**Pattern: Dp on sub sequences/ Subsets**

How to approach every \*\*dp on subsequence\*\* pattern problems**:**

* Express-> will be expressed in terms of (ind, target)
* Explore-> NotTake - take for every a[i] will be used
* Get the required ans

**Problem link**- [Subset Sum Equals to K](https://www.codingninjas.com/codestudio/problems/subset-sum-equal-to-k_1550954?leftPanelTab=0)

\*\***1. recursive- solution: TC =** O(2^n), we have 2 option for every element pick- not pick

**//SC** = O(N) recursion stack space

**bool f(int ind, int target, vector<int> &arr){**

**if(target==0)**

**return true;**

**if(ind==0)**

**return (arr[0]==target);**

**//explore**

**bool notTake = f(ind-1, target, arr);**

**bool take = false;**

**if(arr[ind] <= target)**

**take = f(ind-1, target-arr[ind], arr);**

**//return t/f**

**return (notTake || take);**

**}**

**bool subsetSumToK(int n, int k, vector<int> &arr) {**

**return f(n-1, k, arr);**

**}**

\*\***2. DP- memoization solution: //tc =** O(n\*k) for all new calls

// **sc** = O(n\*k) + O(n) for recursion stack space

**//memoization solution**

**int f(int ind, int target, vector<int> &arr, vector<vector<int>> &dp){**

**if(target==0)**

**return 1;**

**if(ind==0)**

**return (arr[0]==target);**

**if(dp[ind][target] != -1) return dp[ind][target];**

**//explore**

**int notTake = f(ind-1, target, arr, dp);**

**int take = 0;**

**if(arr[ind] <= target)**

**take = f(ind-1, target-arr[ind], arr, dp);**

**//return t/f**

**return dp[ind][target] = (notTake || take);**

**}**

**bool subsetSumToK(int n, int k, vector<int> &arr) {**

**//dp[n+1][k+1]**

**vector<vector<int>> dp(n+1, vector<int>(k+1, -1));**

**return f(n-1, k, arr, dp);**

**}**

\*\***3. DP- tabulation: //TC = O(n\*k),**

**// SC = O(n\*k)**

**//DP\_tabulation (bottom- up approach)**

**//DP\_tabulation (memoization -> tabulation)**

**bool subsetSumToK(int n, int k, vector<int> &arr) {**

**//dp[n+1][k+1]**

**vector<vector<bool>> dp(n+1, vector<bool>(k+1, false));**

**//build base in dp with base cases**

**for(int i=0; i<n; i++)**

**dp[i][0] = true; //target==0, return 1;**

**dp[0][arr[0]] = true; //ind==0, return a[ind]==target**

**//at index 0, we can achieve only target == a[0]**

**//use nested for loops to build dp from bottom to up**

**for(int ind=1; ind<n; ind++){**

**for(int target= 1; target<=k; target++){**

**//explore**

**int notTake = dp[ind-1] [target];**

**int take = false;**

**if(arr[ind] <= target)**

**take = dp[ind-1] [target-arr[ind]];**

**dp[ind][target] = (notTake || take);**

**}**

**}**

**return dp[n-1][k];**

**}**

\***\*4. DP- optimized space:** //TC = O(n\*k),

// SC = O(2k)

**//use a prev[k] to keep track of [i-1]’s [target] & [target- arr[i]] values and use curr[k] for storing current row values.**

**//DP\_tabulation (space- optimized)**

**//DP\_tabulation (space optimization)**

**bool subsetSumToK(int n, int k, vector<int> &arr) {**

**///prev will store the (ind-1) row to get values of [ind-1][target] and [ind-1][target-arr[ind]] = prev[k+1]**

**vector<bool> prev(k+1, 0), curr(k+1, 0);**

**prev[0] = curr[0] = 1; //target = 0, return 1;**

**prev[arr[0]] = 1; //ind=0, prev[arr[0]] = 1**

**//use nested for loops to build dp from bottom to up**

**for(int ind=1; ind<n; ind++){**

**for(int target= 1; target<=k; target++){**

**//explore**

**int notTake = prev[target];**

**int take = false;**

**if(arr[ind] <= target)**

**take = prev[target-arr[ind]];**

**curr[target] = (notTake || take);**

**}**

**//update prev**

**prev = curr;**

**}**

**return prev[k];**

**}**