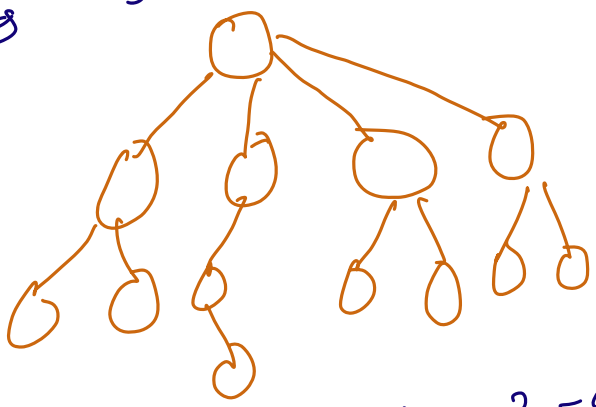
DP - 6 (Trees)

When answer of an entire tree can be solved using answers of subtrees, then it can be considered as DP on Trees

Base Case :

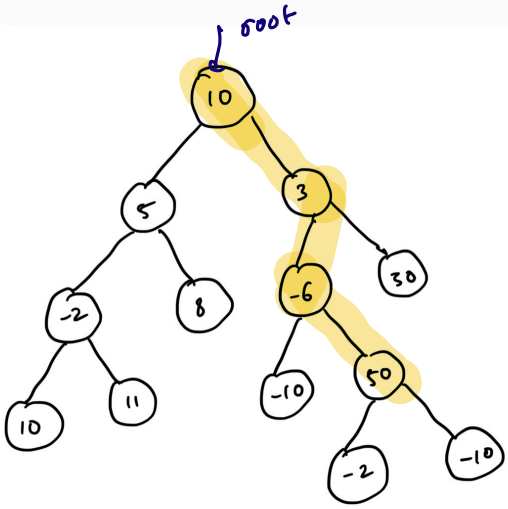
Leaf of K-ary Trees ⇒ root == null would be base case

Any node can have almost K children



Binary Trees ⇒ 2-ary Trees

Question : Given a Binary tree, find the max sum path from root to any node of its subtree

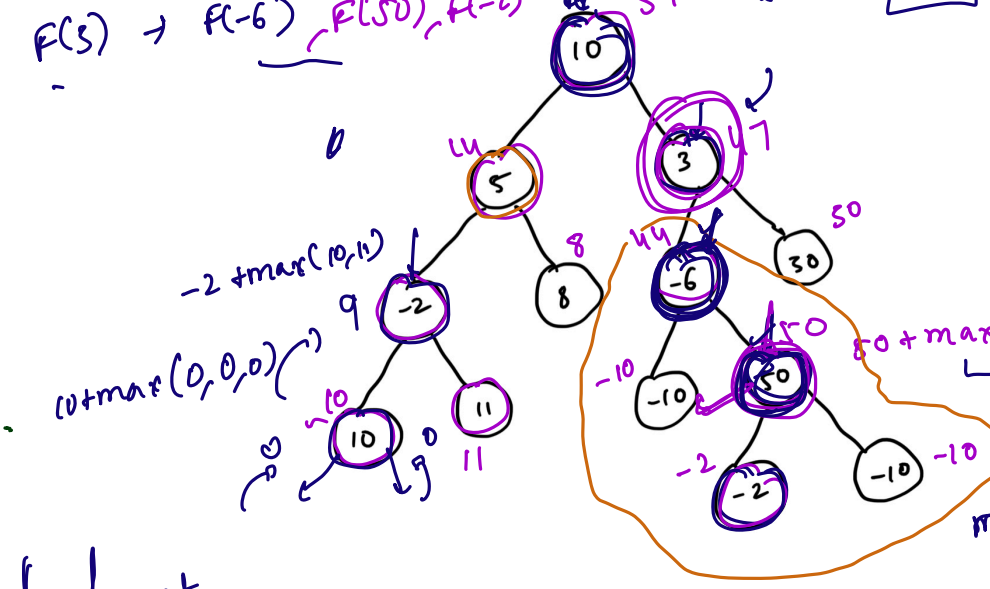


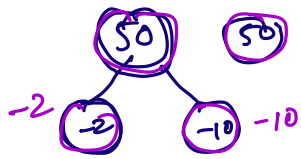
10 + 3 - 6 + 50 = 57

⇒ leafs can -ve

F(10) → F(5), F(-6), F(50), F(-2), F(-2) ⇒ DP

F(5) → F(-6), F(50), F(-2)





50 + max(-2, -10)

= 50 - 2 = 48

Postorder → 10

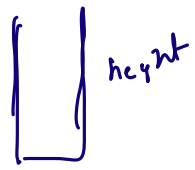
-2 + max(10, 11)

10 + max(0, 0, 0)

50 + max(-2, -10)

50 + max(-2, -10, 0)

node.val + max(left, right, 0)



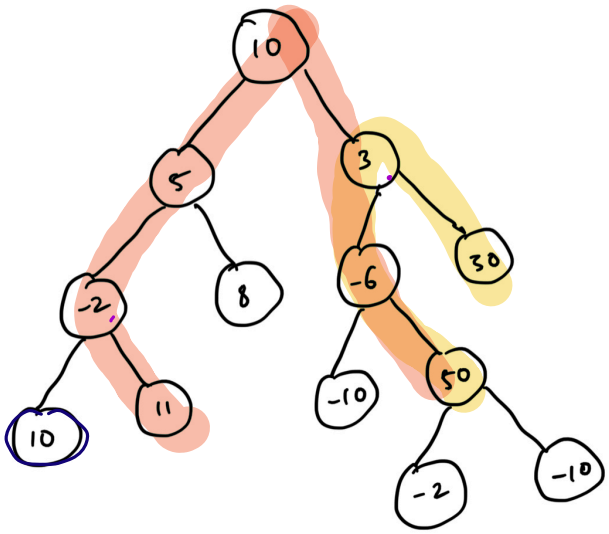
T.C : O(n)

S.C : O(H) → O(n)

```
int pathSum(Node* root){
    if(root==NULL) return 0;

    left = pathSum(root.left);
    right = pathSum(root.right);

    return root.val + max(left, right, 0);
}
```

Recursion → DP

Question: Maximum Sum path from any node to any node



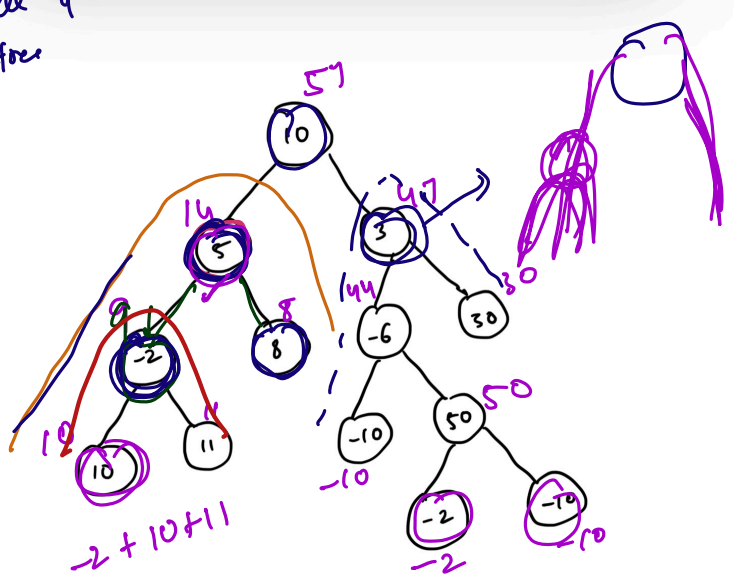
50 - 6 + 3 + 30 = 77

77 - 6 = ?!

1) Does the max sum path always end at leaf : NO

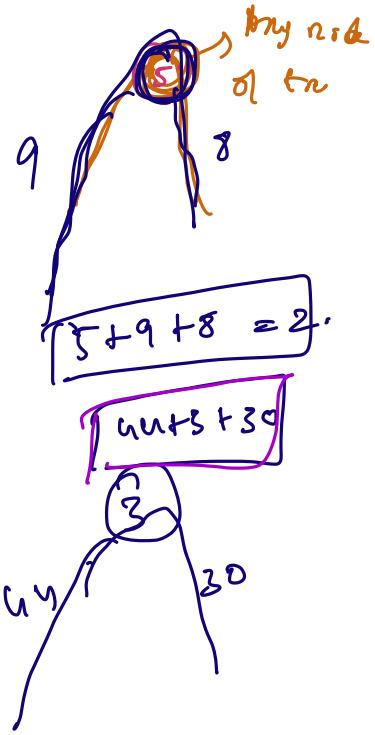
2) Does the max sum path always pass through root of tree : NO

9 + 8 + node val



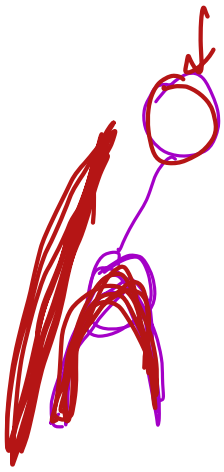
-2 + 10 + 11

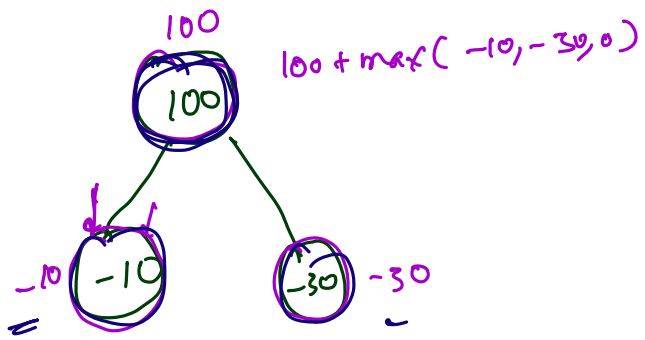
-2



ans = -10

-10 + 0 + 0 = -10





100 + (-0) + (-30)

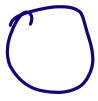
0 + 0 = -30

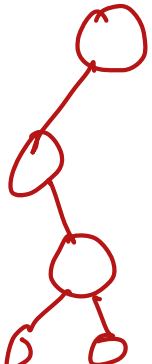
ans = max(ans, node.val + max(left, 0) + max(right, 0));

```
int ans = INT_MIN;
int pathSum(Node* root) {
    if(root == NULL) return 0;

    left = pathSum(root.left);
    right = pathSum(root.right);
    ans = max(ans, root.val + max(left,0) + max(right,0));
    return root.val + max(left, right, 0);
}
```

Max Sum path ⇒ ans

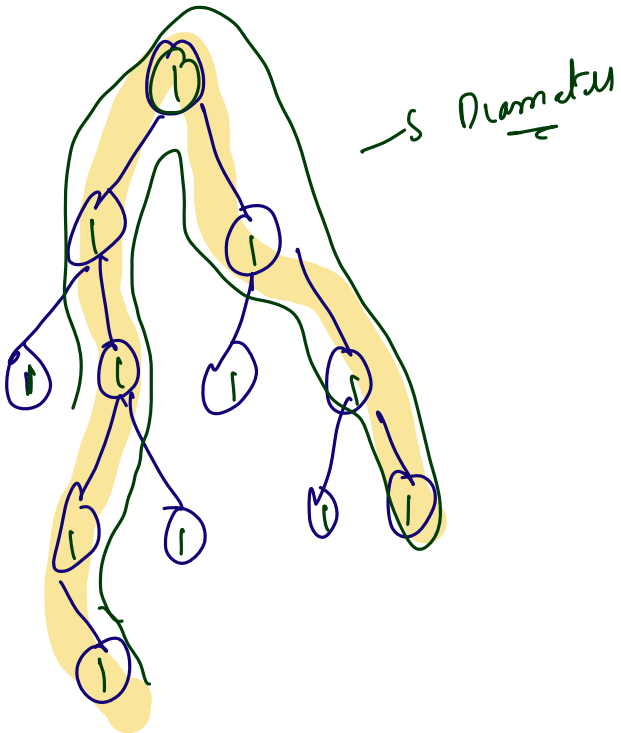
T.C: O(N)
S.C: O(H) → O(N)





Question: Diameter of Tree

Length of longest path between any 2 nodes of the tree

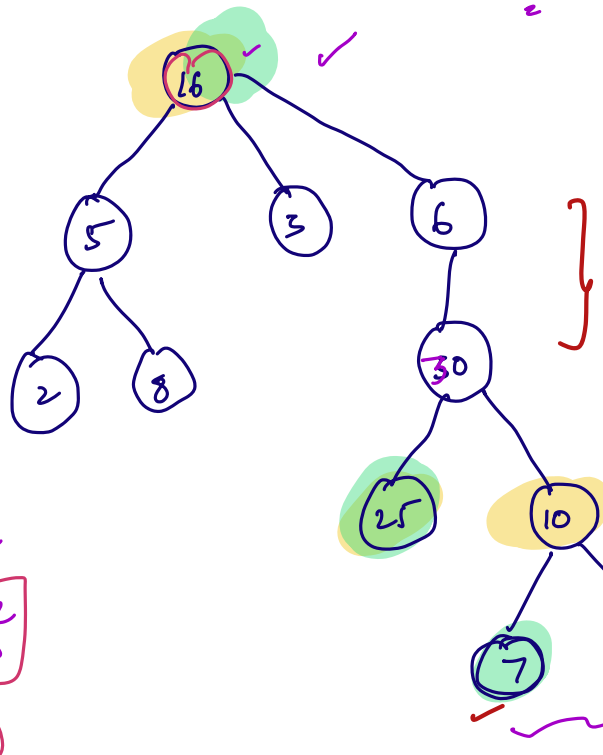


Question: Valuable nodes of k-ary Tree

→ N nodes in the tree, each node has val

→ Select some nodes such that sum of values of selected nodes is maximum

→ If a node is selected, then its child and grandchild cannot be selected



16+25+10 = 51

50+7+25+3+8+2

50+7+25+3+8+2

13

16+25+7+50 = 98

Simple

A =

0	1	2	3	4	5
10	5	-3	-8	-6	100

(i-3)th

(i-2) (i-1)

i

100 + maxSum(i-3)

maxSum(i-1)

choose numbers to maximise sum

Constraints

i) If i^{th} ele is selected, then $(i-1)^{th}$ & $(i-2)^{th}$ ele cannot be selected

Choices for ith element:

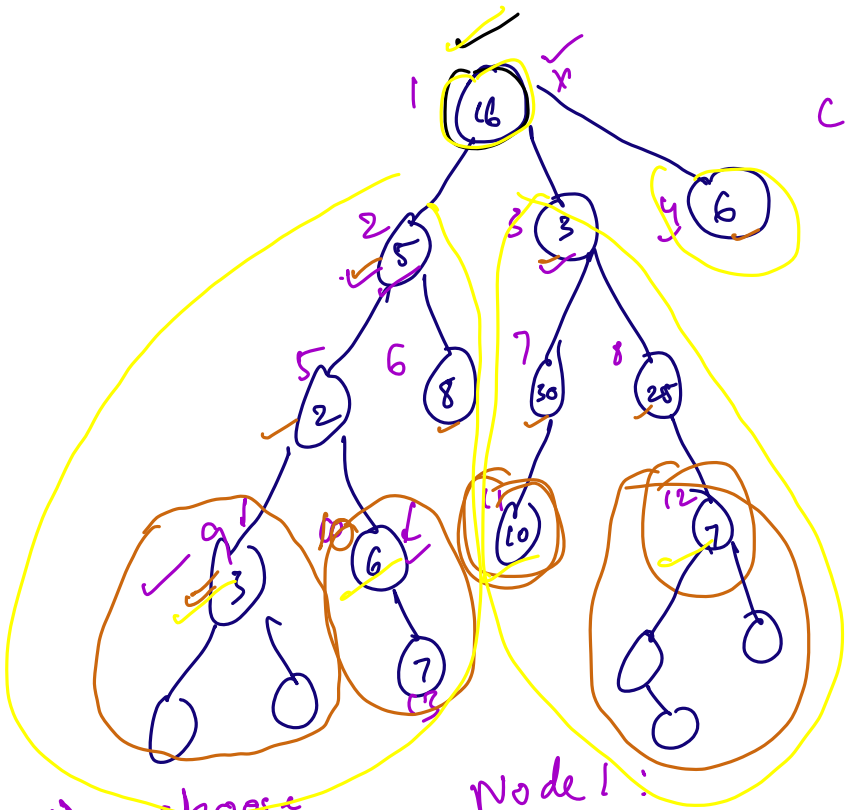
1) Choose the ith : A[i] + maxSum(i-3)

2) Don't choose the ith elemt : maxSum(i-1)

Can we use this technique for our problem.

C1: 16 + maxV(a) + maxV(10) + maxV(11) + maxV(12)

C2: max(2) + max(3) + max(4)



1) Choose Node 1:

16 + maxval(a) + maxval(10 + maxval(11) + maxval(12)

2) Dont' choose Node 1:

1) Case 1 : Choose root

root.val + sum (maxval (great grand Children))

2) Case 2 : Don't chose

sum (maxval (Children))

Tree Representation :

Parent array

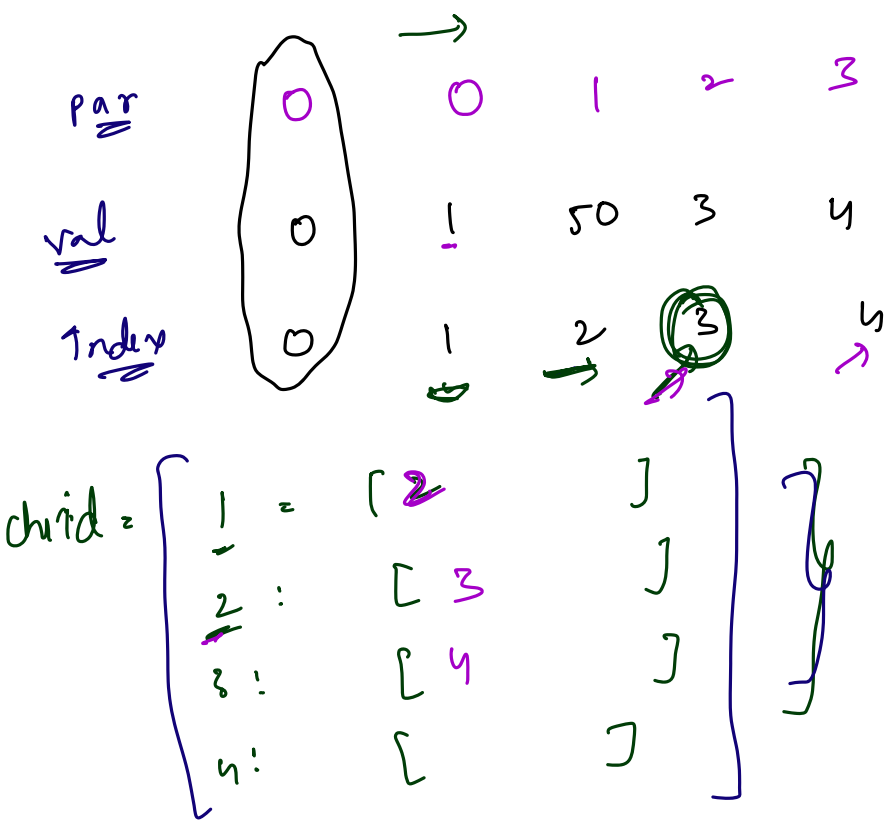
Nodes 1 ..... N

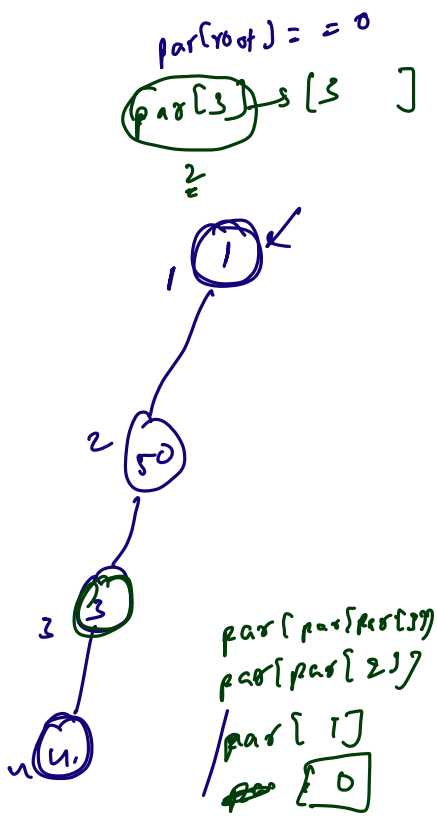
par[i] → Parent of ith node

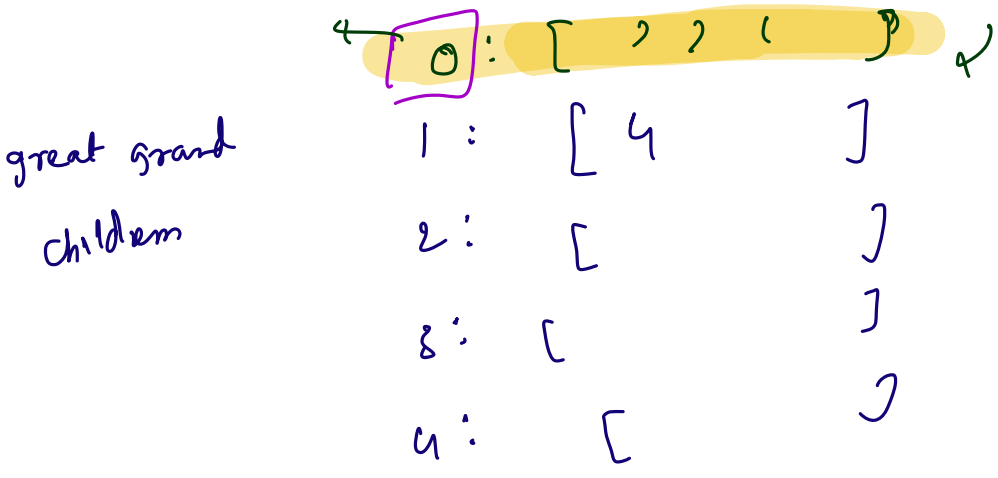
1- indexed Array

N = 4

(par[i] == 0) i is the root







parent : par[i]

grand parent : par[par[i]]

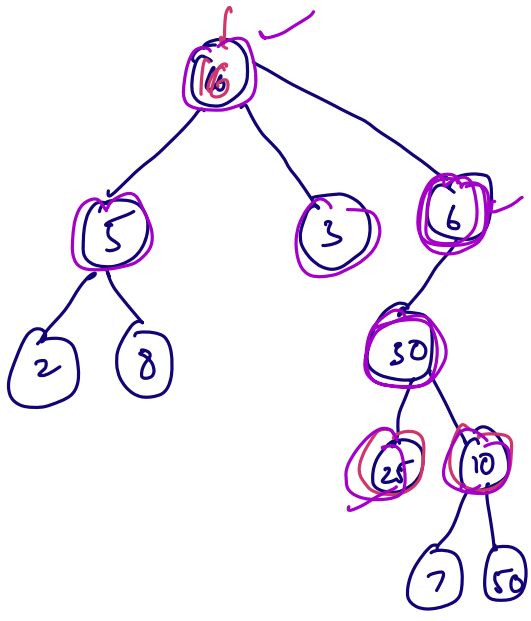
great grand parent : par[par[par[i]]]

```
for(i=1; i<=N; i++) {
    parent = par[i];
    child[parent].push(i);

    ggp = par[par[par[i]]];
    ggg_array[ggp].push(i);
}
```

Pseudode :

```
int maxVal ( int node ) {
    if( dp[node] != -1) return dp[node];

    // choice-1
    val1 = value[node];
    for(i=0; i < gg c[node].size(); i++) {
        val1 += maxVal (gg c[node][i]);
    }
    // choice-2
    val2 = 0
    for(i=0; i < child[node].size(); i++) {
        val2 += max ( child[node][i]);
    }
    dp[node] = max(val1, val2);
    return dp[node];
}
```

E(25), F(10) ✓

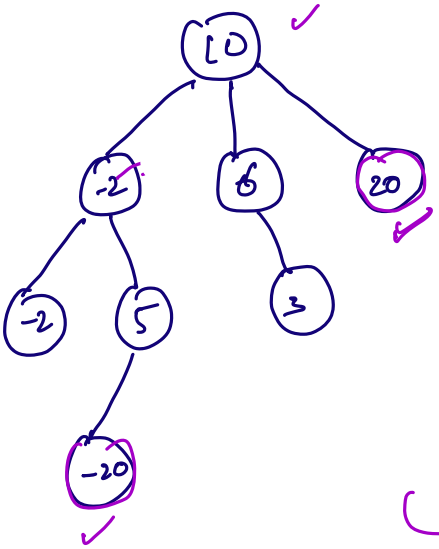


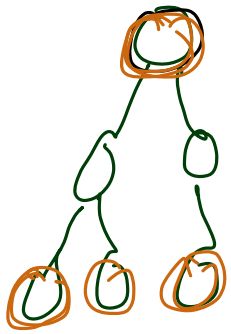
Question:

→ Given a k-ary Tree, each node has a weight.

→ select any 2 nodes such that one node is descendant of other node

→ choose such that absolute difference on weights is maximised.





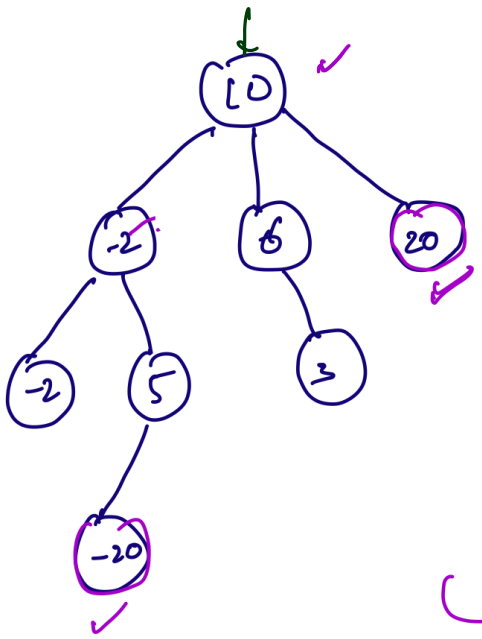
(10, -20)

20 - (-20) = 30

(20, -20)

(100, -20) ⇒ 120

→ Choose 2 values (a, b) such that |a - b| is maximised.



|a-b| is maximum.

a is the root value

Case 1: a ≥ b

|a-b| = a - b

min of all the subtree

Case 2: b ≥ a

|a-b| = b - a

max of all the subtree — fixed

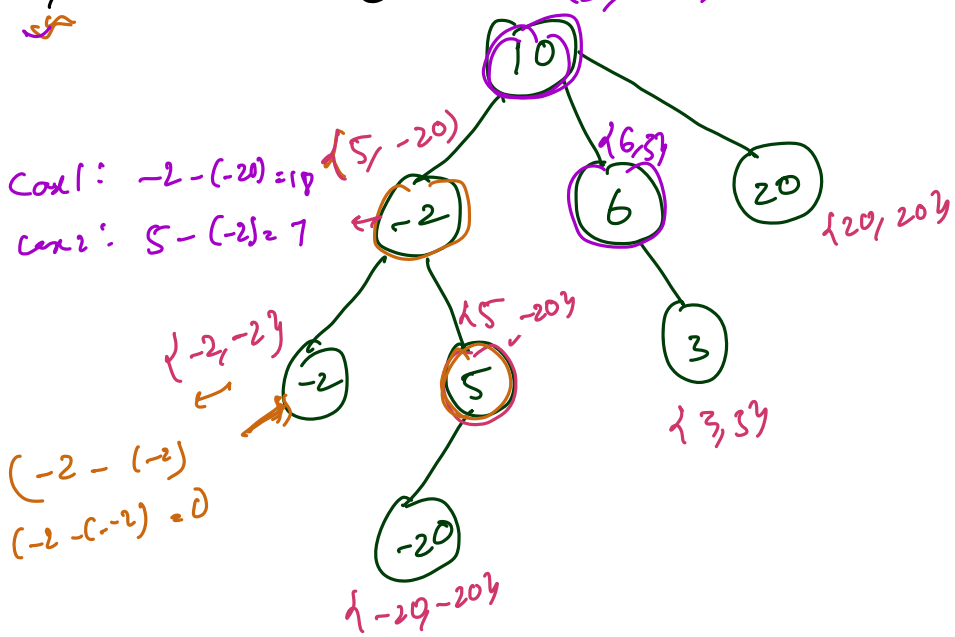
→ Every subtree should return the max and min of its subtree to the parent

ans = ~~0~~ ~~25~~ 30

C1: 10 - (-20) = 30

C2: 20 - 10 = 10

{20, -20}



Case 1: -2 - (-20) = 18 {5, -20}

Case 2: 5 - (-2) = 7

{6,6}

{20, 20}

{-2, -2}

{5, -20}

{3, 3}

{-20, -20}

Case 1: (-2 - (-2)

Case 2: (-2 - (-2) = 0

max(-2, 5, -2)

min(-2, -20, -2)

max(5, 6, 20, 10)

min(-20, 3, 20, 1)

child[]

max(-20, 5)

min(-20, 5)

Case 1: 5 - (-20) = 25

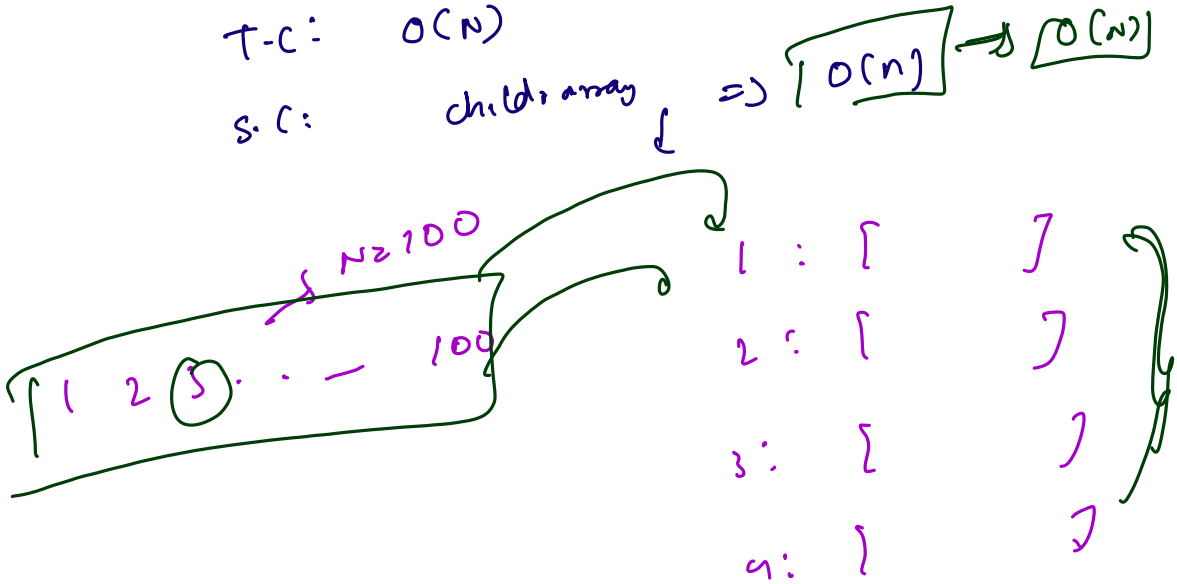
5 - 5 = 20

Case 1: ans = max(node.val - min(subtree))

Case 2: ans = max(max(subtree) - node.val)

T.C: O(N)

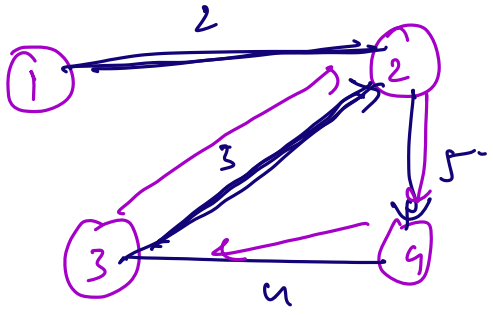
S.C: child array => [O(n)] → O(N)



Question:

Intro to Graphy

→ Set of vertices and edges

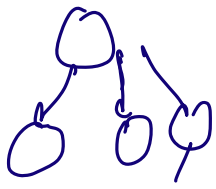


1) undirected
2) Directions

→ Weighted / Unweighted

→ Trees:

N nodes → N-1 edges

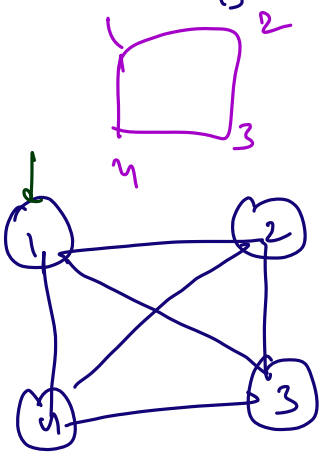


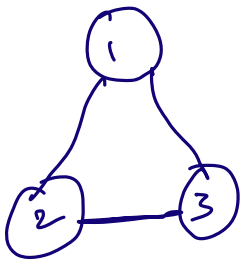
→ Given a tree, convert it into a fully connected graph.

↳ There is an edge between every pair of Node

↳ No. of steps (edges) to reach any node is 1

Undirected edge





→ Given a tree, we have convert to a fully connected graph

→ At T=1, we can add an edge u–w if there is an edge from u–v and v–w

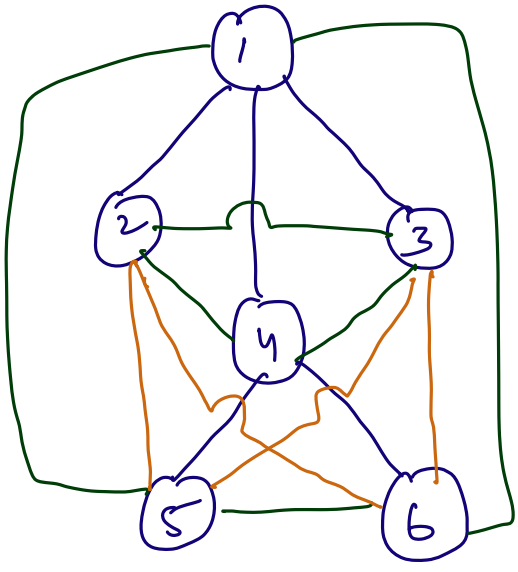
u – w only if u–v & v–w exist

2–1 & 1–3

3–4

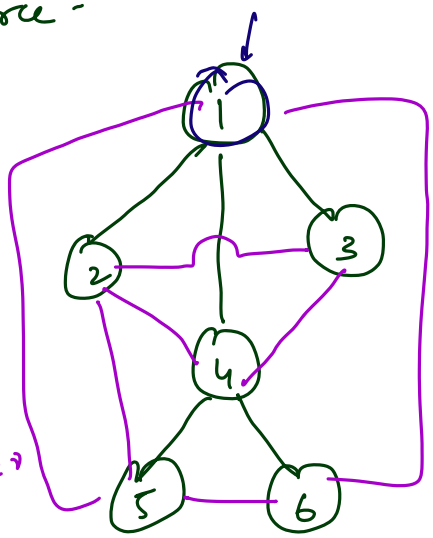
T=1

T=2



Min Time to make the graph fully Connected

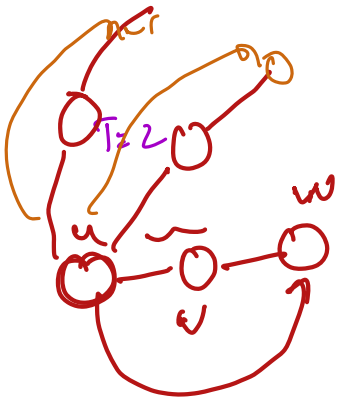
Brute Force :



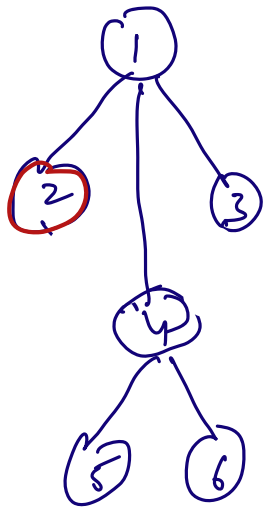
T = 1

T = 2

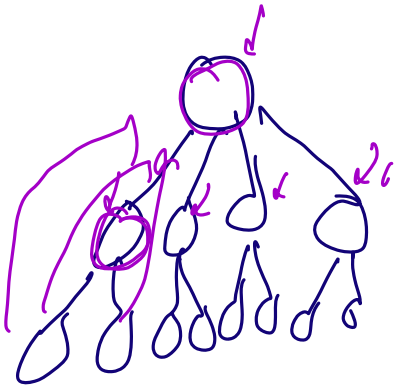
K is the answer



After T=1



2-(x) (a-5.



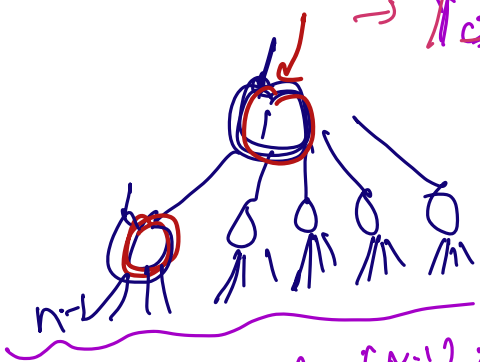
T-C :

child : neighbour

upper bound : $O(K \times n \times n)$

$\Rightarrow \boxed{O(K \cdot n^2)}$

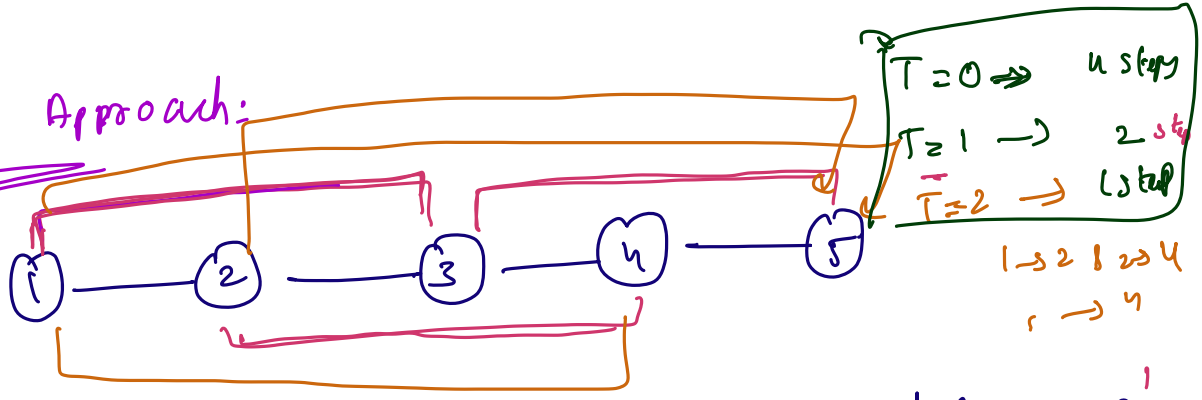
n, n → $(n-1) \times (n-2)$ → $O(n^2)$ for each



At max : (N-1) grand children

$(N-1) \times (N-1) = O(N^2)$ grand children

Magic Approach:

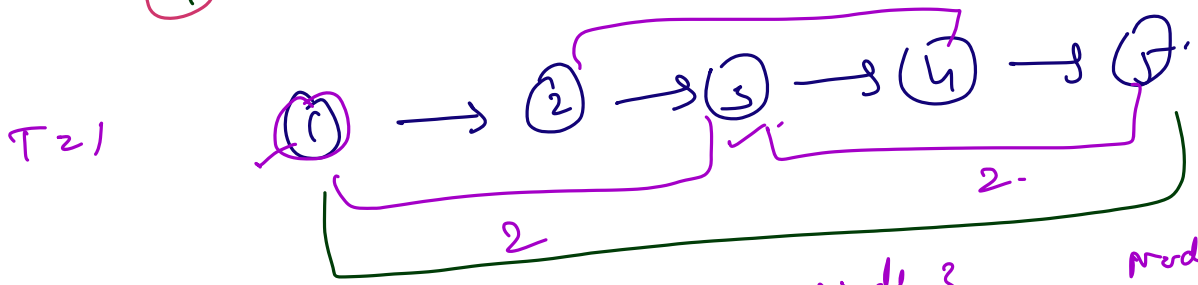


T=1.

We can reach all nodes at a distance 2^1 in just 1 step after T=1

T=2 : 2^2

We can reach all nodes at a distance $(4)(2^2)$ in just 1 step after T=2



Node 3

Node 5.

1-3 3-5.

$K \Rightarrow (2^{10})$

T=3

We can reach all nodes at a distance $8\ (2^3)$ in just 1 step after T=3

After K mins → 2^K

Diameter $\leq 2^K$

Diameter

$\log_2(\text{diameter}) \leq K$.

$K \geq \log_2(\text{diameter})$

$K = \text{ceil}(\log_2(\text{diameter}))$

diameter $= 2^{\log_2 2} = 2$

K.

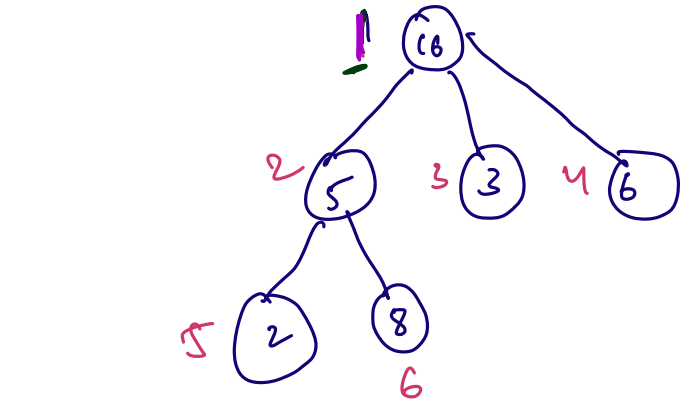
Buy & Sell Stocks

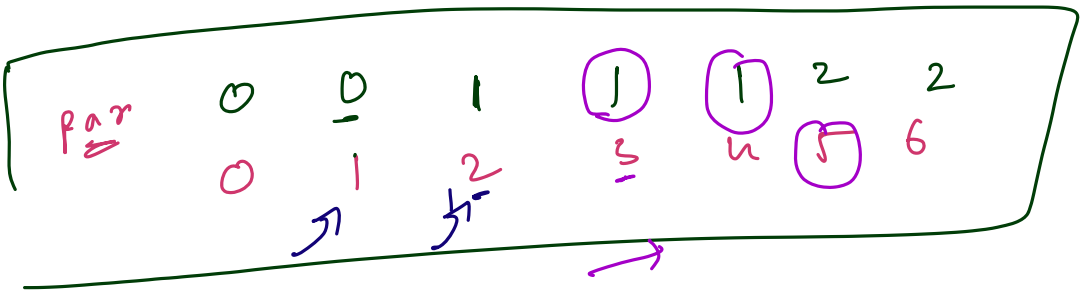
→ as many stocks
→ only 1 stock
→ only 2 stocks
→ K ~~set~~ stocks

- Bottom-up will help us optimize TC.

$O(N^2) \rightarrow O(N)$.

Doubts





par[2] = 1

```
[ 1: [2, 3, 4 ],
  2: [5, 6 ] ]
```

vector<vector<int>>

Backtracking → Trying all Possibilities

Recursion:

Backtrack → DP

dp

Bottom-up

Valuable Nodes

```
ans = maxVal(1);
int maxVal(int node){
      if(dp[node] != -1) return dp[node];

      // Take the current node
      val1 = value[node];

      for(int i = 0; i < ggc[node].size(); i++){
            int ggchild = ggc[i];
            val1 = val1 + maxVal(ggchild);
      }
      val2 = 0;
      for(int i = 0; i < child[node].size(); i++){
            int child_node = child[i];
            val1 = val1 + maxVal(child_node);
      }
      dp[node] = max(val1, val2);
      return dp[node];
}
```

Maximum Absolute difference

```
int findMinMax(node){
      // Find the min and max of its subtrees
      int minm = INT_MAX, maxm = INT_MINM;
      for(int i = 0; i < child[node].size(); i++){
            min_subtree, maxm_subtree = findMinMax(child[node][i]);
            minm = min (minm, min_subtree);
            maxm = max(maxm, max_subtree);
      }
      ans = max(ans, value[node] - minm;
      ans = max(ans, maxm - value[node];

      return min(minm, value[node]), maxm(maxm, value[node]);
}
```