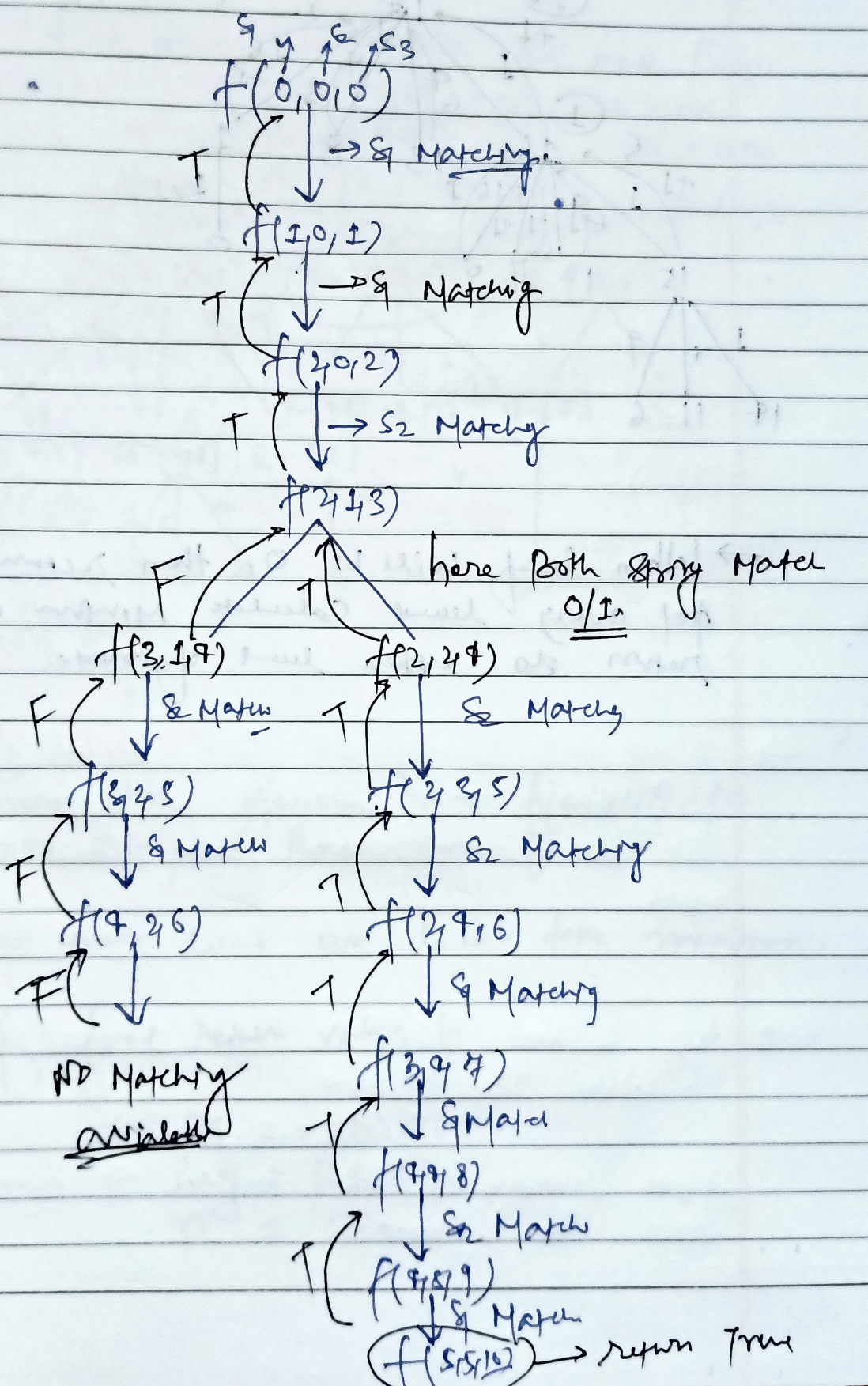
Day 5 Jul 13, 2022

List of Leetcode question

1. Stone Game III - LeetCode
2. Stone Game IV - LeetCode
3. Domino and Tromino Tiling - LeetCode skipped
4. Perfect Squares - LeetCode
5. Unique Binary Search Trees - LeetCode (Calan Number)
6. Wiggle Subsequence - LeetCode
7. Interleaving String - LeetCode

Interleaving String

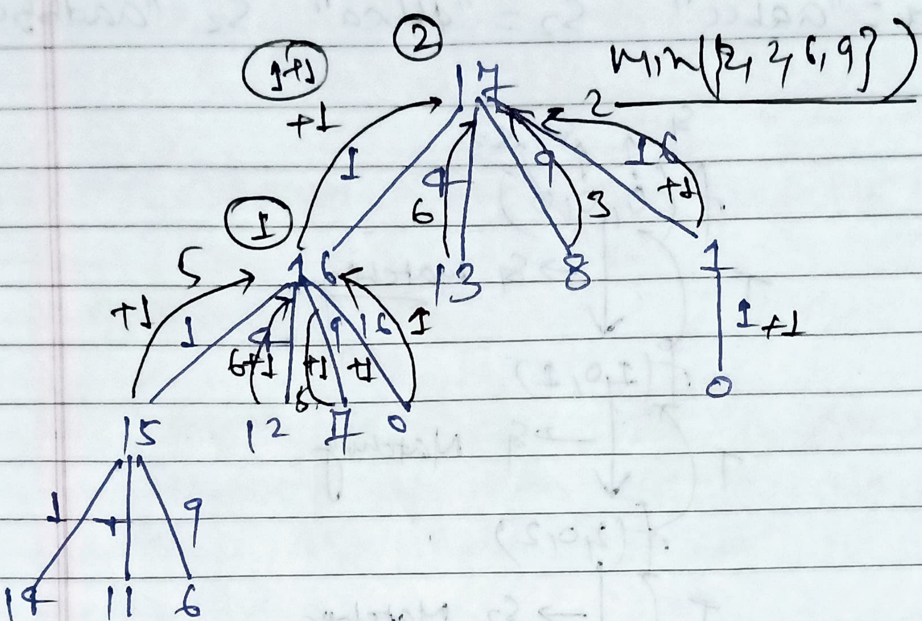
S_1 = "aabcc" S_2 = "dbbca" S_3 = "aadbbcbcac"



```cpp
#include<bits/stdc++.h>
using namespace std;


class Solution
{
public:
    bool getResult(string &s1, string &s2, string &s3, int ns1, int ns2, int ns3, vector<vector<vector<int>>> &dp)
    {
        if ((s1.size() + s2.size()) != s3.size())
        {
            return false;
        }
        if (ns1 < 0 and ns2 < 0 and ns3 < 0)
        {
            return true;
        }
        if (dp[ns1 + 1][ns2 + 1][ns3 + 1] != -1)
        {
            return dp[ns1 + 1][ns2 + 1][ns3 + 1];
        }
        if (ns1 >= 0 and ns2 >= 0 and ns3 >= 0 and s1[ns1] == s3[ns3] and s2[ns2] == s3[ns3])
        {
            return dp[ns1 + 1][ns2 + 1][ns3 + 1] = (getResult(s1, s2, s3, ns1 - 1, ns2, ns3 - 1, dp) or getResult(s1, s2, s3, ns1, ns2 - 1, ns3 - 1, dp));
        }
        else if (ns1 >= 0 and ns3 >= 0 and s1[ns1] == s3[ns3])
        {
            return dp[ns1 + 1][ns2 + 1][ns3 + 1] = getResult(s1, s2, s3, ns1 - 1, ns2, ns3 - 1, dp);
        }
        else if (ns2 >= 0 and ns3 >= 0 and s2[ns2] == s3[ns3])
        {
            return dp[ns1 + 1][ns2 + 1][ns3 + 1] = getResult(s1, s2, s3, ns1, ns2 - 1, ns3 - 1, dp);
        }
        else
        {
            return dp[ns1 + 1][ns2 + 1][ns3 + 1] = false;
        }


    }

    bool isInterleave(string s1, string s2, string s3)
    {
        int ns1 = s1.size();
        int ns2 = s2.size();
        int ns3 = s3.size();
        vector<vector<vector<int>>> dp(ns1 + 2, vector<vector<int>>(ns2 + 2, vector<int>(ns3 + 2, -1)));
        return getResult(s1, s2, s3, s1.size() - 1, s2.size() - 1, s3.size() - 1, dp);
    }
};

int main()
```

```cpp
{
    Solution s;
    string s1 = "aabcc";
    string s2 = "dbbca";
    string s3 = "aadbbcbcac";
    cout << s.isInterleave(s1, s2, s3) << endl;
    return 0;
}

/*
time complexity: O(n^3)
space complexity: O(n^3)
*/
```

Perfect Square



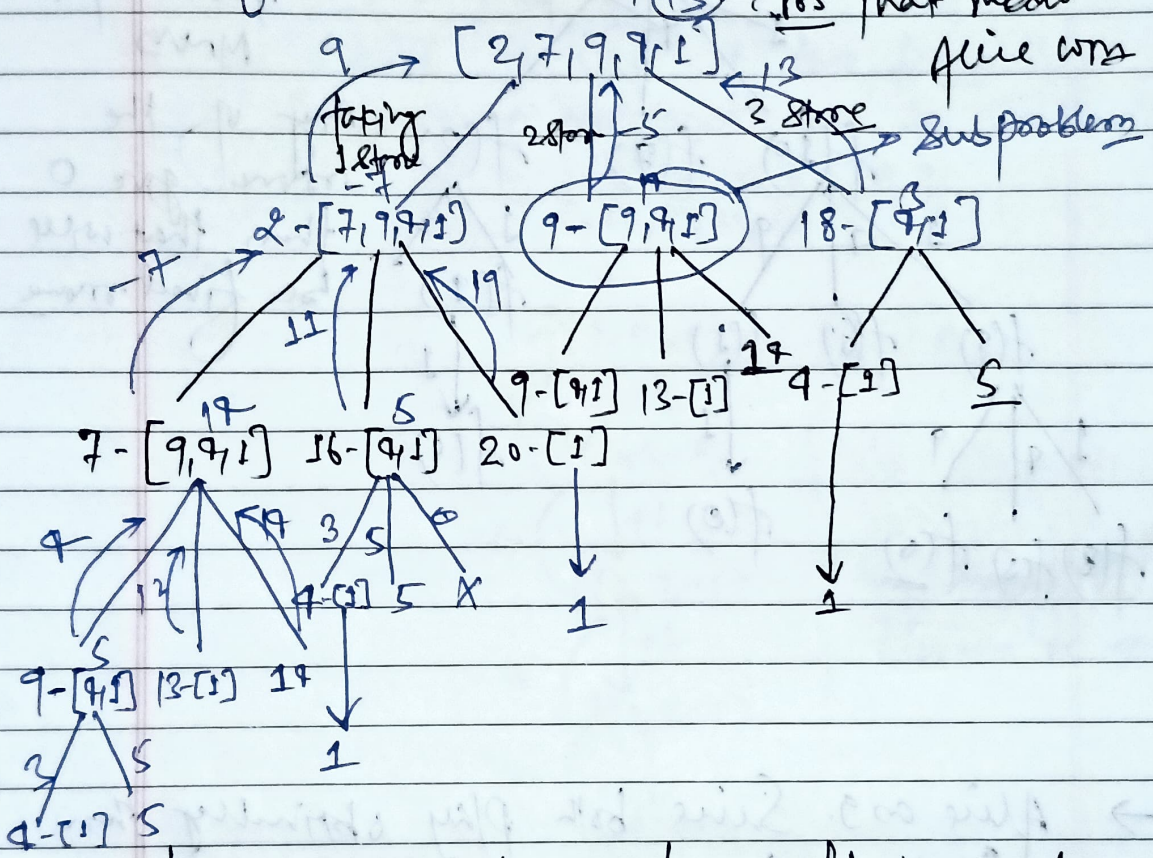
→ When leaf will be 0, then return 0
And every level calculate minimum and return to upper level of node.

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    int getAns(int n)
    {
        /*
        if(n==0)
        {
            return 0;
        }
        int ans=INT_MAX;

        for(int i=1;i*i<=n;i++)
        {
            ans= min(ans,1+getAns(n-i*i));
        }
        return ans;
        */
        vector<int> dp(n + 1, 0);

        for (int num = 1; num <= n; num++)
        {
            int ans = INT_MAX;

            for (int i = 1; i * i <= num; i++)
            {
                ans = min(ans, 1 + dp[num - i * i]);
            }
            dp[num] = ans;
        }

        return dp[n];
    }
    int numSquares(int n)
    {
        return getAns(n);
    }
};

int main()
{
    Solution s;
    cout << s.numSquares(12) << endl;
    return 0;
}

/*
time complexity: O(n)
space complexity: O(n)
*/

```

Stone Game III

Arr = [2, 7, 9, 4, 1]

first move will Alice. 2nd move Bob.



→ here Both player have flexibility to take one, two, three stones.

→ At every time we will take maximum.

→ if final return value is

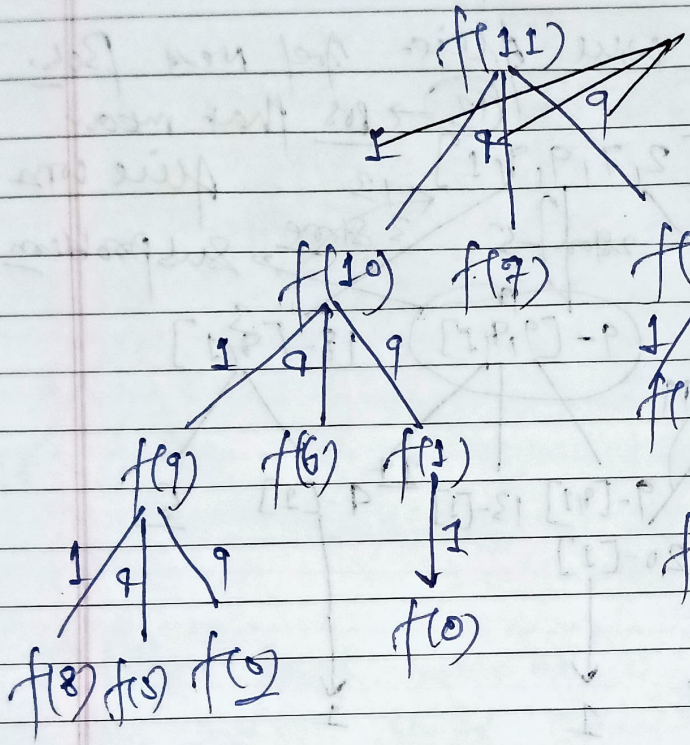
pos → Alice win
neg → Bob win
0 = Tie

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    int findResult(vector<int> &input, int index)
    {
        //        if(index>=input.size())
        //        {
        //            return 0;
        //        }
        //         int result=0;
        //         result= input[index]-findResult(input,index+1);

        //         if(index+1<input.size())
        //         {
        //             result= max(result,((input[index]+input[index+1])-findResult(input,index+2)));
        //         }

        //        if(index+2<input.size())
        //                            {
        //             result= max(result,(input[index]+input[index+1]+input[index+2])-(findResult(input,index+3)));
        //                            }
        //         return result;

        int n = input.size();
        vector<int> dp(n + 1, 0);

        for (int index = n - 1; index >= 0; index--)
        {
            int result = 0;
            result = input[index] - dp[index + 1];

            if (index + 1 < input.size())
            {
                result = max(result, (input[index] + input[index + 1]) - dp[index + 2]);
            }

            if (index + 2 < input.size())
            {
                result = max(result, (input[index] + input[index + 1] + input[index + 2]) - dp[index + 3]);
            }
            dp[index] = result;
        }

        return dp[0];
    }

    string stoneGameIII(vector<int> &stone)
    {

        int result = findResult(stone, 0);
        if (result == 0)
        {
```

```cpp
            return "Tie";
        }
        else if (result < 0)
        {
            return "Bob";
        }
        else
        {
            return "Alice";
        }
    }
};


int main()
{
    Solution s;
    vector<int> stone = {2, 7, 9, 4, 1};
    cout << s.stoneGameIII(stone) << endl;
    return 0;
}

/*
 time complexity : O(n);
    space complexity : O(n);
*/
```

Stone Game IV

n = 11



Trying all possible Approaches / or Moves

Try if the move give 0 then that will be final move

→ Alice as 9. Since both play optimally then 9 will try every possible approach so that next person can loose the match.

→ If any branches give 0, then 9 will be win the game.

If any of the branch doesn't return 0, then modern 9 have to loose the game.

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    bool calc(int n)
    {
        //         if(n==0)
        //         {
        //             return false;
        //         }

        //         for(int i=1;i*i<=n;i++)
        //         {
        // //              i*i= ye alice ka move agar calc(n-i*i)==false that means bob lost because n-i*i stones bobs ko mile the khelne ke liye
        //             if(calc(n-i*i)==false)
        //             {
        //                 return true;
        //             }
        //         }
        //         return false;

        vector<int> dp(n + 1, 0);

        for (int index = 1; index <= n; index++)
        {
            for (int i = 1; i * i <= index; i++)
            {
                if (dp[index - i * i] == false)
                {
                    dp[index] = true;
                }
            }
        }

        return dp[n];
    }

    bool winnerSquareGame(int n)
    {

        return calc(n);
    }
};

int main()
{
    Solution s;
    cout << s.winnerSquareGame(2) << endl;
    return 0;
}
/*
time complexity:O(n)
space complexity:O(n)
*/
```

Wiggle Subsequence

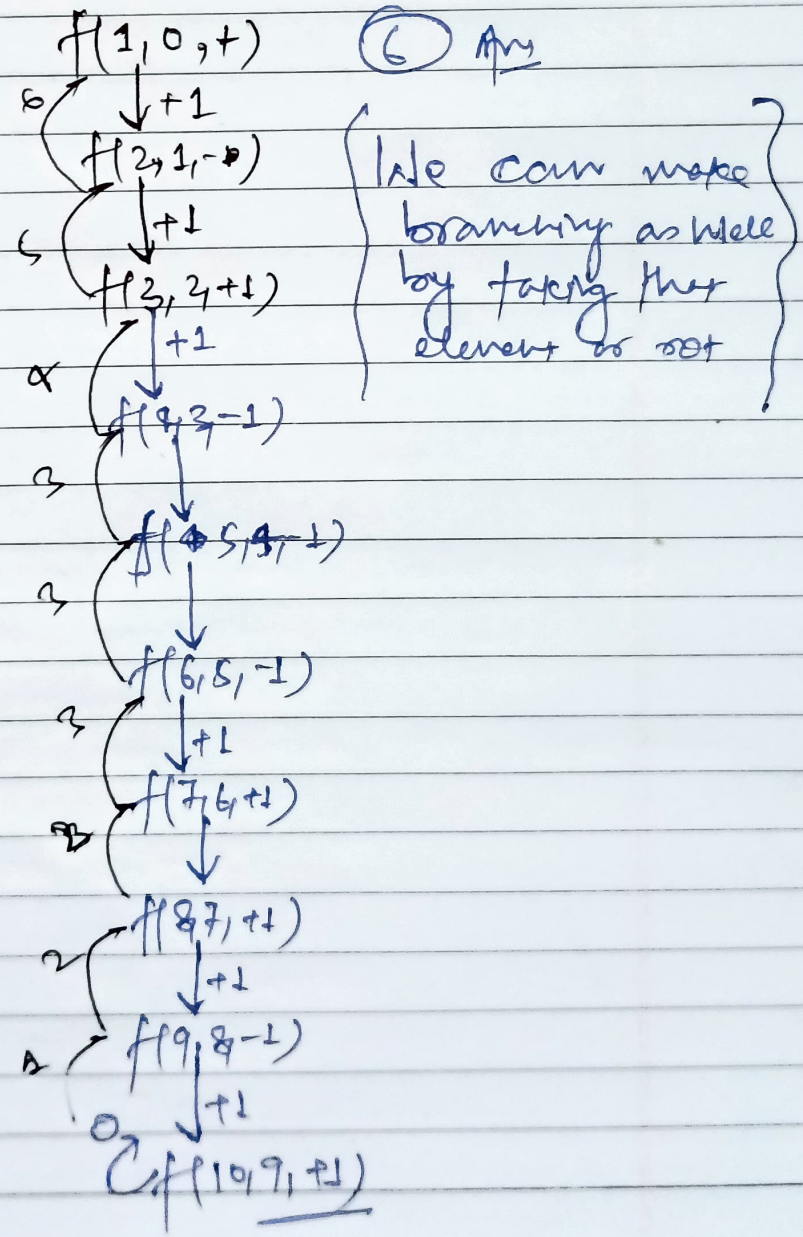
→ In this question we need to check alternate positive negative subsequence or Negative positive subsequence

(+ −) / (− +)

We need to call it two times by 1st Considering + and next call by considering −

0 1 2 3 4 5 6 7 8 9

[1, 17, 5, 10, 13, 15, 10, 5, 16, 8]



6 Ans

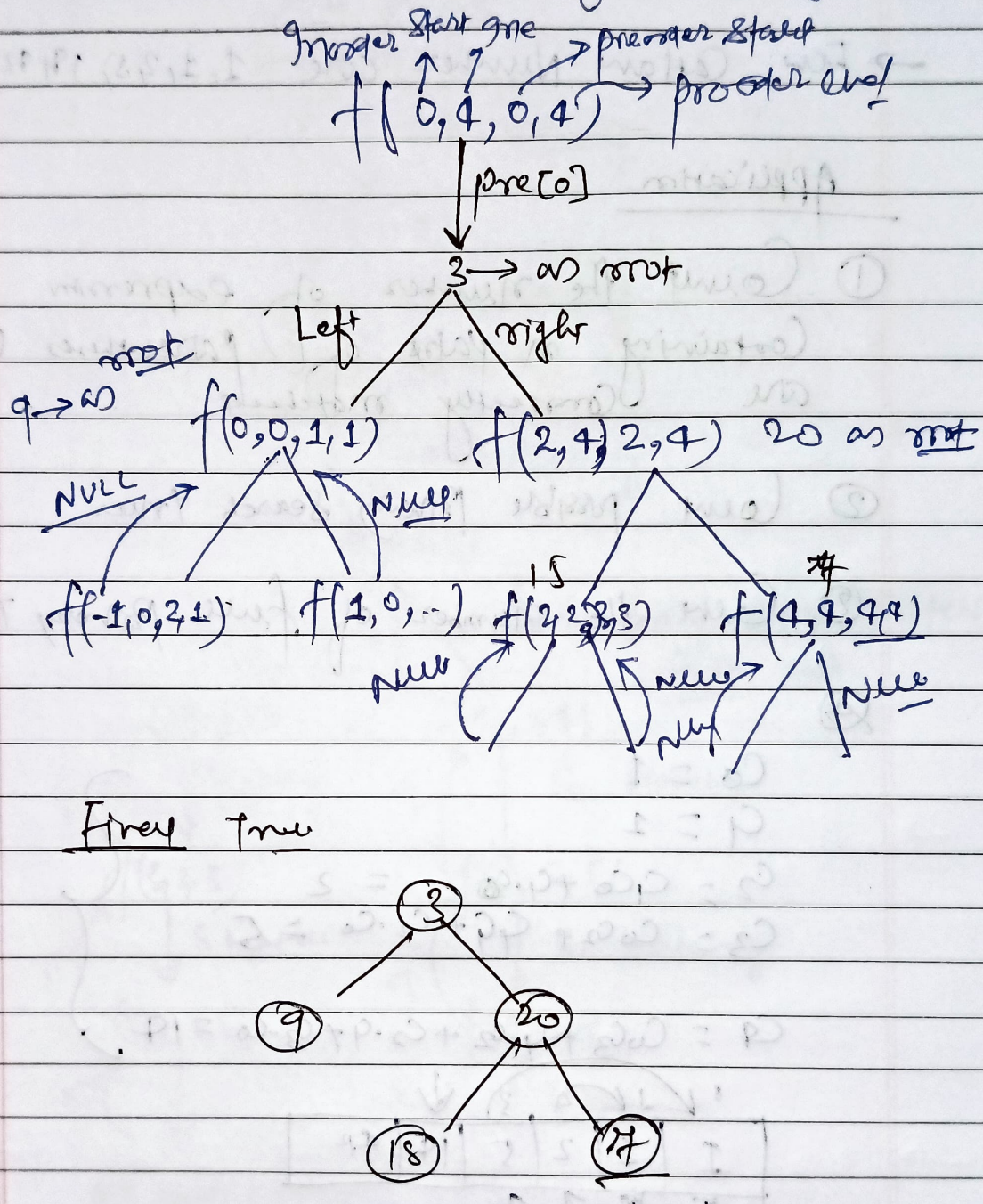
We can make branching as well by taking that element or not

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{

    int wiggle(vector<int> &num, int curr, int prev, bool positive)
    {
        if (curr >= num.size() or prev >= num.size())
        {
            return 0;
        }
        if (positive)
        {
            if (num[prev] < num[curr])
            {
                return 1 + wiggle(num, curr + 1, curr, !positive);
            }
            else
            {
                return wiggle(num, curr + 1, curr, positive);
            }
        }
        else
        {
            if (num[prev] > num[curr])
            {
                return 1 + wiggle(num, curr + 1, curr, !positive);
            }
            else
            {
                return wiggle(num, curr + 1, curr, positive);
            }
        }
    }

public:
    int wiggleMaxLength(vector<int> &nums)
    {

        return 1 + max(wiggle(nums, 1, 0, true), wiggle(nums, 1, 0, false));
    }
};

int main()
{
    Solution s;
    vector<int> nums = {1, 7, 4, 9, 2, 5};
    cout << s.wiggleMaxLength(nums) << endl;
    return 0;
}

/*
time complexity: O(n)
space complexity: O(1)
*/
```

Construct Binary Tree preorder And Inorder

pre = [3, 9, 20, 15, 7] root Left right

Inorder = [9, 3, 15, 20, 7] left root right



Final Tree

```cpp
#include<bits/stdc++.h>
using namespace std;

 struct TreeNode {
     int val;
     TreeNode *left;
     TreeNode *right;
     TreeNode() : val(0), left(nullptr), right(nullptr) {}
     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 };

class Solution
{
public:
    TreeNode *makeTree(vector<int> &preorder, vector<int> &inorder, int ins, int ine, int pres, int pree)
    {
        if (pres > pree)
        {
            return nullptr;
        }
        int _root = preorder[pres];
        int _index = -1;
        for (int i = ins; i <= ine; i++)
        {
            if (inorder[i] == _root)
            {
                _index = i;
                break;
            }
        }

        if (_index == -1)
        {
            return NULL;
        }

        TreeNode *root = new TreeNode(_root);

        root->left = makeTree(preorder, inorder, ins, _index - 1, pres + 1, pres + (_index - ins));
        root->right = makeTree(preorder, inorder, _index + 1, ine, pres + _index - ins + 1, pree);

        return root;
    }

    TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder)
    {
        int n = preorder.size();
        return makeTree(preorder, inorder, 0, n - 1, 0, n - 1);
    }
};

int main()
{
    Solution s;
```

```cpp
    vector<int> preorder = {3, 9, 20, 15, 7};
    vector<int> inorder = {9, 3, 15, 20, 7};
    TreeNode *root = s.buildTree(preorder, inorder);
    return 0;
}
/*
time complexity: O(n)
space complexity: O(n)
*/
```

Unique BST

(2)

Catlan Number

→ Few Catlan Number are 1, 1, 2, 5, 14, 42 ---

Application

① Count the number of expression containing n pairs of parentheses which are correctly matched

② Count possible Binary Search Tree.

③ Count the number of full Binary Tree

④

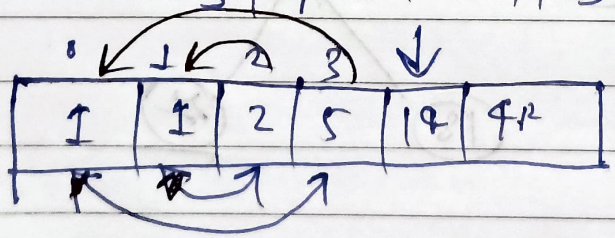
$C_0 = 1$

$C_1 = 1$

$C_2 = C_1C_0 + C_1 \cdot C_0 = 2$

$C_3 = C_0C_2 + C_1C_1 + C_2 \cdot C_0 = 5$

$C_4 = C_0C_3 + C_1 \cdot C_2 + C_2 \cdot C_1 + C_3 \cdot C_0 = 14$



$C_4 = C_0 \cdot C_3 + C_1 C_2 + C_2 \cdot C_1 + C_3 \cdot C_0$

$j \quad (i-j-1)$

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    int catlen_num(int n)
    {
        int dp[n + 1];
        dp[0] = dp[1] = 1;

        for (int i = 2; i <= n; i++)
        {
            dp[i] = 0;
            for (int j = 0; j < i; j++)
            {
                dp[i] += dp[j] * dp[i - j - 1];
            }
        }

        return dp[n];
    }

    int numTrees(int n)
    {
        return catlen_num(n);
    }
};

int main()
{
    Solution s;
    cout << s.numTrees(3) << endl;
    return 0;
}
/*
time complexity: O(n)
space complexity: O(n)
*/
```