Q1. DP on Trees #1

What do you understand from the term dp on trees?
It is a Technique that :

Can make use of simple tree traversals (like inorder) and memory to solve a problem.

Usually works by rooting the given tree at a given node.

All of the above.

Helps evaluate recursive functions defined over nodes of a tree efficiently.

Q2. DP on Trees #2

Using Dynamic Programming on Trees, what is the time complexity to find the depth of a tree with N nodes.

O(logN)

O(N^2)

DP cannot be used to calculate the depth

O(N)

Q3. DP on Trees #3

Given a tree T with N nodes, where each node i (1 to N) has Ci coins attached with it. You have to choose a subset of nodes such that no two adjacent nodes(i.e. nodes connected directly by an edge) are chosen and sum of coins attached with nodes in chosen subset is maximum.

Consider the brute force algorithm:

Root the tree at node number 1

```
check for BASE CASE //current node has no child
if(current node's parent was selected)  // root's parent is not
selected
    recur for children
    return sum of values returned by children.
else
    choice 1 = select this node and recur for children
    choice 2 = don't select this node and recur for children
    return max of choice 1 and choice 2.
```

what is the time complexity of this algorithm?

O(N^3)

O(2^N)

O(N * 2^N)

O(3^N)

Q4. DP on Trees #4

Given a tree T with N nodes, where each node i (1 to N) has Ci coins attached with it. You have to choose a subset of nodes such that no two adjacent nodes(i.e. nodes connected directly by an edge) are chosen and sum of coins attached with nodes in chosen subset is maximum.

Since at every node we had two options either to include or not to include the node.

Let us define:

a) sum[x][0] which represents max possible value for the subtree rooted at node x given that we do not select node x.

b) sum[x][1] which represents max possible value for the subtree rooted at node x given that we select node x.

```
R1) sum[x][0] = summation of [max(sum[k][0], sum[k][1])] for all k,
where k is child of x.

R2) sum[x][1] = value of node x + summation of [max(sum[k][0],
sum[k][1])] for all k, where k is child of x.

R3) sum[x][0] = summation of sum[k][0] for all k, where k is child of
x.

R4) sum[x][1] = value of node x + summation of sum[k][0] for all k,
where k is child of x.
```

Choose the correct option:

R1 and R3 are correct

R2 and R3 are correct

R2 and R4 are correct

R1 and R4 are correct

Q5. DP on Trees #5

Given a tree T with N nodes, where each node i (1 to N) has Ci coins attached with it. You have to choose a subset of nodes such that no two adjacent nodes(i.e. nodes connected directly by an edge) are chosen and sum of coins attached with nodes in chosen subset is maximum.

What is the time complexity of the above algorithm using the **sum[N][2]** matrix and performing a top-down dynamic programming solution to calculate **sum[1][0]** and **sum[1][1]** where node number 1 is the root of the tree.

Select the most appropriate option:

None of the above.

ans is max(sum[1][0], sum[1][1])

Time complexity is O(N)

Both ans option and time complexity option are correct

Q6. DP on Trees #6

Given a tree T with N nodes, calculate the length of the longest path between any two nodes(also known as diameter of tree).

Algorithm:

```
    Pick any node x of the tree.
    Perform a BFS from x and find the farthest node from x call it u.
    Perform a BFS from u to reach the node farthest from u call it v.
    The path length between u and v is the length of the diameter of
the tree.
```

Choose the correct option:

The algorithm is correct and runs in O(NlogN) time.

The algorithm is incorrect but there exists a non-DP algo to solve this problem.

The algorithm is correct and runs in O(N) time

It is not possible to solve this problem efficiently without DP

Q7. DP on Trees #7

Given a tree T with N nodes, calculate the length of the longest path between any two nodes(also known as diameter of tree).

It is possible to come up with a DP solution to the problem by :

Rooting the tree at a particular node, say node number 1.

Defining a function D(X) which represents the diameter of the subtree rooted at node X.

Deriving a recurrence that evaluates D(X) using the children subtree diameters and the depth of the children subtrees.

All of the above.

Q8. DP on Trees #8

Given a tree T with N nodes, where each node i (1 to N) has Ci coins attached with it. You have to choose a subset of nodes such that no two adjacent nodes(i.e. nodes connected directly by an edge) are chosen and sum of coins attached with nodes in chosen subset is maximum.

Let us define the following terms :

```
    D(X) : Diameter of the subtree rooted at X.
    DC(X) : Maximum of D(c1), D(c2), ..., D(ck) where c1,c2, ..., ck
are children of node X.
    h1(X) : depth of a subtree rooted at a child of X and has the
maximum depth among the children of X.
    h2(X) : depth of a subtree rooted at a child of X and has the 2nd
maximum depth among the children of X.
```

If X has no children h1(X) = h2(X) = 0.
If X has only 1 child h2(X) = 0.

choose the correct recurrence :

D(X) = h1(X) + h2(X)

D(X) = max(DC(X) + 1, DC(X) + h1(X))

D(X) = max(DC(X), 1 + h1(X) + h2(X))

D(X) = summation of DC(ci) where ci is a child of X.

Q9. DP on Trees #9

Given a tree T with N nodes, where each node i (1 to N) has Ci coins attached with it. You have to choose a subset of nodes such that no two adjacent nodes(i.e. nodes connected directly by an edge) are chosen and sum of coins attached with nodes in chosen subset is maximum.

The DP algorithm described previously requires to find h1(X), h2(X) and DC(X) before calculating D(X). what is the best possible time complexity in which it can be implemented also compare it to the BFS algorithm to find the diameter of the tree by choosing the correct option.

Time Complexity :

1.N

2.N^2

3.root N

4.NlogN

Comparison :

```
1. BFS algorithm is faster than DP algorithm

2. DP algorithm is faster than BFS algorithm

3. Both algorithms run in equal asymptotic time but correctness of DP
algorithm is easier to prove.

4. Both algorithms run in equal asymptotic time but the correctness of
BFS algorithm is easier to prove.
```

Choose the correct time complexity option and the correct comparison respectively

3 and 2

1 and 3

4 and 4

1 and 2

Q10. DP on Trees #10

Consider a segment tree T used for range minimum query on an array. Choose the most appropriate option, regarding the build function of the tree:

the dp on tree recurrence is tree[i] = min(tree[left(i)], tree[right(i)]) where left(i) and right(i) are children of node i.

the build function runs in O(N) and does not use the dp on tree technique.

the build function runs in O(N) and uses the dp on tree technique.

both B and C.

Q11. DP on Trees #11

In reference to famtree codechef dp on tree video

Consider the algorithm:

```
For every node find the smallest and the largest value in the subtree
of the node using a BFS.
Carefully update the ANS at each node if needed.
```

What is the time Complexity of this algorithm?

O(N) as there are only N nodes in the tree

O(N^2) because BFS is a slow tree Traversal technique

O(N) but only if we had used DFS instead of a BFS at every node.

O(N^2) as we perform a BFS at every node.

Q12. DP on Trees #12

In reference to famtree codechef dp on tree video.

Let us come with a different algorithm to solve the problem.

```
Declare a Global Variable ANS
choose a node of the tree as root
Perform a dfs from the root with two additional parameters LARGE and
SMALL.
(LARGE and SMALL represent the LAGREST and the SMALLEST node values
seen in the path from root to current node).
Whenever We visit a Node in the DFS we Do
ANS = max(ANS,abs(value of Node - LARGE), abs(value of Node - SMALL)

Once the DFS is complete we output ANS.
```

NOTE : At the root LARGE = value of root and SMALL = value of root

Choose the most appropriate option:

The algorithm is incorrect.

The algorithm is correct and is more efficient than the algorithm discussed in the video.

The algorithm is correct and equally efficient.

The algorithm is correct and is less efficient than the algorithm discussed in the video.

Q13. DP on Trees #13

In reference to xor circuit codechef dp on tree

What is optimal time complexity to solve the problem?

O(4*N) as each node may be assigned either 0,1,2 or 3 i.e. 4 different values

O(N^2) because we use dp on tree and work done to calculate result at each node is O(N) on average

O(logN) as height of balanced tree is order logN

O(N) because we use dp on tree and work done to calculate result at each node is O(1) on average

Q14. DP on Trees #14

In reference to xor circuit codechef dp on tree.

Let us define the following :

```
    ways(u,0) : number of ways to assign values to nodes in path from
node u to node N so that their xor is 0.
    ways(u,1) : number of ways to assign values to nodes in path from
node u to node N so that their xor is 1.
    ways(u,2) : number of ways to assign values to nodes in path from
node u to node N so that their xor is 2.
    ways(u,3) : number of ways to assign values to nodes in path from
node u to node N so that their xor is 3.

ways(N,0) = ways(N,1) = ways(N,2) = ways(N,3) = 1.
```

Choose the correct recurrence:

ways(u,0) = product of [sum of ways(k, i) over all k where k is child of u] for i goes from 0 to 3.

ways(u,0) = sum of [product of ways(k, i) over all k where k is child of u] for i goes from 0 to 3.

ways(u,0) = sum of [product of ways(k, i) over all k where k is child of u] for i goes from 1 to 3.

None of the above is correct.

Q15. DP on Trees #15

In reference to xor circuit codechef dp on tree

If the question is modified and now the nodes may be assigned any value from 0 to K, instead of 0 to 3 (K being an additional input). What is the optimal time complexity to solve this problem?

It will still be O(N).

The problem cannot be solved in polynomial time after this modification.

It will become O(N*K)

It will become O(N*K*K)