Sathish Kumar Reddy Yenna
Madhusudhan Reddy Tera

PLANNING A COMPANY PARTY

Problem:

A company is planning a party. The company has hierarchical structure: the supervisor relation forms a tree rooted at the president of the company. The Personnel office has ranked each employee with a conviviality rating, which is a real number. We need to make a list of guests such that the sum of the conviviality ratings of the guests is maximized with the following condition: Both employee and his/her immediate supervisor shouldn't be allowed in the guest list.

Input: A Tree T with root t, and each node v is assigned a real value rating(v).

Output: List of nodes such that sum of the ratings of the nodes is maximum.

Approach:

We solve the problem stated above using the recursive approach.

Here according to the problem statement, when there is only one node in the given tree then we have only one rating and that is of the root. So, the maximum possible rating is the rating of the root.

Thus, PARTY (T)= rating (root) when T has no children.

When the given tree has more than one node, then we need to consider the restriction that a parent and its child can't be selected. So, we need to take a node and its grandchild and so on until we reach the leaves.

But, here when finding the maximum rating for the sub-tree, T we need to find the maximum of (∑PARTY (v) for all v ∈ child (T)) and (rating(root)+ (∑PARTY (v) for all v ∈ child(child(T))) and consider that one as the maximum possible rating for the sub-tree T. In the same way, we can find the maximum possible rating for the entire tree recursively by the following recurrence relation.

$$\text{PARTY(T)= MAX}\left\{\begin{array}{l}\sum PARTY(v), \forall v \in child(T), \\ (\text{rating (root)} + \sum PARTY(v), \forall v \in child(child(T))\end{array}\right\}$$

Thus by combining the two cases we get the following complete recurrence relation.

$$\text{PARTY (T)= } \left\{\begin{array}{l}\text{rating (root) when T has no children,} \\ \text{MAX}\left\{\begin{array}{l}\sum PARTY(v), \forall v \in child(T), \\ (\text{rating (root)} + \sum PARTY(v), \forall v \in child(child(T))\end{array}\right\}\end{array}\right\}$$

We can write the algorithm based on the above recurrence relation as follows.

Algorithm:

```
Procedure MAIN (tree T, root r)
Begin
```
/*Program to compute the above recurrence relation and print the nodes considered for getting the maximum rating of the given tree. Here we use the linked list L which is declared global for storing the nodes after computing the maximum rating of the sub-trees. Start and last are the pointers of the linked list pointing to the starting and ending elements in the linked list. */
```
        Max←PARTY(r);
        PRINT("The maximum rating possible is", Max);
        GENERATE(r);
        s←start;
        PRINT("The List of Nodes selected are");
        While(s!=NULL) do{
            PRINT(s→value);
        }
End MAIN;
Procedure PARTY(t)
Begin
```
/*function for calculating the maximum rating for the tree with root t. visited[1..n] is a boolean array keeps track whether a node is visited or not(That is whether its maximum possible rating is calculated or not). V[1..n] is an array for storing the maximum ratings of all sub-trees. */
```
        if visited[t]=True then return(V[t]);
        visited[t]←True;
        if(there are no childs(t)) then {
                        V[t]←rating[t];
                        Return(rating[t]);
        }
        sum1=rating(t);
        for all v ∈ child(child(t)) do {
                    if (visited[v]=True) then sum1=sum1+V[v];
                    else sum1=sum1+PARTY(v);
        }
        sum2=0;
        for all u ∈ child(t) do {
                   if (visited[u]=True) then sum2=sum2+V[u];
                   else sum2=sum2+PARTY(u);
        }
        if (sum1>sum2) then V[t]←sum1;
        else V[t]←sum2;
        return (MAX(sum1,sum2));
end PARTY
```

Procedure GENERATE(t)
Begin
/* Function to generate the list of nodes with maximum possible rating. The input is a root of the tree for which we need to find the list of nodes to be considered for getting the maximum rating and the array of maximum possible ratings for each node V[1..n]. This function adds the nodes to a global linked list recursively. */

```
sum1←rating(t);
for all v ∈ child(child(t)) do  sum1=sum1+V[v];
sum2←0;
for all u ∈ child(t) do   sum2=sum2+V[u];
if (V[t]=sum1) then {
        ADD(L,t);
        for all v ∈ child(child(t)) do  GENERATE(v);
}
else if (V[t]=sum2) then{
        for all u ∈ child(t) do   GENERATE(u);
}
```

end GENERATE;

Proof of Correctness:

The algorithm is solely based on the recurrence relation, so if we prove the correctness of the recurrence relation then we are done. We will prove it by Structural induction method.

Claim: The following Recurrence relation calculates the maximum possible rating for the tree T.

$$\text{PARTY (T)}= \left\{ \begin{array}{l} \text{rating (root) when T has no children,} \\ \text{MAX}\left\{ \begin{array}{l} \sum PARTY(v), \forall v \in child(T), \\ (\text{rating (root)} + \sum PARTY(v), \forall v \in child(child(T)) \end{array} \right\} \end{array} \right\}$$

Proof:

Basis: Consider the tree with a single node as the basis of induction, and then the maximum possible rating of the tree would be the rating of the root itself.

The recurrence relation shown above implies this. So, the recurrence relation is true for the basis.

Induction: Let T be a tree built by the inductive step of the definition, from the root node N and k smaller trees T_1, T_2, T_3,…,T_k as shown below in Fig-1. We may assume that PARTY(T_i) hold for i=1,2,…,k.

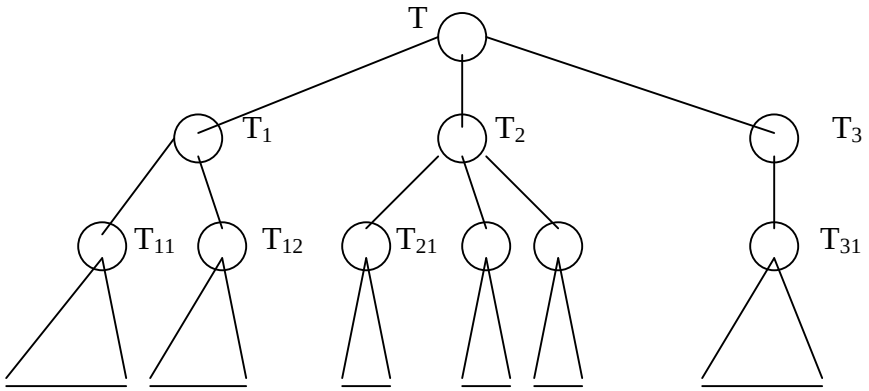

Fig-1

That is, PARTY(T_i) is maximum possible rating for the sub-tree T_i for all i=1…k.

$$\text{PARTY}(T_i) = \left\{ \begin{array}{l} \text{rating (root) when T}i \text{ has no children,} \\ \text{MAX}\left\{ \begin{array}{l} \sum PARTY(v), \forall v \in child(Ti), \\ (\text{rating (root)} + \sum PARTY(v), \forall v \in child(child(Ti)) \end{array} \right\} \end{array} \right\}$$

This assumption implies that the recurrence relation also holds for all the sub-trees of T_i. That is if T_{i1}, T_{i2}, T_{i3},…, T_{im} are the sub-trees of T_i ,then PARTY(T_{ij}) is the maximum possible rating for the sub-tree T_{ij} for all i=1,2,…,k and j=1,2,3,..,m.

Now, the maximum rating of the tree T can be found as follows.

Let V1 and V2 be as follows:

V1= ∑PARTY (v) for all v ∈ child (T_i)

V2= ∑PARTY (v) for all v ∈ child(child(T_i))

V1 is truly the summation of all the maximum ratings of the tree T_i because we assumed that PARTY(T_i) hold for all i=1,2,3,…,k

V2 is truly the summation of all the maximum ratings of the tree T_{ij} because the assumption implies that PARTY(T_{ij}) hold for all i=1,2,3,…,k and j=1,2,3,…,m.

The maximum rating of the entire tree T is given by the following formula.

Maximum Rating (T)= MAX (V1, rating(root)+V2)

Our recurrence relation exactly implies the above calculation, So it holds for the tree T also.

Thus, by Structural Induction the recurrence relation is correct for all trees.

Time Complexity:

The time complexity of the algorithm is the sum of the times taken for the two functions PARTY(root r) and GENERATE(root r).

Time complexity of PARTY(root r):

Since PARTY is called exactly once for each node in the graph. There are n calls for the procedure PARTY. The algorithm takes a time in O(n) for the procedure calls.

In all the calls for PARTY, the body of the first For Loop is at most executed 'n' times and hence it takes time in O(n). Similarly, in all the calls for PARTY, the body of the second For Loop is at most executed 'n' times and hence it also takes time in O(n).

Time Complexity T (PARTY (root)) $\in$ O(n + n + n)

Time complexity T (PARTY (root)) $\in$ O (n)

Time complexity of GENERATE (root r):

The function is called from the main function once and with root, r of the tree as the parameter. This function adds the nodes to the linked list recursively. This function is called at most 'n' times once for each node. So, the algorithm takes time in O (n) for the procedure calls.

According to the argument provided above the For loops in the GENERATE also take time in O(n).

Time Complexity T (GENERATE (root)) $\in$ O(n + n + n)

Time complexity T (GENERATE (root)) $\in$ O (n)

Time Complexity T(n) of the MAIN( ) function is given as follows:

T(n) = T(PARTY(root)) + T(GENERATE(root))
$\in$ O (n + n)
$\in$ O (n).

References: *Introduction to Algorithms*, Cormen, et al., McGraw Hill.