

Dynamic Programming - 2

- Karun Karthik

Contents

- 16. Best Time to Buy and Sell Stock
- 17. Best Time to Buy and Sell Stock II
- 18. Best Time to Buy and Sell Stock III
- 19. Best Time to Buy and Sell Stock IV
- 20. Best Time to Buy and Sell Stock with Cooldown
- 21. Best Time to Buy and Sell Stock with Transaction Fee
- 22. Jump Game
- 23. Jump Game II
- 24. Reach a given Score
- 25. Applications of Catalan Numbers
- 26. Nth Catalan Number
- 27. Number of Valid Parenthesis Expression
- 28. Unique Binary Search Trees

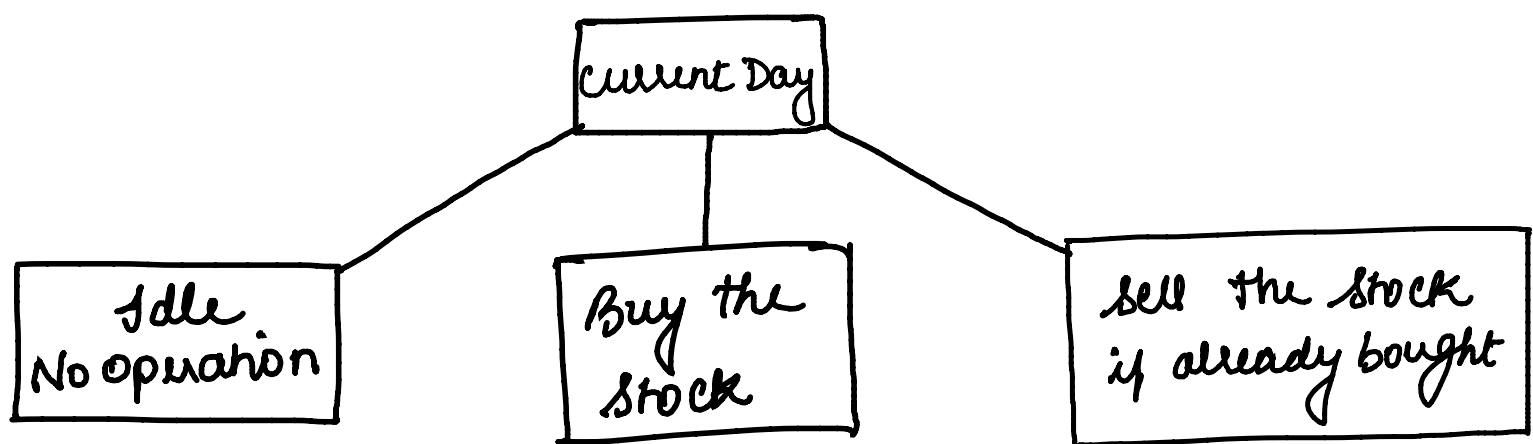
⑩ Best time to Buy & Sell Stock →

Given an array of prices, find the max profit if we are allowed to do one transaction

Eg

prices = [7, 1, 5, 3, 6, 4] → we get maxProfit when we buy at day 0 & sell on day 4
 0 1 2 3 4 5
 ⇒ profit = 6 - 1 = 5.

lets look at choices we have,



→ to handle the case that transaction could occur once, we use a variable called transaction = 1.

→ to handle these cases, we use a variable called canBuy.

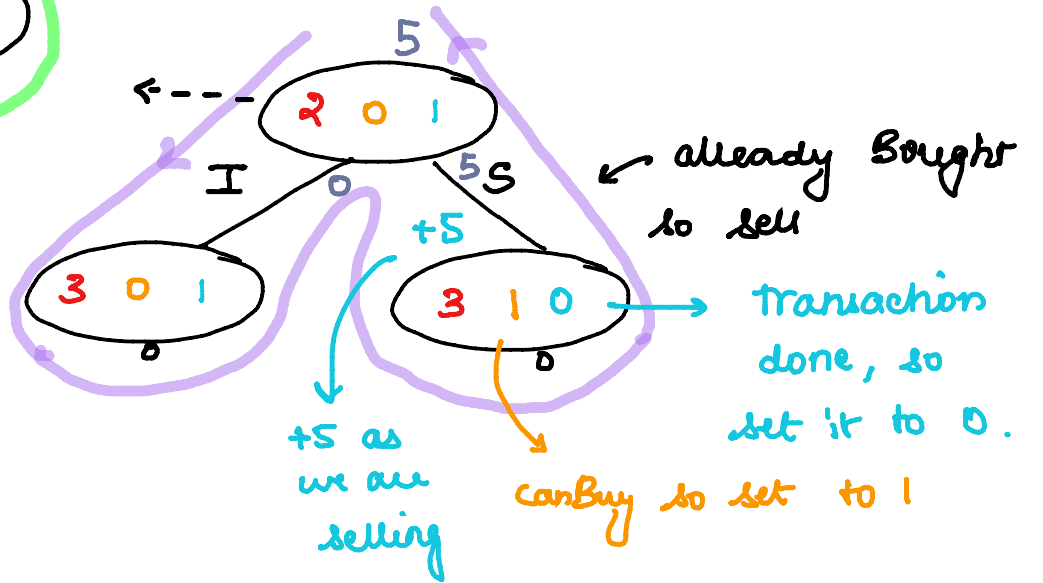
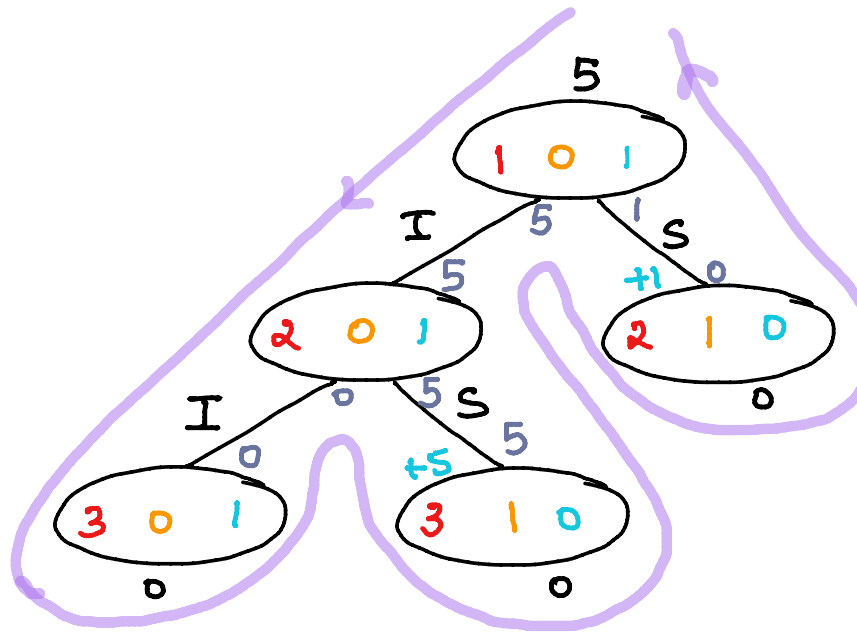
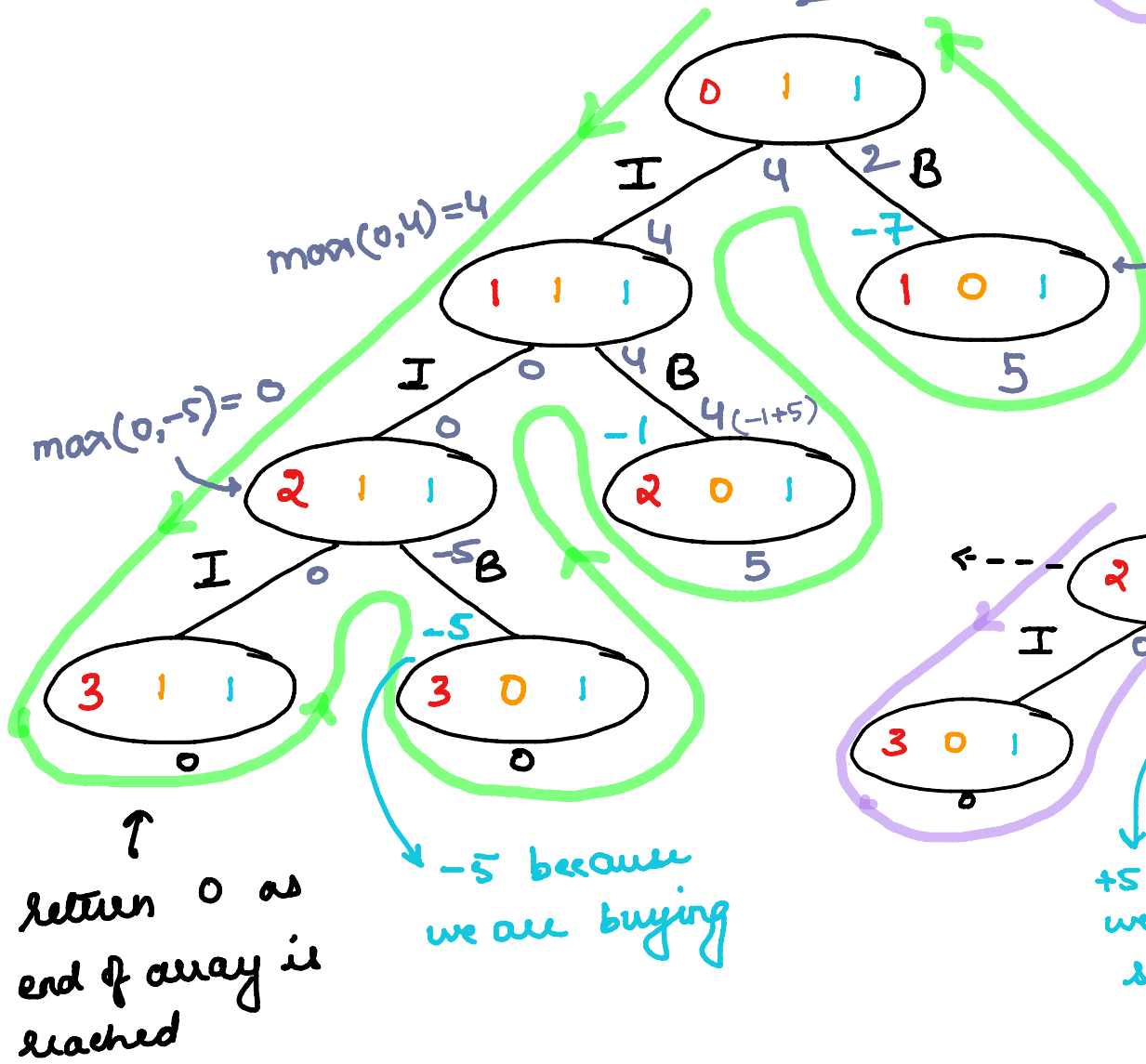
→ once bought canBuy = false
→ once sold canBuy = true

∴ Our recursive structure would be as follows →

current Day, canBuy, transaction

Eg $[7, 1, 5]$
0 1 2

Result →
4



code →

```
1 class Solution {
2 public:
3     int find(vector<int> &prices, int currDay, int k, bool canBuy, vector<vector<int>> &memo){
4
5         if(currDay >= prices.size() || k<=0 ) return 0;
6
7         if(memo[currDay][canBuy] != -1) return memo[currDay][canBuy];
8
9         if(canBuy)
10        {
11            int idle = find(prices, currDay+1, k, canBuy, memo);
12            int buy = -prices[currDay] + find(prices, currDay+1, k, !canBuy, memo);
13            return memo[currDay][canBuy] = max(buy, idle);
14        }
15        else
16        {
17            int idle = find(prices, currDay+1, k, canBuy, memo);
18            int sell = prices[currDay] + find(prices, currDay+1, k-1, !canBuy, memo);
19            return memo[currDay][canBuy] = max(sell, idle);
20        }
21    }
22    int maxProfit(vector<int>& prices) {
23        int n = prices.size();
24        vector<vector<int>> memo(n,vector<int> (2,-1));
25        // canBuy = true and transaction as k = 1
26        return find(prices,0,1,true,memo);
27    }
28    };
```


⑪ Best time to Buy & Sell Stock - II →

→ In this we can have many transactions that can be done.

Eg → prices = [7, 1, 5, 3, 6, 4]

↳ Buy on 1 & sell on 2 profit = 5 - 1 = 4

Buy on 3 & sell on 4 profit = 6 - 3 = 3

Total Profit = 7 Ans

code →

Remove the parameter K i.e transaction limit.

```
1  class Solution {
2  public:
3      int find(vector<int> &prices, int currDay, bool canBuy, vector<vector<int>> &memo){
4
5          if(currDay >= prices.size()) return 0;
6
7          if(memo[currDay][canBuy] != -1) return memo[currDay][canBuy];
8
9          if(canBuy)
10         {
11             int idle = find(prices, currDay+1, canBuy, memo);
12             int buy = -prices[currDay] + find(prices, currDay+1, !canBuy, memo);
13             return memo[currDay][canBuy] = max(buy, idle);
14         }
15         else
16         {
17             int idle = find(prices, currDay+1, canBuy, memo);
18             int sell = prices[currDay] + find(prices, currDay+1, !canBuy, memo);
19             return memo[currDay][canBuy] = max(sell, idle);
20         }
21     }
22     int maxProfit(vector<int>& prices) {
23         int n = prices.size();
24         vector<vector<int>> memo(n, vector<int> (2, -1));
25         // canBuy = true and transaction are infinite so ignore k
26         return find(prices, 0, true, memo);
27     }
28 };
```

⑮ Best time to Buy & Sell stock - III →

In this maximum profit has to be achieved by making atmost 2 transactions.

Eg prices = ^{0 1 2 3 4 5 6 7}
[3, 3, 5, 0, 0, 3, 1, 4]

↳ Buy on 4 & sell on 5 profit = 3 - 0 = 3
Buy on 6 & sell on 7 profit = 4 - 1 = 3

Total Profit = 6 Ans

code →

In the base condition is no. of transactions ≥ 2
then return 0.

(Line 6)

↳ i.e possible transactions are when it is = 0, 1

```
1 class Solution {
2 public:
3     int find(vector<int> &prices, int currDay, int transaction, bool canBuy,
4             vector<vector<vector<int>>> &memo){
5
6         if(currDay >= prices.size() || transaction >= 2) return 0;
7
8         if(memo[currDay][canBuy][transaction] != -1) return memo[currDay][canBuy][transaction];
9
10        if(canBuy)
11        {
12            int idle = find(prices, currDay+1, transaction, canBuy, memo);
13            int buy = -prices[currDay] + find(prices, currDay+1, transaction, !canBuy, memo);
14            return memo[currDay][canBuy][transaction] = max(buy, idle);
15        }
16        else
17        {
18            int idle = find(prices, currDay+1, transaction, canBuy, memo);
19            int sell = prices[currDay] + find(prices, currDay+1, transaction+1, !canBuy, memo);
20            return memo[currDay][canBuy][transaction] = max(sell, idle);
21        }
22    }
23    int maxProfit(vector<int>& prices) {
24        int n = prices.size();
25        vector<vector<vector<int>>> memo(n, vector<vector<int>>(2, vector<int>(2, -1)));
26        // canBuy = true and transactions are allowed 2 times
27        return find(prices, 0, 0, true, memo);
28    }
29 };
```

⑪ Best time to Buy & Sell Stock - IV →

This is a generalised version of previous problem, instead of limiting it to 2 transactions, we need to allow atmost k transactions.

code →

Pass k as an argument & use it to limit transaction in base condition. (Line 6)

```
1 class Solution {
2 public:
3     int find(vector<int> &prices, int currDay, int transaction, int k, bool canBuy,
4             vector<vector<vector<int>>> &memo){
5
6         if(currDay >= prices.size() || transaction >= k) return 0;
7
8         if(memo[currDay][canBuy][transaction] != -1) return memo[currDay][canBuy][transaction];
9
10        if(canBuy)
11        {
12            int idle = find(prices, currDay+1, transaction, k, canBuy, memo);
13            int buy = -prices[currDay] + find(prices, currDay+1, transaction, k, !canBuy, memo);
14            return memo[currDay][canBuy][transaction] = max(buy, idle);
15        }
16        else
17        {
18            int idle = find(prices, currDay+1, transaction, k, canBuy, memo);
19            int sell = prices[currDay] + find(prices, currDay+1, transaction+1, k, !canBuy, memo);
20            return memo[currDay][canBuy][transaction] = max(sell, idle);
21        }
22    }
23    int maxProfit(int k, vector<int>& prices) {
24        int n = prices.size();
25        vector<vector<vector<int>>> memo(n, vector<vector<int>>(2, vector<int>(k+1, -1)));
26        // canBuy = true and transactions are allowed atmost k times
27        return find(prices, 0, 0, k, true, memo);
28    }
29 };
```

②0 Best time to Buy & Sell stock with Cooldown →

In this, cooldown means that we cannot buy a stock on the immediate day after it is sold.

⇒ The day after sold should be skipped.

code →

To skip day after sell, increment the currDay by 2.
(Line 18)

```
1  class Solution {
2  public:
3      int find(vector<int> &prices, int currDay, bool canBuy, vector<vector<int>> &memo){
4
5          if(currDay >= prices.size()) return 0;
6
7          if(memo[currDay][canBuy] != -1) return memo[currDay][canBuy];
8
9          if(canBuy)
10         {
11             int idle = find(prices, currDay+1, canBuy, memo);
12             int buy = -prices[currDay] + find(prices, currDay+1, !canBuy, memo);
13             return memo[currDay][canBuy] = max(buy, idle);
14         }
15         else
16         {
17             int idle = find(prices, currDay+1, canBuy, memo);
18             int sell = prices[currDay] + find(prices, currDay+2, !canBuy, memo);
19             return memo[currDay][canBuy] = max(sell, idle);
20         }
21     }
22     int maxProfit(vector<int>& prices) {
23         int n = prices.size();
24         vector<vector<int>> memo(n, vector<int> (2, -1));
25         // canBuy = true & transaction = infinite so ignore k & while sell, currDay +=2
26         return find(prices, 0, true, memo);
27     }
28 };
```


②1 Best time to Buy & Sell Stock with Transaction Fee →

In this variation, we do not have limit on transaction but while making a transaction i.e. selling it, some fee has to be paid i.e. transaction fee.

code →

Deduct the fee from the selling day's amount.

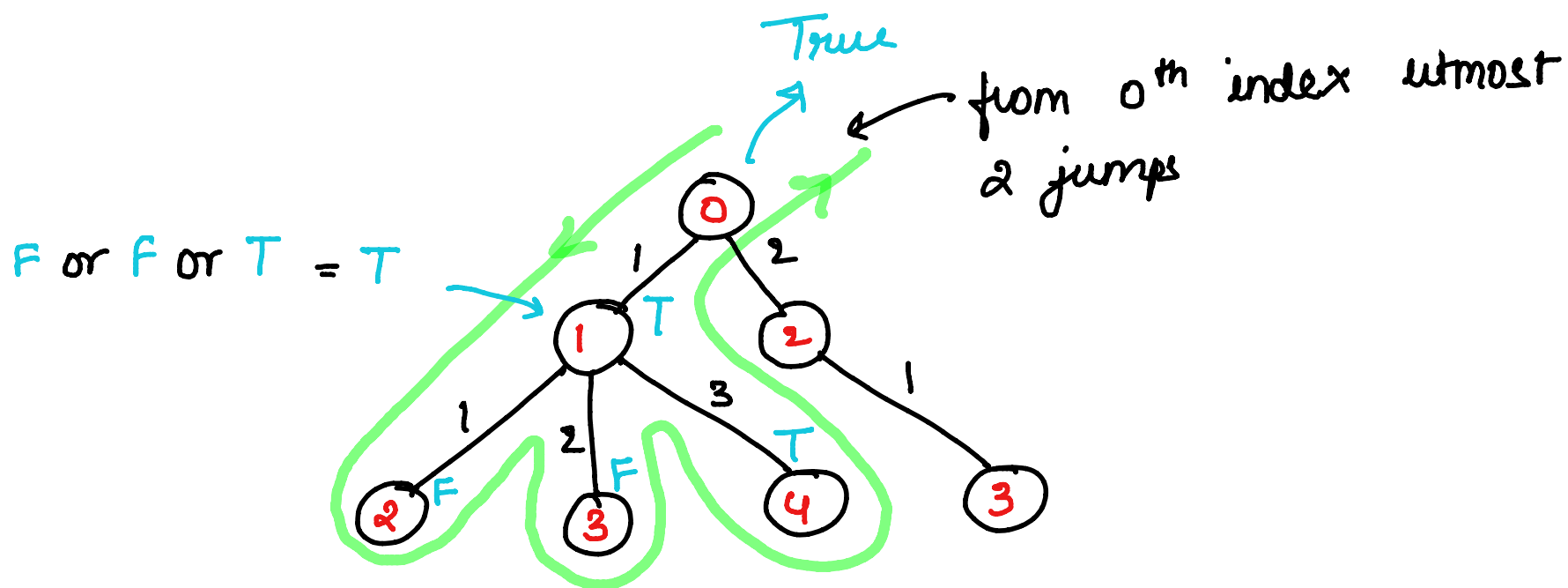
(Line 18)

```
1 class Solution {
2 public:
3     int find(vector<int> &prices, int currDay, int fee, bool canBuy, vector<vector<int>> &memo){
4
5         if(currDay >= prices.size()) return 0;
6
7         if(memo[currDay][canBuy] != -1) return memo[currDay][canBuy];
8
9         if(canBuy)
10        {
11            int idle = find(prices, currDay+1, fee, canBuy, memo);
12            int buy = -prices[currDay] + find(prices, currDay+1, fee, !canBuy, memo);
13            return memo[currDay][canBuy] = max(buy, idle);
14        }
15        else
16        {
17            int idle = find(prices, currDay+1, fee, canBuy, memo);
18            int sell = (prices[currDay]-fee) + find(prices, currDay+1, fee, !canBuy, memo);
19            return memo[currDay][canBuy] = max(sell, idle);
20        }
21    }
22
23    int maxProfit(vector<int>& prices, int fee) {
24        int n = prices.size();
25        vector<vector<int>> memo(n, vector<int> (2, -1));
26        // canBuy = true & transaction = infinite so ignore k & while selling deduce fee
27        return find(prices, 0, fee, true, memo);
28    }
29 };
```

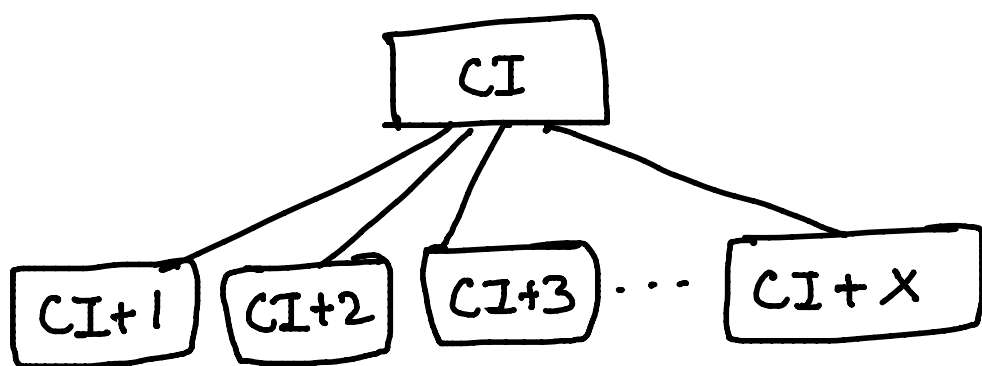
②② Jump Game →

Given array of nums which indicate max number of jump from any index. Return true if you can reach last index.

Eg $nums = [2, 3, 1, 1, 4]$
0 1 2 3 4



Therefore, $[_ _ \frac{x}{CI} _ _]$



can be implemented using a for loop.

Note: Submitting DP solution gives TLE. This is just for understanding. Optimal solution involves Greedy approach.

$$T_c \rightarrow O(\underbrace{\max(\text{nums}[i])}_\text{max time for forloop} * n)$$

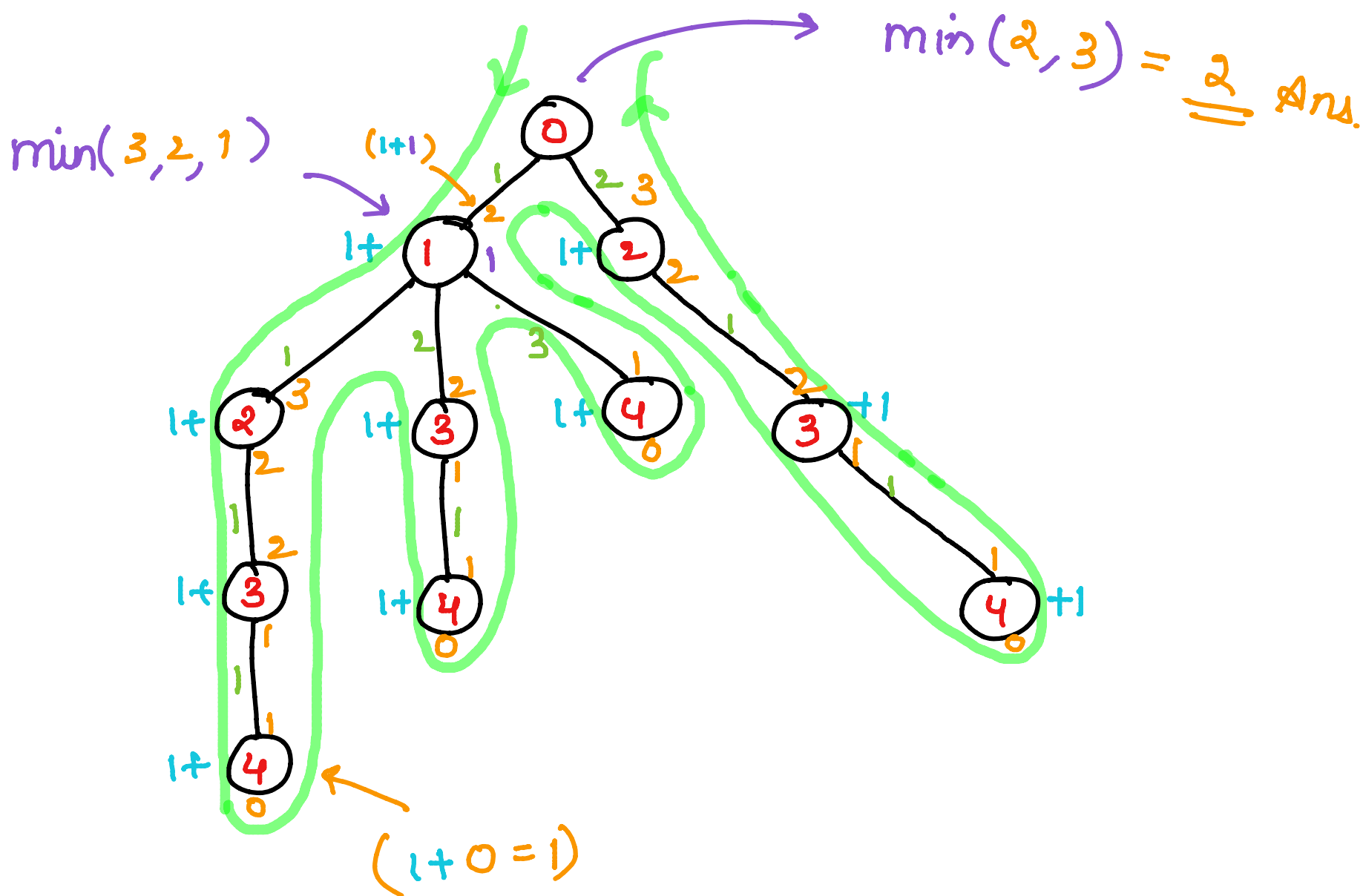
code →

```
1  class Solution {
2  public:
3      bool isPossible(vector<int>&nums, int curr, unordered_map<int,bool>&memo)
4      {
5          if(curr >= nums.size()-1) return true;
6
7          int currKey = curr;
8
9          if(memo.find(currKey)!=memo.end()) return memo[currKey];
10
11         int currJump = nums[curr];
12
13         if(currJump >= nums.size() - curr) return true;
14
15         bool ans = false;
16
17         for(int i=1; i<=currJump; i++){
18             bool tempAns = isPossible(nums,curr+i,memo);
19             ans = ans || tempAns;
20         }
21         return memo[currKey] = ans;
22     }
23
24     bool canJump(vector<int>& nums){
25         unordered_map<int,bool>memo;
26         return isPossible(nums, 0, memo);
27     }
28 };
```

②③ Jump Game II →

Given array of nums which indicate max number of jump from any index. Reach last index in minimum number of moves.

Eg $nums = [2, 3, 1, 1, 4]$
0 1 2 3 4



→ If $currentIndex \geq lastIndex$
then return 0.

while returning add 1 for counting ways!

Code →

```
1  class Solution {
2  public:
3
4      int minJumps(vector<int>& nums,int curr,vector<int>&memo)
5      {
6          if( curr >= nums.size()-1) return 0;
7
8          int currKey = curr;
9          if(memo[currKey]!=-1) return memo[currKey];
10
11         int currJump = nums[curr];
12
13         // some large value
14         int ans = 10001;
15
16         for(int i=1;i<=currJump;i++){
17             int tempans = 1 + minJumps(nums,curr+i,memo);
18             ans = min(ans, tempans);
19         }
20         return memo[currKey] = ans;
21     }
22
23     int jump(vector<int>& nums) {
24         vector<int> memo(nums.size()+1,-1);
25         return minJumps(nums, 0, memo);
26     }
27 };
```

②④ Reach a given Score →

Given 3 scores [3, 5, 10] & 'n'.

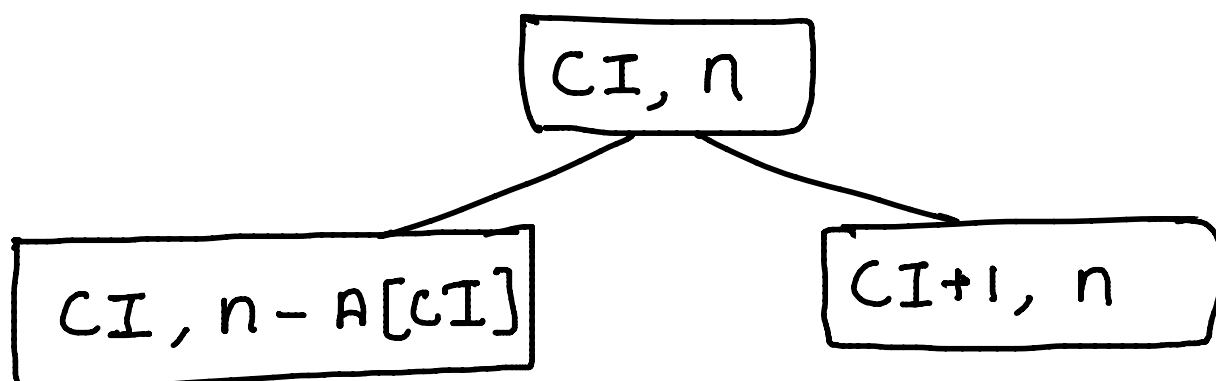
Return total number of ways to create n using the
stones.

Ex $n=8$ then no. of ways to create 8 from $[3, 5, 10]$ is 1. $(3+5)$

$n = 13$ then no. of ways to create 13 from $[3, 5, 10]$ is 2 $(3+5+5)$ & $(3+10)$

$n=20$ then no. of ways to create 20 from $[3, 5, 10]$
is 4 $(3+3+3+3+3+5)$ & $(5+5+5+5)$
& $(5+5+10)$ & $(10+10)$

$\therefore$ let's say $A = [3, 5, 10]$ then



code →

```
1  typedef long long LL;
2
3  LL ways(int curr, LL n, vector<int>&score, vector<vector<int>>&vec)
4  {
5      if(n==0) return 1;
6
7      if(curr>=score.size()) return 0;
8
9      if(vec[curr][n]!=-1) return vec[curr][n];
10
11     LL consider = 0;
12
13     if(score[curr]<=n)
14         consider = ways(curr,n-score[curr],score,vec);
15
16     LL notconsider = ways(curr+1,n,score,vec);
17
18     return vec[curr][n] = consider + notconsider;
19 }
20
21 LL count(LL n)
22 {
23     vector<int>score{3,5,10};
24     vector<vector<int>>vec(score.size(),vector<int>(1001,-1));
25     return ways(0,n,score,vec);
26 }
```

②⑤ Applications of Catalan Number →

Catalan Numbers are defined using the formula

$$C_n = \frac{(2n)!}{(n+1)! n!} = \prod_{k=2}^n \frac{n+k}{k} \quad \text{for } n \geq 0$$

this can be used recursively as follows,

$$C_{n+1} = \sum_{i=0}^n C_i C_{i-1} \quad \left. \vphantom{\sum_{i=0}^n} \right\} \quad n \geq 0 \quad \& \quad C_0 = 1$$

$$\rightarrow C_0 = \underline{\underline{1}}.$$

$$\rightarrow C_1 = \underline{\underline{1}}.$$

$$\rightarrow C_2 = C_0 \cdot C_1 + C_1 \cdot C_0 = 1 \cdot 1 + 1 \cdot 1 = \underline{\underline{2}}.$$

$$\rightarrow C_3 = C_0 C_2 + C_1 C_1 + C_2 C_0 = 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = \underline{\underline{5}}.$$

$$\begin{aligned} \rightarrow C_4 &= C_0 C_3 + C_1 C_2 + C_2 C_1 + C_3 C_0 \\ &= 1 \cdot 5 + 1 \cdot 2 + 2 \cdot 1 + 5 \cdot 1 = \underline{\underline{14}}. \end{aligned}$$

apps →

1. No. of possible BST with n keys.
2. No. of valid combinations for n pair of parenthesis.

26) Nth Catalan Number →

To find Nth Catalan Number we can use formula

$$C_{n+1} = \sum_{i=0}^n C_i C_{i-1} \quad \left. \vphantom{\sum_{i=0}^n} \right\} n \geq 0 \quad \& \quad C_0 = 1$$

↳ this can be implemented by

- i) having base condition for $n == 0$ & $n == 1$
- ii) using a loop to sum values from $i = 0$ to n

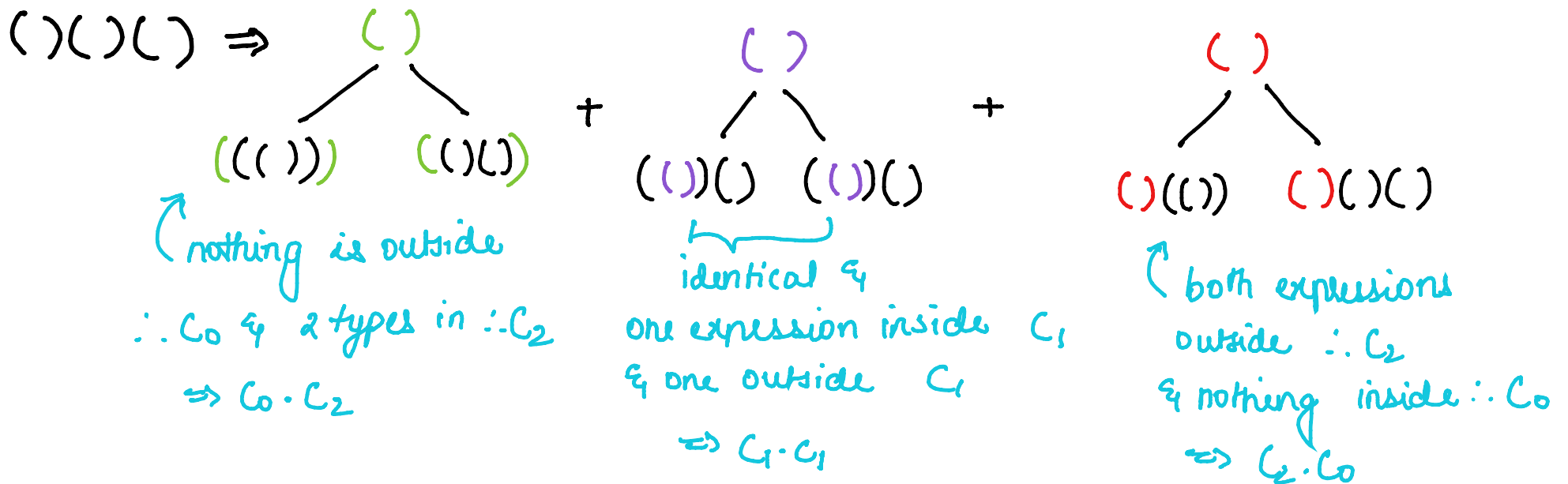
Code →

```
1  class Solution
2  {
3      public:
4      cpp_int ncatalan(int n, vector<cpp_int>& memo) {
5          if(n == 0 || n == 1) return 1;
6
7          int curr = n;
8          if(memo[curr] != -1) return memo[curr];
9
10         cpp_int catalan = 0;
11
12         for(int i=0; i<n; i++) {
13             catalan += ncatalan(i, memo)*ncatalan(n-i-1, memo);
14         }
15
16         memo[curr] = catalan;
17         return memo[curr];
18     }
19
20     cpp_int findCatalan(int n)
21     {
22         vector<cpp_int> memo(1001, -1);
23         return ncatalan(n, memo);
24     }
25 };
```


27) Number of valid Parenthesis Expression →

Given N , find total number of ways in which we can arrange N pair of parenthesis in a Balanced way.

Eg $N=4 \Rightarrow ()()(), ()(()) , (())(), ((())) \therefore \text{res} = 4$



$\Rightarrow C_0 \cdot C_2 + C_1 \cdot C_1 + C_2 \cdot C_0 = C_3 \Rightarrow \text{for } n \text{ we need to find } \text{ncatalan}(n/2)$

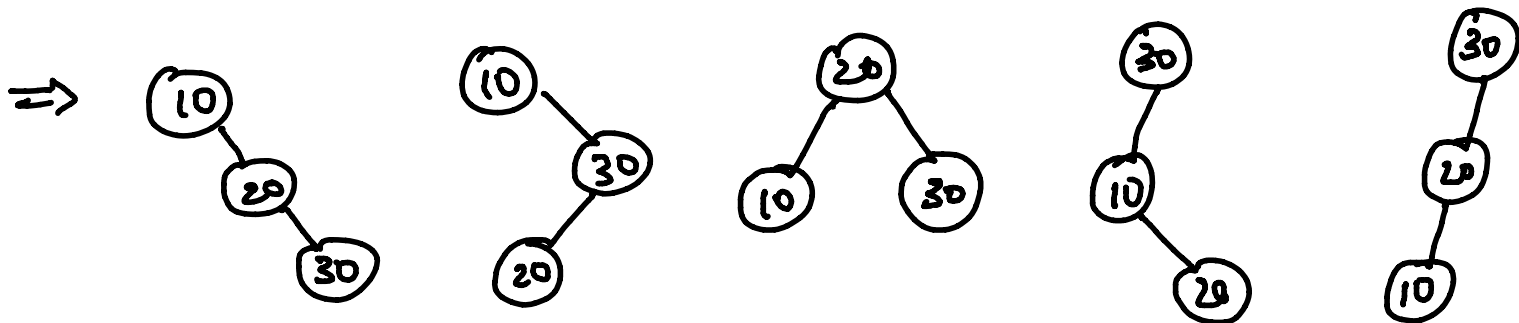
Code →

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int ncatalan(int n, unordered_map<int,int>& memo) {
5      if(n == 0 || n == 1) return 1;
6
7      int curr = n;
8      if(memo[curr] != -1) return memo[curr];
9
10     int catalan = 0;
11
12     for(int i=0; i<n; i++) {
13         catalan += ncatalan(i, memo)*ncatalan(n-i-1, memo);
14     }
15
16     memo[curr] = catalan;
17     return memo[curr];
18 }
19
20 int countValidParenthesis(int n)
21 {
22     unordered_map<int,int> memo;
23     return ncatalan(n/2, memo);
24 }
25
26 int main(){
27     int n;
28     cin>>n;
29     cout<<countValidParenthesis(n);
30 }
```

28) Unique Binary Search Trees →

given integer N , return no. of unique BST that can be formed.

Eg $n=3$ & let's say elements are $[10, 20, 30]$



∴ For $n=3$, the result is 5.

∴ The catalan number gives us the result.

code →

```
1 class Solution {
2 public:
3
4     int uniqueBST(int n, vector<int>& memo)
5     {
6         if(n==0 || n==1) return 1;
7
8         if(memo[n] != -1) return memo[n];
9
10        int ans = 0;
11        for(int i=0; i<n; i++)
12            ans += uniqueBST(i, memo) * uniqueBST(n-i-1, memo);
13
14        return memo[n] = ans;
15    }
16
17    int numTrees(int n) {
18        vector<int> memo(n+1, -1);
19        return uniqueBST(n, memo);
20    }
21 };
```

Find the rest on

<https://linktr.ee/KarunKarthik>

Follow **Karun Karthik** For More Amazing Content !