Nick Alekhine
CS 4800
Programming Assignment
November 22, 2014

Mathematical Formulation

$find\ S \subseteq E,\ such\ that\ |S| = k,\ such\ that:$

$$max\ f(S)$$

$S = e_1 ... e_n$

$f(S) = \sum_{n \in P} val(n),\ where\ P = (path(e_1) \cup path(e_2)\ ...\ \cup path(e_n))$

High Level English Description

We start off by piping in data from a text file and create a list of employees, where the index of the list is the same as the id of the employee. Each employee keeps track of their id, boss id, utility, total utility, direct children, influence path, and highest child. From that list, a priority queue is built in the same manner as the company hierarchy. Once that is all set up, we go through all of the employees in the priority queue by popping the top off, getting the highest child, summing up the utilities in the influence path (if an employee hasn't been used already), and adding the summed utility to a result list. Then, all the employees in the influence path get set to False. We repeat this process until all employees have been chosen.

From there, we pick the k largest values in the result list, sum them up, and return the value.

Pseudo-code Description

```
sol( < textFile.in > textfile.out ):
- pipes in data from a text file
- reads data line by line, adding each line of data as an instance
  of an employee to a list of a employees.
  - *each employee has a:
    - unique id
```

        - boss id
        - utility
        - total utility
        - children
        - influence path
    - for every new employee added, set the influence path to be the boss plus the boss’ influence path. Also set the total utility to be the current employee’s total utility to be this utility + the boss’ total utility. Then, compare this employee’s total utility against the influence path’s highest child’s total utility (if the current total utility > the employee’s highest child, set the current as the employee’s highest child).
- a priority queue is built off of the list of employees (same structure as company hierarchy)
- while there are still employees in the priority queue:
    - pop the top
    - get the highest child
    - sum up the utility of everyone in the influence path of the highest child (if hasn't been used yet)
    - append result to a result list
    - mark the top, highest child, and everyone in the influence path as already used
    - repeat.
- pick the k highest values in the results list
- sum up these values. return the summed value

Proof of Correctness

Building the employees list and the employees priority queue is a trivial manner. For every line in the input file, an instance of an employee is created from that data and appended to an employees list. Once we have run out of lines, the loop terminates, returning a list of employees.

The same applies for the priority queue. We go through every employee in the employees list and append each one to the priority queue. This terminates once we have run out of employees to append.

The core of this algorithm is finding the summed utilities of the top k leaf nodes (leaf employees). We do this via a loop that pops the top off of the priority queue, asks that employee for its highest-valued child, and sums up all of the utilities from the employee to its highest-valued child. In order to not count duplicates, the employee list is referenced to make sure that anything in the path that is going to be summed has not been used yet. If it has, then we don’t include it in the summed utility for that path.

After the summation occurs, all of the employees in the path are set to False, denoting they have been used.

This process of popping the top, finding the highest-valued child, and summing the utility in the path is continued until all employees have been popped from the priority queue.

At this point, the k highest values are picked from the results list, summed up, and returned.

Running Time Analysis

Reading files from input, adding to employees list: **O(n)**
Creating priority queue from employees list: **O(n)**
Main loop of algorithm: **O(n*log(n))**
  summing up everyone in influence path: **O(log(n))**
  setting everyone in the influence path to False: **O(log(n))**
Get the k biggest values from RESULT **O(k)**

running time: **O(n*log(n))**