Assignment – 4: Divide and Conquer

Submitted by – Farhan Ishmam, 180041120, CSE – A.

Date – 06-11-2020

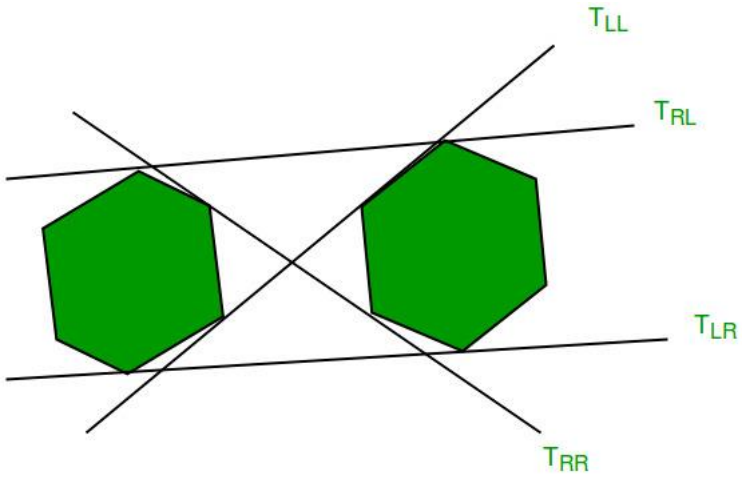
## Problem – 1

Md Atiqur Rahman – 180041123

Syed Rifat Ryan - 180041205

## Problem – 2

As per question, $a_1$ and b1 are the points with maximum x for the set of points CH(A) and CH(B). By interconnecting the points of CH(A) and CH(B), we get a number of lines. Among them, the lines that pass through two points only are called tangents. The uppermost line among the tangents is called upper tangent. We need to prove that the points $(a_i,b_i)$ will be upper tangent if and only if the value of y for these points are maximized.




According to the figure, we can easily say, $T_{RL}$ is the upper tangent. To prove the value of Y co-ordinate is maximized in this case, we can prove using contradiction.

Suppose, there we take a tangent where the value of y isn't maximum for both the points. $T_{LL}$ and $T_{RR}$ is an example of such a tangent. In both cases, there is another value of y which will be greater than the values that form these two tangents. By the definition of upper tangent, we know, the upper tangent must be above every other tangent. So, the points connecting the upper tangent must be above every other point i.e. must have a greater value in y-axis. For $T_{LL}$ and $T_{RR}$, clearly, there is one point which doesn't have the highest value in y-axis. So, both cannot be upper tangent as there exists another line which is above both of them. Hence, there can be no point where the y co-ordinate isn't maximum and stay as an upper tangent at the same time. Hence, for the line connecting (ai, bj), it will be the upper tangent if and only if the values at y co-ordinate i.e. y(I,j) is maximized.

## Problem – 3(a):

A counter example where the greedy algorithm will not work is given below:


Here the summation of children of adjacent nodes is greater (2 + 3 > 4) and thus the given algorithm will fail.

## Problem – 3(b):

We will use dynamic programming to solve the problem.

**Sub-problem**: We assume two arrays dp1(u) and dp2(u) for our subtrees indicating the maximum profit whose root is u, with and without the root u respectively.

**Relate:** If v is not the leaf node,

$Dp1(v) = p_v + sum(dp2(adj[v][i])$ where {I = 0 to v.size(1) – 1}

Dp2(v) = max(dp1(adj[v][i],dp2(adj[v][i]) where {I = 0 to v.size(1) – 1}

**Base:** $dp1(v) = p_v$

Dp2(v) = 0

**Solution:** The topologically sorted nodes are traversed using a simple dfs. The nodes are to be traversed in reverse order so that we start with the leaf nodes.

The final answer max(dp1(x), dp2(x)) is returned.

**Runtime:** As DFS is performed, the complexity is O(V+E).

Since, the graph is acyclic and undirected, it forms a tree.
For a tree, E = V-1

So, O(V+V-1) = O(2V-1) = O(V).

So, our graph has **liner time complexity** in worst case.

## Problem – 3(c):

**The proposed algorithm** is given below:

- Just like our previous problem, we will traverse the graph in reverse order DFS order. So, the starting nodes will be leaf nodes.
- For each node, we insert V to the set A along with deleting the parent of V if it is present in A.
- Return the set of vertices A as the result of the problem

**Correctness of the algorithm:**

Given that all the nodes have equal profit margin. According to the order of traversal, we are starting with leaf nodes and deleting their parents as we put them in our set A. This ensures that the last node of the set is a leaf node because of the order we traversed and inserted in set A.

**Runtime:**

As DFS is performed, we have O(V+E) time complexity. The complexity of insertion and deletion doesn't add up to this complexity and the end-complexity.

Since, this is a tree, E = V – 1

So, O(V + V -1) = O(2V) = O(V)

Our algorithm has **linear time complexity**.

## Problem – 3(d):

As we now have a cycle in our graph, we can't use the dynamic approach in solving this problem any longer. To do so, our dependency graph must be acyclic.

Now there are multiple ways to solve this problem. As no efficiency has been mentioned a simple algorithm could be –

- Iterate over all the possible subsets of node V
- For each subset find the calculate the profit by summing
- Then find the maximum among all the calculated sums and return it

The simple **brute force approach** is however very inefficient with **exponential time complexity**.

Another approach can be to transform the acyclic graph to a cyclic one by making multiple copies of the graph. This methodology was followed to solve shortest path with cycles using dynamic programming.

The algorithm is quite self-explanatory. We make 'k' copies of the graph and traverse the k+1 th in our relations. Here, we add an extra parameter k along with the i in x(i) function from the algorithm in 3(b). So, the new relation x(i,k) will traverse to the (k+1)th copy at every iteration. By doing the cycles will be avoided since for every edge a new node is visited.

We will make V-1 extra copies of the graph where V is the number of vertices. So, with the original graph there will be V copies in total. The **runtime** for this algorithm is however, **polynomial**.