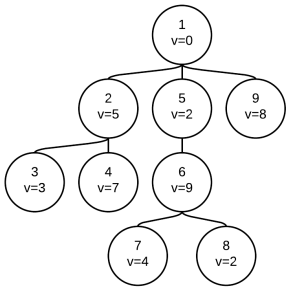
After all of these computations have been performed, we are able to sort the list of residual effect values in decreasing order. From this, we can then select the given number of employees with the highest residual influence from the list and compute the sum of these influence values. This is ultimately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

Proof of Correctness

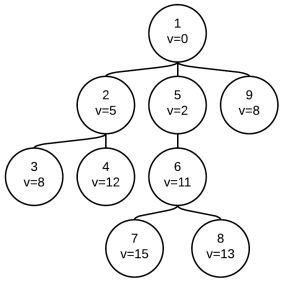
Suppose that initially the algorithm has not selected any employees to influence. It then picks employee e_1 with the highest residual value. Suppose instead that influencing e_1' is more optimal. Since e_1 provides the highest residual influence value, and no employees have been influenced yet, he or she is the most influential employee in the entire hierarchy, thus the value of influencing e_1 is greater than the value of influencing e_1', yielding a contradiction that e_1' is more optimal. Suppose that now we have chosen to influence $\{e_1\}$ which has been shown to be optimal. Then suppose this algorithm selects employee e_2 with the highest residual value. Suppose instead that influencing e_2' is more optimal. Since e_2 provides the highest residual influence value in the residual hierarchy, and the only employee that has been influenced was the optimal choice, e_2 is the most influential employee in the entire hierarchy, thus the value of influencing e_2 is greater than the value of influencing e_2', yielding a contradiction that e_2' is more optimal. This can be further expanded such that the k-1 employees with the highest residual influence value have already been selected. Suppose that the algorithm selects e_k with the highest residual influence value. Suppose instead that influencing e_k' is more optimal. Since e_k provides the highest residual influence value, and the k-1 employees with the highest residual influence value have already been selected, he or she is the most influential employee in the entire hierarchy, thus the value of influencing e_k is greater than the value of influencing e_k' and therefore influencing e_k is more optimal, thus a contradiction.

Suppose a simple hierarchy with employees 1, 2, and 3. Employees 2 and 3 are direct subordinates to employee 1 who is the CEO. Suppose employee 1 has a value 0, employee 2 has a value 1, and employee 3 has a value 5. This algorithm first walks down the tree setting the value such that the residual value of influencing

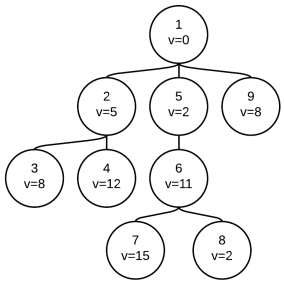
an employee is the value of the employee and the sum of the values of all of that employee's managers. Once it has finished walking down the tree, it walks back up from the leaf nodes. At each manager, it determines the maximum that was reported back to it from any of its subordinates. Each of the non-maximum subordinates is then recurred on again, but it passes a value of 0 so that the residual value assignment starts over again so as to not include the values of the managers thus far. Suppose that it is not correct to reset the value assignment so as to exclude the value of a non-maximum branch's managers. If it is not optimal to reset the values, then the algorithm would not need to perform any value changes as it walks back up the tree, thus meaning that the value assigned to each employee is his or her own value plus the sum of the influences of his or her managers. Thus, if this were the case, then the optimal solution would re-count the influence values of some managers several times, thus violating the problem statement and therefore yielding a contradiction.



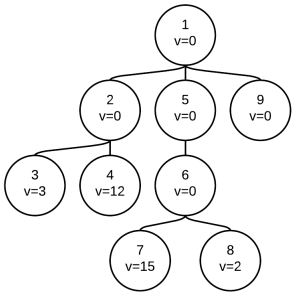
Initial Tree



Residual Value Tree
After walking down the tree



Residual Value Tree
Starting walk back up



Final Residual Value Tree
After walk back up finished

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 *k* 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) 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 which is the setting of the new residual value.

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 *d* times where *d* is the depth of the tree. For example, the lowest-level employee with the minimum influence value in the company will be visited up to *d* times since at each of his subordinates, if there is another employee whose percolated value is greater than this minimum, then the algorithm will recursively walk down the non-maximum branches of the subtree at each manager employee to "fix" the values such that the residual influence value calculated does not duplicate manager employee influence values. Since *d* is usually O(log n), this means that the lowest-level employee with

the minimum influence value may be visited at most *d* times, thus the time required to traverse the entire tree is O(n log n).

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 *k* are computed. According to the Python Wiki TimeComplexity page, this requires O(*k*) time to create a slice of the residualEffect array to perform the sum operation on. Then, a sum operation is performed on this slice which requires O(*k*) time as well.

The overall run time required by this algorithm is O(n + n log n + n log n + k + k). 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 (https://wiki.python.org/moin/TimeComplexity).

Pseudo Code Description

```
# id: the id of the employee to look at, value: value to pass
FixTree(id, value):
  Set residualEffect of this employee to value + this employee's value
  If employee with id == id has no subordinates:
    return (id, residualEffect[id], residualEffect[id])
  Let branchMax = residualEffect[id], and branchMaxId = id
  For each subordinate of employee with id == id:
    Call FixTree passing the subordinate id, and the current residualEffect
    Retrieve the subId, subEffect, and subBranchMax from the return value
    If the subBranchMax value > branchMax: Update the branchMaxId and branchMax values
  Set the residualEffect of this employee to 0 # better to influence a subordinate
  For all subordinates that are not the maximum of this employee:
    Call FixTree passing the subordinate id and a residualEffect of 0
  Return (id, residualEffect[id], branchMax)

# id: the id of the employee to look at
ScanTree(id):
  If employee with id == id has no subordinates:
    return (id, residualEffect[id], residualEffect[id])
  Let branchMax = residualEffect[id], and branchMaxId = id
  For each subordinate of employee with id == id:
    Call ScanTree passing the subordinate id
    Retrieve the subId, subEffect, and subBranchMax from the return value
    If the subBranchMax value > branchMax: Update the branchMaxId and branchMax values
  Set the residualEffect of this employee to 0 # better to influence a subordinate
  For all subordinates that are not the maximum of this employee:
    Call FixTree passing the subordinate id and a residualEffect of 0
  Return (id, residualEffect[id], branchMax)

Call scanTree(1, 0)
Sort the residualEffect array in increasing order
Print the sum of the first number to influence array entries.
```