**DP on Subsequence**

Q.1 <https://naukri.com/code360/problems/subset-sum-equal-to-k_1550954?leftPanelTabValue=PROBLEM>

* So here what basically used the logic of pick and not to pick for recursion and the base condition is when target ==0 then return 1
* Also if ind==0 and arr[0]==target return 1 else 0
* Call on pick to not pick return OR of it ||
* For transition into memorization check how many parameter are changing according to that declare dp array
* Same for tabulation check how many state are changing that many nested for loop will come

|  |
| --- |
| #include <bits/stdc++.h>  bool helper(vector<int>&arr,int target,int ind,vector<vector<int>> &dp)  {      if(target==0) return true;      if(ind==0)      {          if(arr[0]==target) return true;          return false;      }      if(dp[ind][target]!=-1) return dp[ind][target];      bool nottake=helper(arr,target,ind-1,dp);      bool take=false;      if(target>= arr[ind])      {          take=helper(arr,target-arr[ind],ind-1,dp);      }        dp[ind][target]= take|| nottake;      return dp[ind][target];  }  bool subsetSumToK(int n, int k, vector<int> &arr) {      // Write your code here.      vector<vector<bool>> dp(n,vector<bool>(k+1,0));      // return helper(arr,k,n-1,dp);      for(int i=0;i<n;i++)      {          dp[i][0]=true;      }      dp[0][arr[0]]=true;      for(int ind=1;ind<n;ind++)      {          for(int target=1;target<=k;target++)          {              bool nottake=dp[ind-1][target];              bool take=false;              if(target>=arr[ind])              {                  take=dp[ind-1][target-arr[ind]];              }              dp[ind][target]=take || nottake;          }      }      return dp[n-1][k];  } |

Recursion : TC:-O(2^n) SC:O(N)

Memo : TC:-O(N\*target) SC:O(N\*target) dp+O(N) recursion stack space

Tab: TC:O(N\*target) SC:O(N\*target)

Q.2 <https://leetcode.com/problems/partition-equal-subset-sum/description/>

* Here we have to partition the array into subset such that both have the same some
* Logic 🡪 partition in 2 subset 🡪 so if the whole array sum suppose s then the one partition will have sum as s/2
* If the total\_sum is odd then we can make it happen to partition in 2 subset or subsequence
* So if one subsequence has sum s/2 it return true then other will also have s/2
* After that use logic of subset of sum equal to k

|  |
| --- |
| class Solution {  public:      bool helper(vector<int>&arr,int target,int ind,vector<vector<int>> &dp)  {      if(target==0) return true;      if(ind==0)      {          return arr[0]==target;      }      if(dp[ind][target]!=-1) return dp[ind][target];      bool nottake=helper(arr,target,ind-1,dp);      bool take=false;      if(arr[ind]<=target)      {          take=helper(arr,target-arr[ind],ind-1,dp);      }          return dp[ind][target]= take|| nottake;  }      bool subsetSumToK(int n, int k, vector<int> &arr) {      // Write your code here.      vector<vector<bool>> dp(n,vector<bool>(k+1,0));      // return helper(arr,k,n-1,dp);      for(int i=0;i<n;i++)      {          dp[i][0]=true;      }      if(arr[0]<=k) dp[0][arr[0]]=true;      for(int ind=1;ind<n;ind++)      {          for(int target=1;target<=k;target++)          {              bool nottake=dp[ind-1][target];              bool take=false;              if(target>=arr[ind])              {                  take=dp[ind-1][target-arr[ind]];              }              dp[ind][target]=take || nottake;          }      }      return dp[n-1][k];     }      bool canPartition(vector<int>& nums) {            int n=nums.size();          int total\_sum=0;          for(int i=0;i<n;i++) total\_sum+=nums[i];            // because we can't devide it into 2 equal halfs          if(total\_sum % 2==1) return false;          else          {          int target=total\_sum/2;          vector<vector<int>> dp(n,vector<int>(target+1,-1));          // return helper(nums,target,n-1,dp);          return subsetSumToK(n,target,nums);          }        }  }; |

Recursion : TC:-O(2^n) +O(N) SC:O(N)

Memo : TC:-O(N\*target)+O(N) SC:O(N\*target) dp+O(N) recursion stack space

Tab: TC:O(N\*target) +O(N) SC:O(N\*target)

Q.3 <https://www.naukri.com/code360/problems/partition-a-set-into-two-subsets-such-that-the-difference-of-subset-sums-is-minimum_842494?leftPanelTabValue=PROBLEM>

🡪so In this question we have to find subset of array whos difference is minimum

* The main logic here is we are going to use tabulation code of subset sum equal target question
* Why because the last row of dp array signify the sum is possible for that index till n
* Iterate through last row till s/2 because the sum is going to reapeat after s/2 because both the subset will have same value and check which sum are true findout absolute difference and return it.

|  |
| --- |
| int minSubsetSumDifference(vector<int>& arr, int n)  {      // Write your code here.            int sum=0;          for(int i=0;i<n;i++)          {              sum+=arr[i];          }          vector<vector<bool>> dp(n,vector<bool>(sum+1,false));         for(int i=0;i<n;i++)          {              dp[i][0]=1;          }          if(arr[0]<=sum) dp[0][arr[0]]=1;          for(int i=1;i<n;i++)          {              for(int k=1;k<=sum;k++)              {                  bool pick=false;                  if(arr[i]<=k) pick=dp[i-1][k-arr[i]];                  bool notpick=dp[i-1][k];                  dp[i][k]=pick || notpick;              }          }          int mini=1e9;          for(int s1=0;s1<=sum/2;s1++)          {              if(dp[n-1][s1]==true)mini=min(mini,abs(s1-(sum-s1)));          }          return mini;  } |

Tab: TC:O(N\*Sum) +O(N) +O(Sum/2) SC:O(N\*Sum)

Q.4 <https://www.naukri.com/code360/problems/count-subsets-with-sum-k_3952532>

* Use same logic for subset sum==k
* Just use index start it from zero here because there are some test case which won’t work for reverse index fashion
* Base case target < 0 return zero
* If i==N and target==0 return 1 else return 0
* Pick and not pick
* Return sum of it

|  |
| --- |
| int helper(vector<int> &arr,int ind,int n,int target,int mod,vector<vector<int>> &dp)  {      if(target<0) return 0;        if(ind==n)      {          if(target==0) return 1;          return 0;      }      if(dp[ind][target]!=-1) return dp[ind][target];      int pick=0;      if(arr[ind]<=target) pick=helper(arr,ind+1,n,target-arr[ind],mod,dp);      pick+=helper(arr,ind+1,n,target,mod, dp);        dp[ind][target]=(pick)%mod;      return dp[ind][target];  }  int findWays(vector<int>& arr, int k)  {      // Write your code here.      const int mod = 1000000007;      int n=arr.size();      vector<vector<int>> dp(n+1,vector<int>(k+1,0));      // return helper(arr,0,n,k,mod,dp);      dp[0][0]=1;      for(int i=1;i<=n;i++)      {          for(int target=0;target<=k;target++)          {              int notpick=dp[i-1][target];              int pick=0;              if(arr[i-1]<=target)              {                  pick=dp[i-1][target-arr[i-1]];              }              dp[i][target]=(pick+notpick)%mod;          }      }      return dp[n ][k];  } |

Recursion : TC:-O(2^n) SC:O(N)

Memo : TC:-O(N\*target) SC:O(N\*target) dp+O(N) recursion stack space

Tab: TC:O(N\*target) SC:O(N\*target)

Q.5 <https://www.naukri.com/code360/problems/partitions-with-given-difference_3751628>

* From the 2 equation you can built the logic first s1-s2=d and s1+s2=sum so s1=sum+d /2
* The base condition is when ind==0 and sum==0 and arr[ind]==0 return true
* If sum==0 || sum==arr[ind] return 1
* Rest logic same as pick and not pick

|  |
| --- |
| #include <bits/stdc++.h>  #define mod 1000000007  int helper(int ind,vector<int> &arr,int sum,vector<vector<int>> &dp)  {      // if(sum==0) return 1;      if(ind==0)      {          if(sum==0 && arr[ind]==0) return 2;          if(sum==0 || sum==arr[ind]) return 1;          return 0;      }      if(dp[ind][sum]!=-1) return dp[ind][sum];      int pick=0;      if(arr[ind]<=sum)pick=helper(ind-1,arr,sum-arr[ind],dp);      int notpick=helper(ind-1,arr,sum,dp);      return dp[ind][sum]=(pick+notpick)%mod;  }  int countPartitions(int n, int d, vector<int> &arr) {      // Write your code here.        int sum=0;      for(int i=0;i<n;i++)      {          sum+=arr[i];      }      if((sum+d)%2==1) return 0;      int s1=(sum+d)/2;      vector<vector<int>> dp(n,vector<int>(s1+1,0));      // return helper(n-1,arr,s1,dp);       if(arr[0]==0) dp[0][0]=2;     else dp[0][0]=1;     if(arr[0]!=0 && arr[0]<=s1) dp[0][arr[0]]=1;      for(int i=1;i<n;i++)      {          for(int k=0;k<=s1;k++)          {          //  if(i==0 && k==0) continue;      int pick=0;      if(arr[i]<=k)pick=dp[i-1][k-arr[i]];      int notpick=0;       notpick=dp[i-1][k];      dp[i][k]=(pick+notpick)%mod;          }      }      return dp[n-1][s1];  } |

Recursion : TC:-O(2^n) SC:O(N)

Memo : TC:-O(N\*target) SC:O(N\*target) dp+O(N) recursion stack space

Tab: TC:O(N\*target) SC:O(N\*target)

Q.6 <https://leetcode.com/problems/coin-change/description/>

* Main logic is we can pick up single element multiple time so here in pick we don’t need jump on index we have to stay on same index and for not pick we can jump to next index
* Base case ind==0 then if amount%coins[ind] return amount/coins[ind] else return 1e9
* Write pick and not pick condition and return min of it
* For counting denomination make sure to add 1 in each pick step

|  |
| --- |
| class Solution {  public:      int helper(int ind,vector<int>&coins,int amount,vector<vector<int>> &dp)      {          if(ind==0)          {              if(amount%coins[ind]==0) return amount/coins[ind];              else{                  return 1e9;              }          }          if(dp[ind][amount]!=-1) return dp[ind][amount];          int pick=1e9;              if(coins[ind]<=amount) pick=1+helper(ind,coins,amount-coins[ind],dp);          int notpick=helper(ind-1,coins,amount,dp);            return dp[ind][amount]=min(pick,notpick);      }      int coinChange(vector<int>& coins, int amount) {         int n=coins.size();         vector<vector<int>> dp(n,vector<int>(amount+1,0));         //int ans=helper(n-1,coins,amount,dp);          for(int t=0;t<=amount;t++)          {              if(t%coins[0]==0)              {                  dp[0][t]=t/coins[0];              }              else              {                  dp[0][t]=1e9;              }          }          for(int i=1;i<n;i++)          {              for(int t=0;t<=amount;t++)              {                  int pick=1e9;                  if(coins[i]<=t) pick=1+dp[i][t-coins[i]];                  int notpick=dp[i-1][t];                  dp[i][t]=min(pick,notpick);              }          }          int ans=dp[n-1][amount];          if(ans!=1e9) return ans;          return -1;          }  }; |

Recursion : TC:-O(2^n) SC:O(N)

Memo : TC:-O(N\*target) SC:O(N\*target) dp+O(N) recursion stack space

Tab: TC:O(N\*target) SC:O(N\*target)