Report for Programming Problem 1

Team:
Student ID: 2019216753 Name: João Carlos Borges Silva
Student ID: 2019216826 Name: Pedro Afonso Ferreira Lopes Martins

1. Algorithm description

1.1 Sub-problem description:

The sub-problem can be described as a valid solution in which, starting at a given node we can find the optimal solution (minimal number of visited nodes that allows it to know the entire tree and maximizes the recruitment cost).

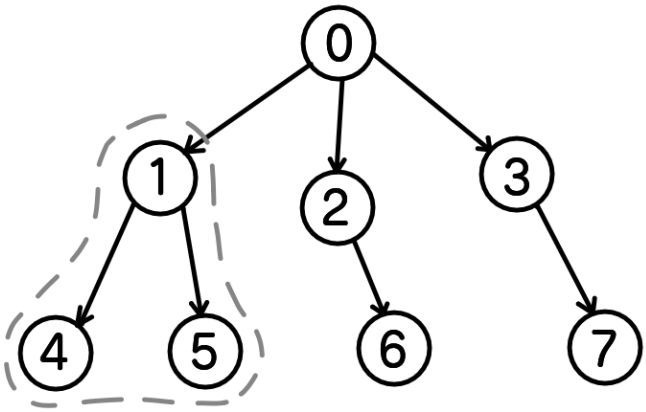

Fig. 1 – Marked sub-problem for tree starting at node 1

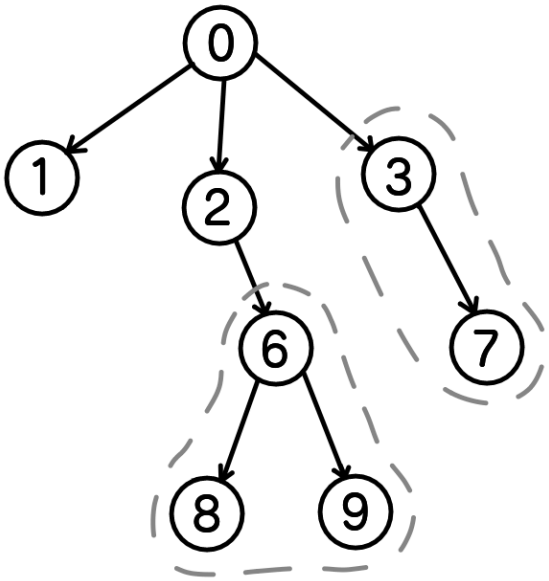

Fig. 2 – Marked sub-problems for trees starting at nodes 6 and 3

1.2 Algorithm approach:

Firstly, we have a superfluous node inserted before any other valid nodes. After this we proceed to iterate over all of the nodes inserted into the tree in order to discover which ones are leaves and fathers (node with at least one child). For each leaf we store its sub-solution which will be 1 for the minimal number of visited nodes and an integer that represents its cost of recruitment. After this step we compute the depth of each node and then we sort our nodes (only the fathers) by decreasing order of depth.

For the final piece of the algorithm, we iterate over the fathers in order to calculate the sub-solution for each one of them. For this we will also iterate over all of their children in order to find whether or not it's better to resort to its children or grandchildren. This consists in the minimal number of visited nodes that validates the solution and the maximum cost of recruitment. The solution for the entire problem will be located at the superfluous node described above.

1.3 Speed-up tricks:

Depth calculus: In order to calculate the depth of which node we proceed in a recursive step. Before that we used to proceed with a slower approach in which each depth is computed in an iterative fashion. This in turn sped up the main portion of our algorithm, reducing the time complexity of this operation.

2. Data Structures

Struct node: This struct represents each member of the Pyramid Scheme. Each one is represented by its number of children (**integer**), cost of recruitment (**integer**), depth in the given tree (**integer**), a **vector of integers** which contains all of the member recruited by him and finally an **array of integers** (with size 2) that represents the sub-solution.

Nodes (indexation): Represented by an **unordered map** which encapsulates all of the nodes (**struct node**).

3. Correctness

Optimal Substructure

Given a tree T = $\{N_0, N_1, \ldots, N_n\}$, let PS($N_k$) be the optimal solution for the tree that start at N_k. Then if N_k is removed from T then PS(N_{k_j}) , 1 < j < nº of children of N_k , will be the optimal solutions for the subproblems of the tree that starts at N_{k_j}.

Assume that PS(N_k) is the optimal solution for the tree that starts at N_k, whilst PS(N_{k_j}) is the optimal solution for one of the children of N_k and is used in the solution of PS(N_k).

PS(N_{k_j}) is not the optimal solution for the given subproblem.

Then PS'(N_{k_j}) would be the optimal solution, where both the number of nodes is smallest and the cost of recruitment is bigger than that of PS(N_{k_j}).

But when we merge all of nodes, we conclude that PS(N_k) no longer is the optimal solution for the tree that start at N_k. Which leads to a contradiction of the initial assumption. Therefore PS(N_{k_j}) is the optimal solution for the given subproblem.

4. Algorithm analysis

Time Complexity:

In order to calculate the time complexity of an algorithm we must first take a look at the worst case, which correlates to the scenario where all the father nodes have 10 children (maximum number of children possible for each node). We transverse each father once, as well as each child and grandchild once.

The calculus of the time complexity for the algorithm is as follows,

$T(n) = 10 * 10 * n + \mathrm{n} * \log(\mathrm{n}) + c$ **Time complexity:** O(N*log(N))

Each father node corresponds to n, the first 10 corresponds to the maximum number of children and the second 10 equates to the maximum number of children each child can have. The N*log(N) represents the sort operation. Finally, T(n) represents the total temporal cost and c is the constant time of the call.

Spatial Complexity:

In order to calculate the spatial complexity of our algorithm we will need to compute the space necessary to store the required information. This will give us $(3 + 2 + 10) * \mathrm{N}$, which is the same as $15\mathrm{N}$ required space for the all the tree. The $3 + 2 + 10$ corresponds to the size of struct node, where 3 represents 3 integers (number of children, depth, cost of recruitment), 2 to another 2 integers (the optimal solution for the node) and 10 to the maximum number of children each node can have.
This will in turn give us a spatial complexity of,

$S(n) = (3 + 2 + 10) * n + c$ **Spatial complexity:** O(N)