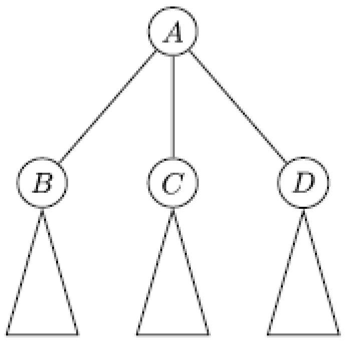
CS5206 : Fundamentals in Algorithms
Sample Homework Problem and Solution

R4. [Fun Company Party]

Your company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, this is, a tree with the company CEO at the root. The organizers of the party have assigned a real number to each employee measuring how "fun" the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the CEO of the company *must* attend the party, even though he has a negative fun rating; it's *his* company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the "fun" ratings of the guests.

Solution: We identify each employee with a node in the tree; the root is the president of the company. For every node x, let fun(x) denote the fun rating of the corresponding employee. For every node x in the tree, we compute the values $Y(x)$ and $N(x)$ which represent the optimum 'fun' value of a party where x is invited or not invited respectively.

Consider the following concrete example, which shows part of the company hierarchy. Let A be a node in the tree and B, C, and D be its children.



If A is invited to attend the optimal party, then none of its children can be invited. The optimum party must contain an optimal party just for B's employees (since otherwise the whole party could be made more fun). So, if we recursively compute the optimal parties for the hierarchies rooted at B, C, and D, then we can construct an optimal party for everyone. Thus,

$$Y(A) = \text{fun}(A) + N(B) + N(C) + N(D).$$

Likewise, if in the optimal solution A is *not* invited, then each of its children could either be invited or not. In either case, we must have the optimal party for the subtree rooted at that child. The optimal party must include the maximum of these two options for each child. Thus,

$$N(A) = \max\{Y(B), N(B)\} + \max\{Y(C), N(C)\} + \max\{Y(D), N(D)\}.$$

Our algorithm computes the solutions to the subproblems recursively. For a leaf in the tree (an employee with no subordinates), the optimal solution depends entirely on whether the 'fun' rating of that employee is positive or negative.

```
PARTY($u$):
    $Y(u) \leftarrow \text{fun}(u)$
    $N(u) \leftarrow 0$
    for all children $v$ of $u$
        PARTY($v$)        ⟪Compute $Y(v)$ and $N(v)$⟫
        $Y(u) \leftarrow Y(u) + N(v)$
        $N(u) \leftarrow N(u) + \max\{Y(v), N(v)\}$
```

Once we have computed the Y and N values for all the nodes, the optimum solution is $Y(\text{root})$, since the president must attend the party. We can get the corresponding guest list by also storing, for each node v, whether $Y(v)$ or $N(v)$ contributed to $N(\text{parent}(u))$, and performing a second tree traversal.

Time: For each internal node in the tree, we spend time proportional to the number of its children, *i.e.*, its degree. For each leaf, we only need constant time. Since the degree sum of the nodes in a tree with n nodes is $\Theta(n)$, the total running time of the algorithm is $\Theta(n)$.

Space: We only need to store the Y and N values for each node. Therefore, the space required is $\Theta(n)$.