19/01/2024

DYNAMIC PROGRAMMING
CLASS - 7

1. Partition Equal Subset Sum (Leetcode-416)

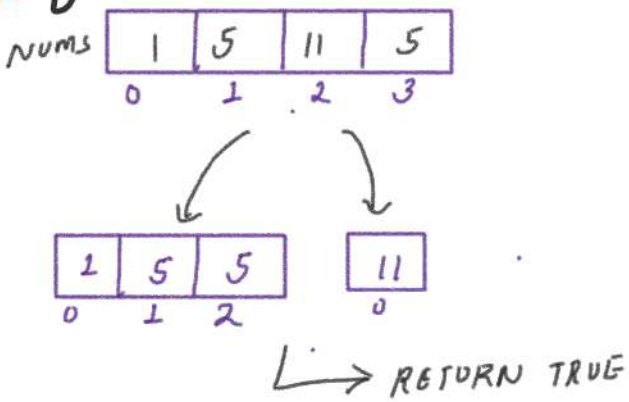
Include and exclude pattern

Problem Statement:
Given an integer array **nums**, return **true** if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or **false** otherwise.

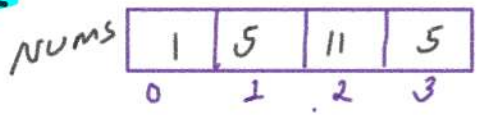
Example 1:
Input: nums = [1,5,11,5]
Output: true

Explanation:

way 1



way 2



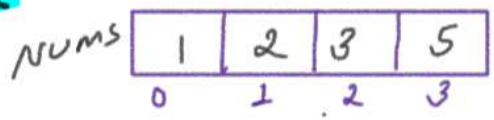
sum = 1 + 5 + 11 + 5
= 22/2

We can divide sum into two part when it is even.
↳ RETURN TRUE

Example 2:
Input: nums = [1,2,3,5]
Output: false

Explanation:

way 2

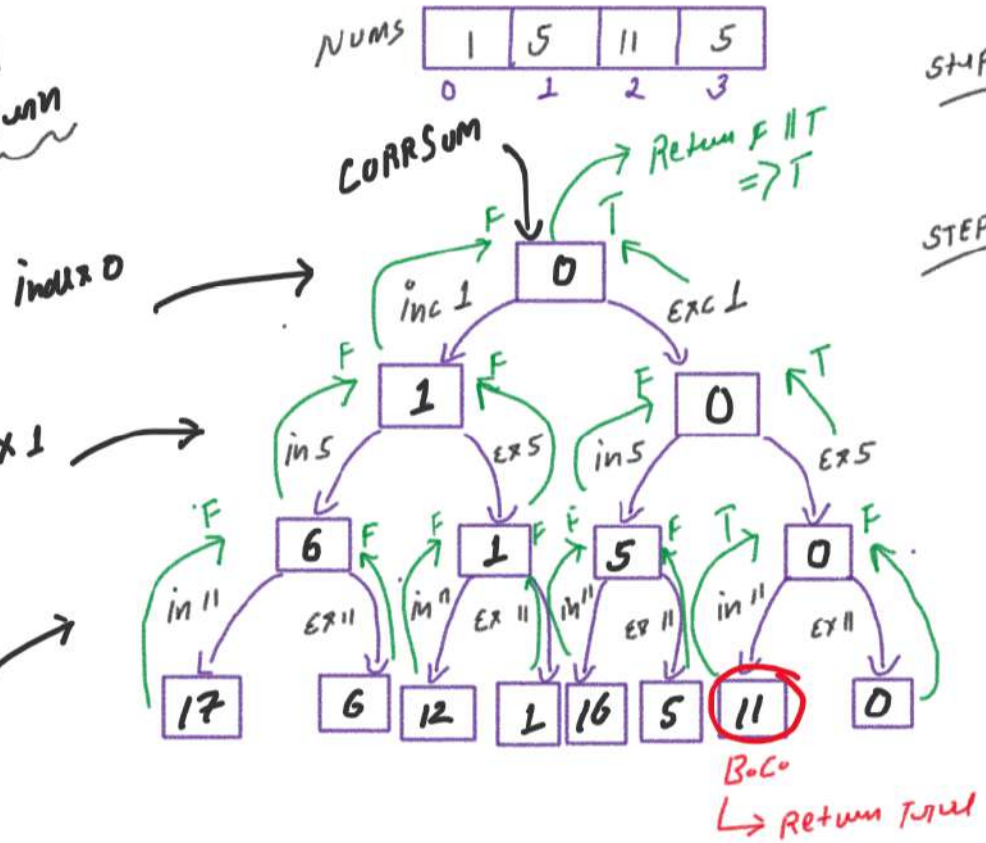


sum = 1+2+3+5
= 11

We can divide sum into two part when it is even.
↳ RETURN FALSE

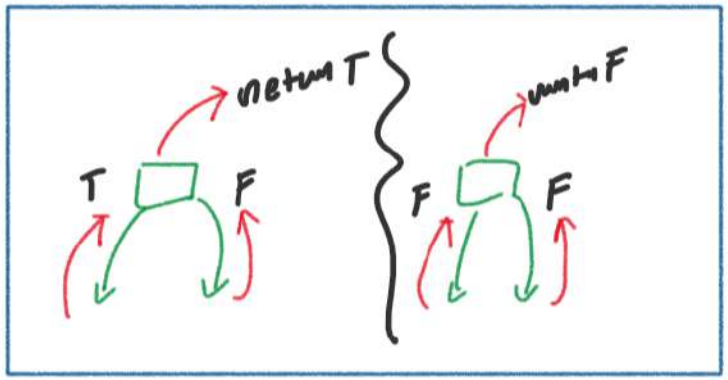
BUILD LOGIC

Include and exclude pattern



STEP1 Target = 1+5+11+5
= 22/2
= 11

STEP2 Find Target using sum of Elements of array



```cpp
// 1. Partition Equal Subset Sum (Leetcode-416)
// Approach 1: Normal Recursion Approach

class Solution {
public:
    bool solveUsingRec(vector<int> &nums, int index, int currSum, int target){
        // Base case
        if(index >= nums.size()){
            return false;
        }
        if(currSum > target){
            return false;
        }
        if(currSum == target){
            return true;
        }

        // Recursive call
        bool include = solveUsingRec(nums, index+1, currSum+nums[index], target);
        bool exclude = solveUsingRec(nums, index+1, currSum+0, target);
        bool ans = include || exclude;
        return ans;
    }

    bool canPartition(vector<int>& nums) {
        int totalSum = 0;
        for(int i=0; i<nums.size(); i++){
            totalSum += nums[i];
        }

        if(totalSum & 1){
            // Odd can't be partioned
            return false;
        }

        int target = totalSum/2;
        int index = 0;
        int currSum = 0;
        bool ans = solveUsingRec(nums, index, currSum, target);
        return ans;
    }
};
```

TLE

```cpp
// 1. Partition Equal Subset Sum (Leetcode-416)
// Approach 2: Top Down Approach

class Solution {
public:
    bool solveUsingMemo(vector<int> &nums, int index, int currSum, int target,
vector<vector<int>> &dp){
        // Base case
        if(index >= nums.size()){
            return false;
        }
        if(currSum > target){
            return false;
        }
        if(currSum == target){
            return true;
        }

        // Step 3: if ans already exist then return ans
        if(dp[index][currSum] != -1){
            return dp[index][currSum];
        }

        // Step 2: store ans and return ans using DP array
        // Recursive call
        bool include = solveUsingMemo(nums, index+1, currSum+nums[index], target, dp);
        bool exclude = solveUsingMemo(nums, index+1, currSum+0, target, dp);
        dp[index][currSum] = include || exclude;
        return dp[index][currSum];
    }

    bool canPartition(vector<int>& nums) {
        int totalSum = 0;
        for(int i=0; i<nums.size(); i++){
            totalSum += nums[i];
        }

        if(totalSum & 1){
            // Odd can't be partioned
            return false;
        }

        int target = totalSum/2;
        int index = 0;
        int currSum = 0;
        // Step 1: create DP array
        vector<vector<int>> dp(nums.size()+1, vector<int> (target+1, -1));
        bool ans = solveUsingMemo(nums, index, currSum, target, dp);
        return ans;
    }
};
```

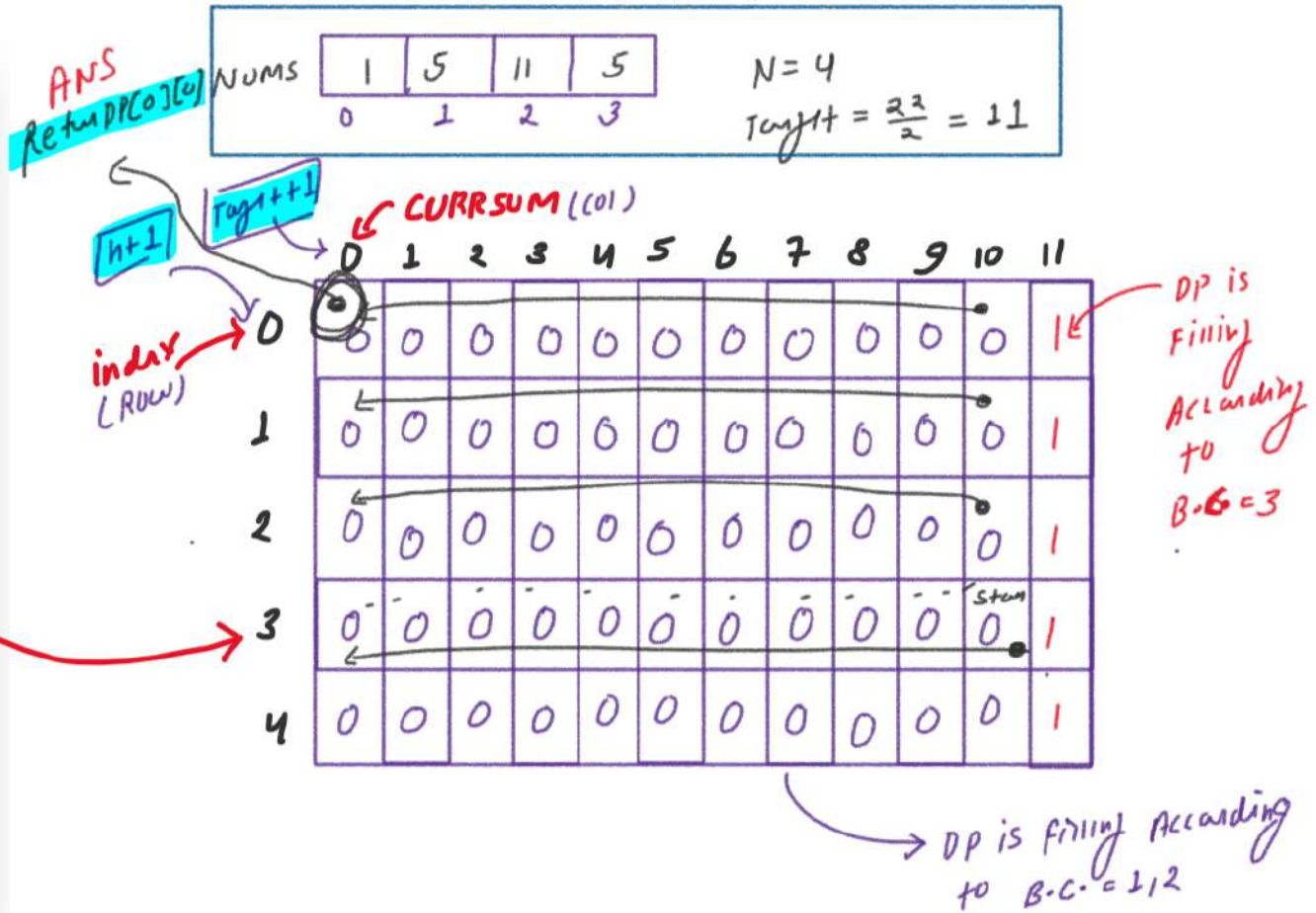
NO TLE

B-S-1

B-S-2

B-S-3

```cpp
// 1. Partition Equal Subset Sum (Leetcode-416)
// Approach 3: Bottom-up

class Solution {
public:
    bool solveUsingTabu(vector<int> &nums, int target){
        // Step 1: create DP array
        // Step 2: fill initial data in DP array according to recursion base case
        int n = nums.size();
        vector<vector<int>> dp(n+1, vector<int> (target+1, 0));
        for(int row=0; row<=n; row++){
            dp[row][target] = 1;
        }

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int index = n-1; index >= 0; index--){
            for(int currSum = target; currSum >= 0; currSum--){
                // Recursive call
                bool include = 0;
                if(currSum+nums[index] <= target){
                    include = dp[index+1][currSum+nums[index]];
                }
                bool exclude = dp[index+1][currSum+0];
                dp[index][currSum] = include || exclude;
            }
        }
        return dp[0][0];
    }

    bool canPartition(vector<int>& nums) {
        int totalSum = 0;
        for(int i=0; i<nums.size(); i++){
            totalSum += nums[i];
        }

        if(totalSum & 1){
            // Odd can't be partioned
            return false;
        }

        int target = totalSum/2;
        bool ans = solveUsingTabu(nums, target);
        return ans;
    }
};
```



Optima Solution

ANS · DeRuals

DP[index][currSum]

curr row

same column

next row

Any Colom

DP[index+1][currSum + nums[index]]

next row

same colum

DP[index+1][currSum]

Return next[0]

next curr

index (row) → 0

next curr 1

index → curr[] → next 2

index+1 → next[] → 3

4

size = target + 1

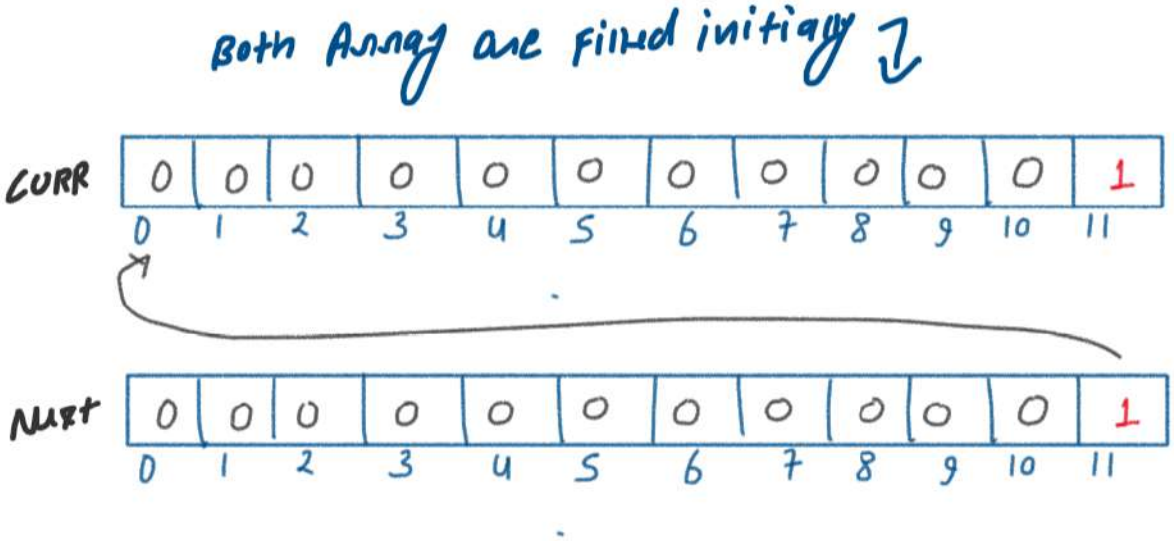
currSum (col)

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1

curr[target] = 1

next[target] = 1

```cpp
// 1. Partition Equal Subset Sum (Leetcode-416)
// Approach 4: Space Optimization Approach
// Without inter changing loop

class Solution {
public:
    bool solveUsingTabuOS(vector<int> &nums, int target){
        int n = nums.size();
        vector<int> curr(target+1,0);
        vector<int> next(target+1,0);

        curr[target] = 1;
        next[target] = 1;

        // Loop row wise hi chalana hai mujhe
        for(int index = n-1; index >= 0; index--){
            for(int currSum = target; currSum >= 0; currSum--){
                // Recursive call
                bool include = 0;
                if(currSum+nums[index] <= target){
                    include = next[currSum+nums[index]];
                }
                bool exclude = next[currSum+0];
                curr[currSum] = include || exclude;
            }
            // Shift Karna Bhool Jata hu
            next = curr;
        }
        return next[0];
    }

    bool canPartition(vector<int>& nums) {
        int totalSum = 0;
        for(int i=0; i<nums.size(); i++){
            totalSum += nums[i];
        }

        if(totalSum & 1){
            // Odd can't be partioned
            return false;
        }

        int target = totalSum/2;
        bool ans = solveUsingTabuOS(nums, target);
        return ans;
    }
};
```



2. Number of Dice Rolls With Target Sum (Leetcode-1155)

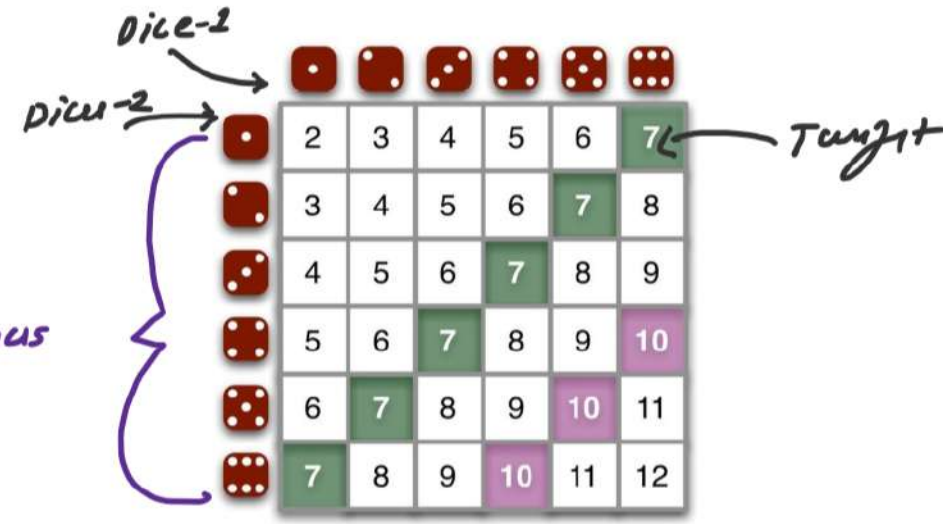
Explore All Possible Ways Pattern

Problem Statement:
You have **n dice**, and each dice has **k faces** numbered from **1** to **k**.

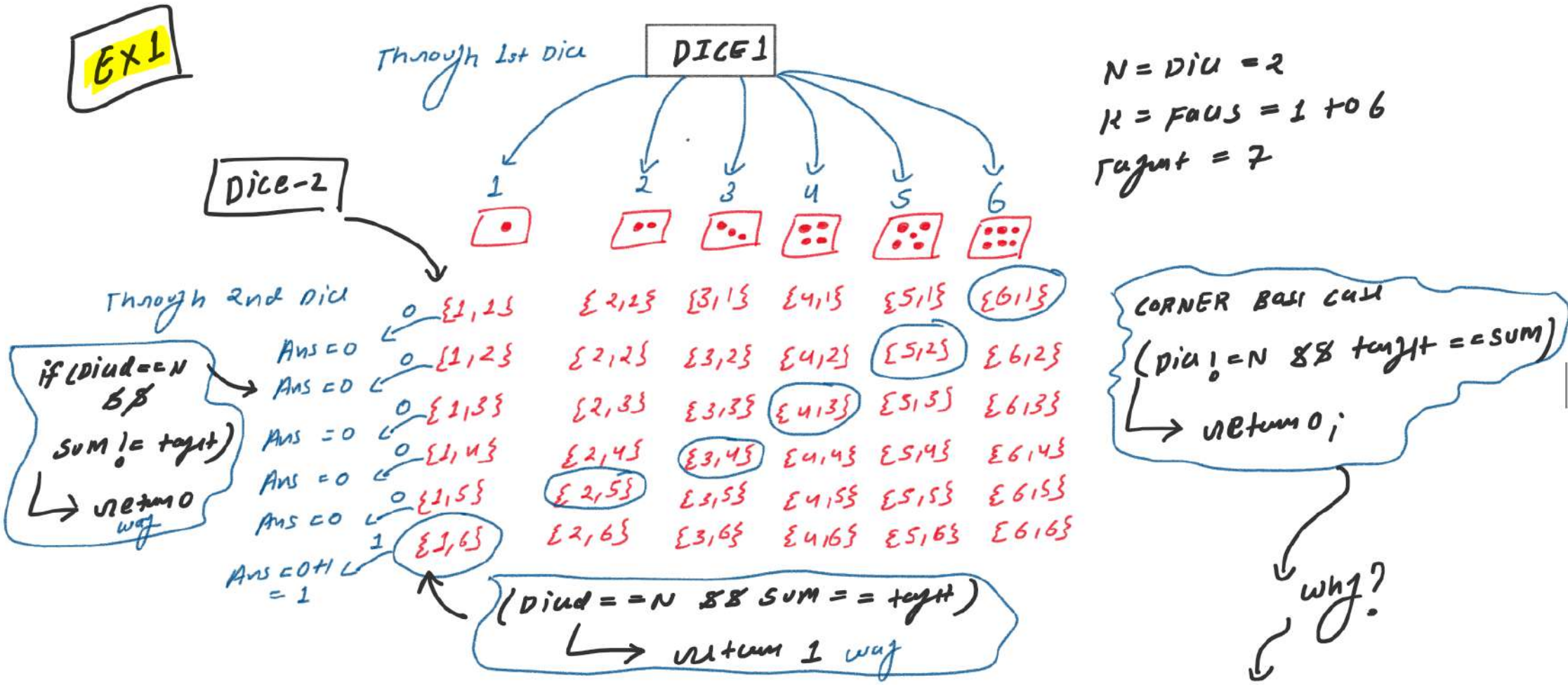
Given three integers **n**, **k**, and **target**, return the number of possible ways (**out of the k^n total ways**) to roll the dice, so the sum of the face-up numbers equals target. Since the answer may be too large, **return it modulo 10^9 + 7**.

Example 1:
Input: n = 2, k = 6, target = 7
Output: 6

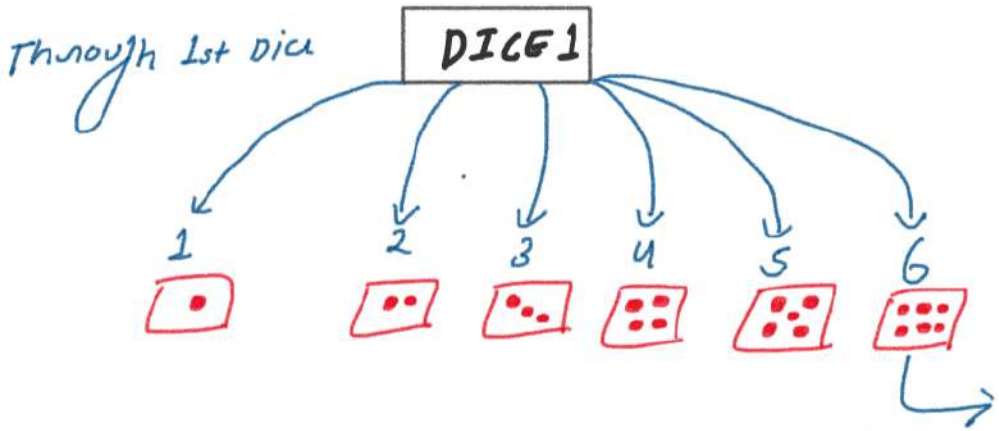
Explanation: You throw two dice, each with 6 faces.
There are 6 ways to get a sum of 7: 1+6, 2+5, 3+4, 4+3, 5+2, 6+1.



when Target = 10
↳ Output ⇒ 3
↳ sum of 10 : 4+6, 5+5, 6+4


EX1
Through 1st Dice
DICE1
N = Dice = 2
K = Faces = 1 to 6
Target = 7
Dice-2
1
2
3
4
5
6
Through 2nd Dice
{1,1} {2,1} {3,1} {4,1} {5,1} {6,1}
{1,2} {2,2} {3,2} {4,2} {5,2} {6,2}
{1,3} {2,3} {3,3} {4,3} {5,3} {6,3}
{1,4} {2,4} {3,4} {4,4} {5,4} {6,4}
{1,5} {2,5} {3,5} {4,5} {5,5} {6,5}
{1,6} {2,6} {3,6} {4,6} {5,6} {6,6}
Ans = 0
Ans = 0
Ans = 0
Ans = 0
Ans = 0
Ans = 0+1 = 1
if (Dice==N && sum != target)
return 0 way
(Dice==N && sum == target)
return 1 way
CORNER BASE CASE
(Dice != N && target <= sum)
return 0;
why?

Explain CORNER Base Case



N = Dice = 2

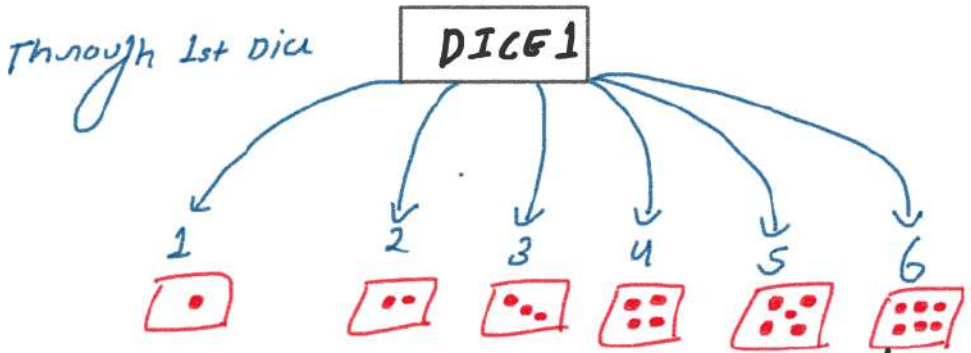
K = Faces = 1 to 6

Target = 6

Target found for return 0 way kyunki MUJHE second dice ke face ko add karke batana tha ki wo Target ke equal honi chaiye.

```
if (Target == sum && Dice_0 != N)
    → Return 0
```

Ex-2



N = Dice = 2

K = Faces = 6

Target = 6

Output => 1

(Target == Sum && Dice == N)

↳ Return 1

REC

```cpp
// 2. Number of Dice Rolls With Target Sum (Leetcode-1155)
// Approach 1: Normal Recursion Approach (TLE)

class Solution {
public:
    long long int MOD = 1000000000+7;
    int solveUsingRec(int n, int k, int target, int usedDice, int currSum){
        //Base case
        if(currSum == target && usedDice == n){
            return 1;
        }
        if(currSum != target && usedDice == n){
            return 0;
        }
        if(currSum == target && usedDice != n){
            return 0;
        }

        // Recursive call
        int ans = 0;
        for(int face = 1; face <= k; face++){
            ans = ((ans)%MOD + solveUsingRec(n, k, target, usedDice+1, currSum+face)%MOD)%MOD;
        }
        return ans;
    }
    int numRollsToTarget(int n, int k, int target) {
        int usedDice = 0;
        int currSum = 0;
        int ans = solveUsingRec(n, k, target, usedDice, currSum);
        return ans;
    }
};
```

CORNER B.C

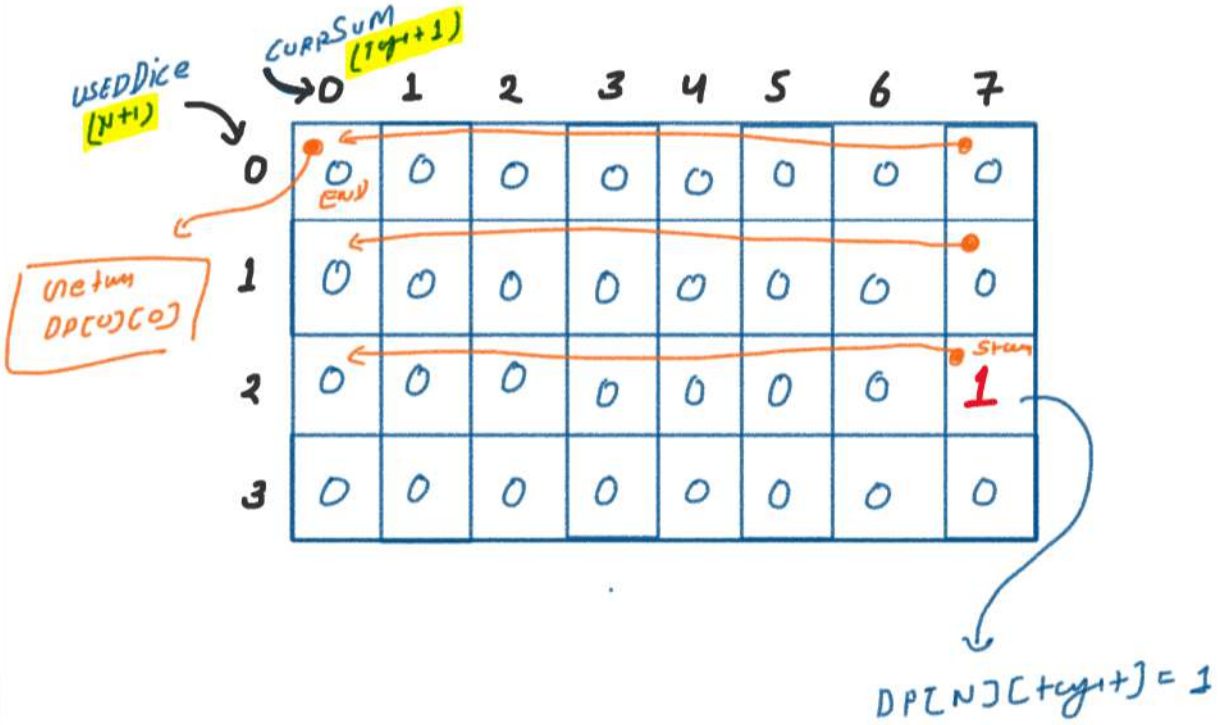
TOP DOWN

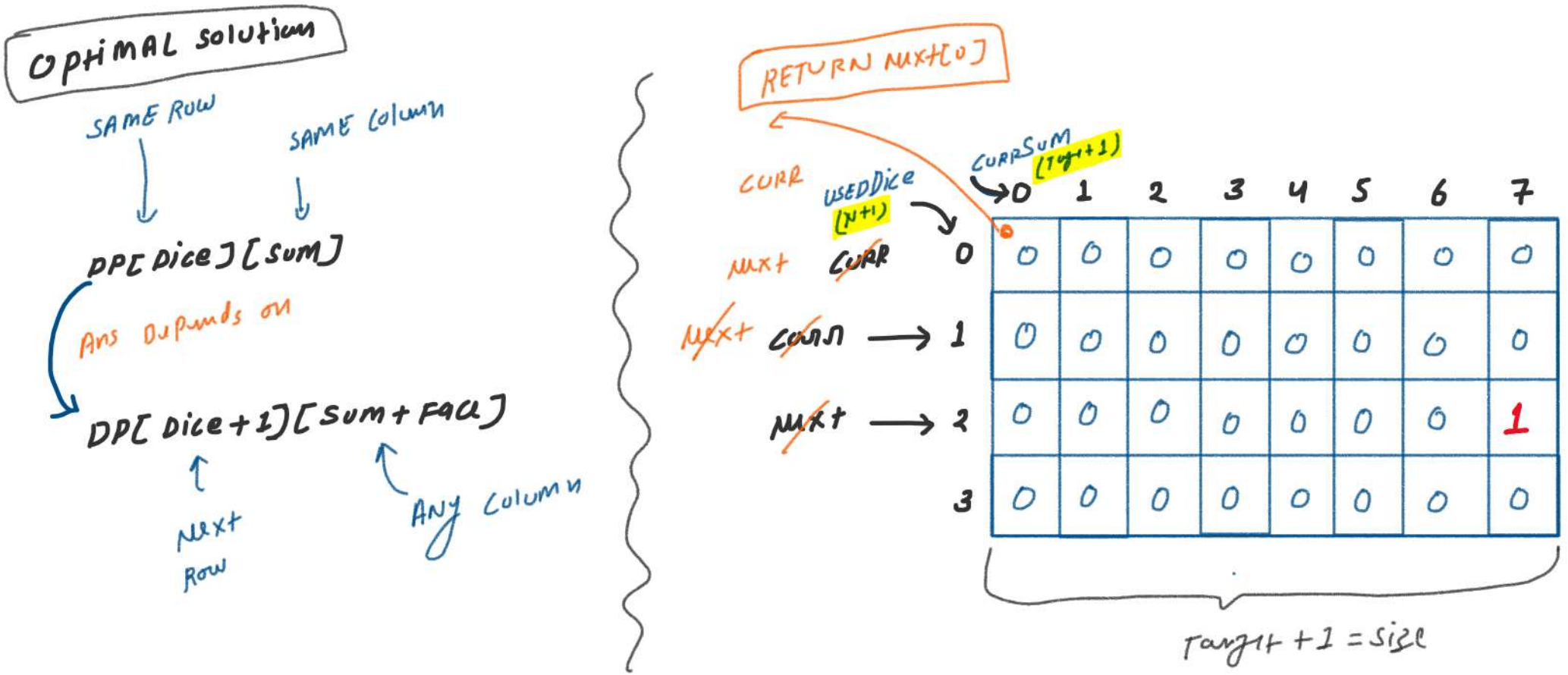
```cpp
// 2. Number of Dice Rolls With Target Sum (Leetcode-1155)
// Approach 2: Top Down Approach (No TLE)

class Solution {
public:
    long long int MOD = 1000000007;
    int solveUsingMemo(int n, int k, int target, int usedDice, int currSum, vector<vector<int>> &dp){
        //Base case
        if(currSum == target && usedDice == n){
            return 1;
        }
        if(currSum != target && usedDice == n){
            return 0;
        }
        if(currSum == target && usedDice != n){
            return 0;
        }

        // Step 3: if ans already exist then return ans
        if(dp[usedDice][currSum] != -1){
            return dp[usedDice][currSum];
        }

        // Step 2: store ans and return ans using DP array
        // Recursive call
        int ans = 0;
        for(int face = 1; face <= k; face++){
            int recAns = 0;
            if(currSum+face <= target){
                // Corner Case -> resolve runtime error occurs due to array index out of bound
                recAns = solveUsingMemo(n, k, target, usedDice+1, currSum+face, dp);
            }
            ans = ((ans)%MOD + (recAns)%MOD)%MOD;
        }
        dp[usedDice][currSum] = ans;
        return dp[usedDice][currSum];
    }
    int numRollsToTarget(int n, int k, int target) {
        int usedDice = 0;
        int currSum = 0;
        // Step 1: create DP array
        vector<vector<int>> dp(n+1, vector<int> (target+1, -1));
        int ans = solveUsingMemo(n, k, target, usedDice, currSum, dp);
        return ans;
    }
};
```

BOTTOM UP

```cpp
// 2. Number of Dice Rolls With Target Sum (Leetcode-1155)
// Approach 3: Bottom-up (NO TLE)

class Solution {
public:
    long long int MOD = 1000000007;
    int solveUsingTabu(int n, int k, int target){
        // Step 1: create DP array
        // Step 2: Initially, filled dp array according to base case 2, 3
        vector<vector<int>> dp(n+1, vector<int> (target+1, 0));
        // Step 2: Initially, filled dp array according to base case 1
        dp[n][target] = 1;

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int usedDice = n-1; usedDice >= 0; usedDice--){
            for(int currSum = target; currSum >= 0; currSum--){
                // Recursive call
                int ans = 0;
                for(int face = 1; face <= k; face++){
                    int recAns = 0;
                    if(currSum+face <= target){
                        // Corner Case
                        recAns = dp[usedDice+1][currSum+face];
                    }
                    ans = ((ans)%MOD + (recAns)%MOD)%MOD;
                }
                dp[usedDice][currSum] = ans;
            }
        }
        return dp[0][0];
    }

    int numRollsToTarget(int n, int k, int target) {
        int ans = solveUsingTabu(n, k, target);
        return ans;
    }
};
```

Dice => N = 2
Face => K = 6
Target = 7




Optimal solution
SAME ROW
SAME Column
DP[Dice][sum]
Ans Depends on
DP[Dice+1][sum + FACE]
Next Row
ANY Column
RETURN NXT[0]
CURR
USEDDice (N+1)
CURRSUM (Target+1)
NXT
CURR
NXT
count
NXT
0 1 2 3 4 5 6 7
0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 1
3 0 0 0 0 0 0 0 0
Target +1 = size

```cpp
// 2. Number of Dice Rolls With Target Sum (Leetcode-1155)
// Approach 4: Space Optimization Approach (NO TLE)

class Solution {
public:
    long long int MOD = 1000000007;
    int solveUsingTabu(int n, int k, int target){
        // Initially, filled dp array according to base case 2, 3
        vector<int> curr(target+1, 0);
        vector<int> next(target+1, 0);
        // Initially, filled dp array according to base case 1
        next[target] = 1;


        for(int usedDice = n-1; usedDice >= 0; usedDice--){
            for(int currSum = target-1; currSum >= 0; currSum--){
                // Recursive call
                int ans = 0;
                for(int face = 1; face <= k; face++){
                    int recAns = 0;
                    if(currSum+face <= target){
                        // Corner Case
                        recAns = next[currSum+face];
                    }
                    ans = ((ans)%MOD + (recAns)%MOD)%MOD;
                }
                curr[currSum] = ans;
            }
            // Shift Karna Bhool Jata hu
            next = curr;
        }
        return next[0];
    }

    int numRollsToTarget(int n, int k, int target) {
        int ans = solveUsingTabu(n, k, target);
        return ans;
    }
};
```