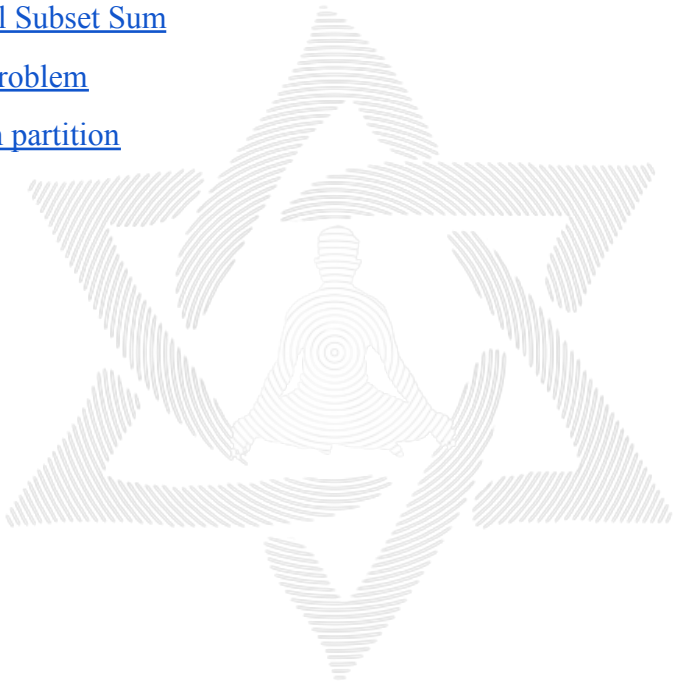
DP
Lesson 4

1. 0 - 1 Knapsack Problem
2. Subset Sum Problem
3. Partition Equal Subset Sum
4. Perfect Sum Problem
5. Minimum sum partition
6. Target-sum



0 - 1 Knapsack Problem

You are given weights and values of **N** items, put these items in a knapsack of capacity **W** to get the maximum total value in the knapsack. Note that we have only **one quantity of each item**.

In other words, given two integer arrays **val[0..N-1]** and **wt[0..N-1]** which represent values and weights associated with **N** items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of **val[]** such that sum of the weights of this subset is smaller than or equal to **W.** You cannot break an item, **either pick the complete item or don't pick it (0-1 property)**.

Example 1:

Input:
N = 3
W = 4
values[] = {1,2,3}
weight[] = {4,5,1}
Output: 3

Example 2:

Input:
N = 3
W = 3
values[] = {1,2,3}
weight[] = {4,5,6}
Output: 0

Expected Time Complexity: O(N*W).

Expected Auxiliary Space: O(N*W)

Constraints:

$1 \leq N \leq 1000$

$1 \leq W \leq 1000$

$1 \leq wt[i] \leq 1000$

$1 \leq v[i] \leq 1000$

```
vector<vector<int>> dp;
int knap(int W, int wt[], int val[], int n)
{
   if(n==0)
      return 0;
   if(W<=0)
      return 0;
   if(dp[W][n]==-1)
   {
   if(wt[n-1]<=W)
      dp[W][n]=max(knap(W,wt,val,n-1),knap(W-wt[n-1],wt,val,n-1)+val[n-1]);
   else
      dp[W][n]=knap(W,wt,val,n-1);
   }
   return dp[W][n];
}
class Solution
{
   public:
   //Function to return max value that can be put in knapsack of capacity W.
   int knapSack(int W, int wt[], int val[], int n)
   {
      // Your code here
      dp.assign(W+1,vector<int>(n+1,-1));
      return knap(W,wt,val,n);
   }
}
```

```cpp
vector<vector<int>> dp;
class Solution
{
  public:
  //Function to return max value that can be put in knapsack of capacity W.
  int knapSack(int W, int wt[], int val[], int n)
  {
    // Your code here
     dp.assign(W+1,vector<int>(n+1));
     for(int i=0; i<=W; i++)
        dp[i][0]=0;
     for(int i=0; i<=n; i++)
        dp[0][i]=0;
     for(int i=1; i<=W; i++)
     {
        for(int j=1; j<=n; j++)
        {
           if(wt[j-1]<=i)
              dp[i][j]=max(dp[i][j-1],dp[i-wt[j-1]][j-1]+val[j-1]);
           else
              dp[i][j]=dp[i][j-1];
        }
     }
     return dp[W][n];
  }
};
```

Subset Sum Problem

Given an array of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

Example 1:

Input:

N = 6

arr[] = {3, 34, 4, 12, 5, 2}

sum = 9

Output: 1

Explanation: Here there exists a subset with sum = 9, 4+3+2 = 9.

Example 2:

Input:

N = 6

arr[] = {3, 34, 4, 12, 5, 2}

sum = 30

Output: 0

Explanation: There is no subset with sum 30.

Expected Time Complexity: O(sum*N)

Expected Auxiliary Space: O(sum*N)

Constraints:

1 <= N <= 100

1<= arr[i] <= 100

1<= sum <= 105

```
class Solution{
public:
    bool isSubsetSum(int n, int nums[], int sum){
        // code here
            int s=0;
            for(int i=0; i<n; i++)
                s+=nums[i];
            bool dp[n+1][s+1];
            memset(dp,0,sizeof(dp));
            for(int i=0; i<=n; i++)
                dp[i][0]=1;
            for(int i=1; i<=n; i++)
            {
                dp[i][nums[i-1]]=1;
                for(int j=1; j<=s; j++)
                {
                    if(dp[i-1][j])
                    {
                        dp[i][j+nums[i-1]]=1;
                        dp[i][j]=1;
                    }
                }
            }
            if(sum>s)
                return false;
            else
                return dp[n][sum];
    }
};
```

Partition Equal Subset Sum

Given an array **arr[]** of size **N**, check if it can be partitioned into two parts such that the sum of elements in both parts is the same.

Example 1:

Input: N = 4

arr = {1, 5, 11, 5}

Output: YES

Explanation:

The two parts are {1, 5, 5} and {11}.

Example 2:

Input: N = 3

arr = {1, 3, 5}

Output: NO

Explaination: This array can never be partitioned into two such parts.

Expected Time Complexity: O(N*sum of elements)

Expected Auxiliary Space: O(N*sum of elements)

Constraints:

$1 \leq N \leq 100$

$1 \leq arr[i] \leq 1000$

```
vector<vector<int>> ans;
int find(int n, int arr[], int s)
{
    if(n==0)
    {
        return 0;
    }
    if(s==0)
        return 1;
    if(s<0)
        return 0;
    if(ans[n][s]==-1)
    {
        if(find(n-1,arr,s-arr[n-1])==1)
            ans[n][s]=1;
        else if(find(n-1,arr,s)==1)
            ans[n][s]=1;
        else
            ans[n][s]=0;
    }
    return ans[n][s];
}
class Solution{
public:
    int equalPartition(int N, int arr[])
    {
        // code here
        int s=0;
        for(int i=0; i<N; i++)
            s+=arr[i];
        if(s%2==1)
            return 0;
        s=s/2;
        ans.assign(N+1,vector<int>(s+2,-1));
        return find(N,arr,s);
    }
};
```

```
vector<vector<int>> ans;
class Solution{
public:
    int equalPartition(int N, int arr[])
    {
        // code here
        int s=0;
        for(int i=0; i<N; i++)
            s+=arr[i];
        if(s%2==1)
            return 0;
        s=s/2;
        //ans.assign(N+1,vector<int>(s+1,-1));
        ans.assign(N+1,vector<bool>(s+1,false));
        //return find(N,arr,s);
        ///*
        for(int i=0; i<=N; i++)
            ans[i][0]=true;
        for(int i=1; i<=N; i++)
        {
            for(int j=1; j<=s; j++)
            {
                if(j-arr[i-1]>=0)
                {
                    ans[i][j]=(ans[i-1][j]||ans[i-1][j-arr[i-1]]);
                }
                else
                    ans[i][j]=ans[i-1][j];
            }
        }
        return ans[N][s];
        //*/
    }
};
```

Perfect Sum Problem

Given an array **arr[]** of integers and an integer **sum**, the task is to count all subsets of the given array with a sum equal to a given **sum**.

Note: Answer can be very large, so, output answer modulo 109+7

Example 1:

Input: N = 6, arr[] = {2, 3, 5, 6, 8, 10} sum = 10
Output: 3
Explanation: {2, 3, 5}, {2, 8}, {10}

Example 2:

Input: N = 5, arr[] = {1, 2, 3, 4, 5} sum = 10
Output: 3
Explanation: {1, 2, 3, 4}, {1, 4, 5}, {2, 3, 5}

Expected Time Complexity: O(**N*sum**)

Expected Auxiliary Space: O(**N*sum**)

Constraints:

$1 \leq$ **N*sum** ≤ 106

```
#define MOD 1000000007
vector<vector<int>> ans;
int perfect(int arr[], int n, int sum)
{
    if(sum==0)
        return 1;
    if(n==0 || sum<0)
        return 0;
    if(ans[n][sum]==-1)
    {
        int x=perfect(arr,n-1,sum-arr[n-1]);
        int y=perfect(arr,n-1,sum);
        ans[n][sum]=(x+y)%MOD;
    }
    return ans[n][sum];
}
class Solution{

        public:
        int perfectSum(int arr[], int n, int sum)
        {
    // Your code goes here
    ans.assign(n+1,vector<int>(sum+1,-1));
    return perfect(arr,n,sum);
        }

};
```

```
#define MOD 1000000007
vector<vector<int>> ans;
class Solution{

        public:
        int perfectSum(int arr[], int n, int sum)
        {
    // Your code goes here
    ans.assign(n+1,vector<int>(sum+1,0));
    for(int i=0; i<=n; i++)
       ans[i][0]=1;
    for(int i=1; i<=n; i++)
    {
       for(int j=1; j<=sum; j++)
       {
          if(arr[i-1]<=j)
          {
             ans[i][j]=(ans[i][j]+ans[i-1][j-arr[i-1]])%MOD;
          }
          ans[i][j]=(ans[i][j]+ans[i-1][j])%MOD;
       }
    }
    return ans[n][sum];
        }

};
```

Minimum sum partition

Given an integer array **arr** of size **N**, the task is to divide it into two sets S1 and S2 such that the absolute difference between their sums is minimum and find the minimum difference

Example 1:

Input: N = 4, arr[] = {1, 6, 11, 5}

Output: 1

Explanation:

Subset1 = {1, 5, 6}, sum of Subset1 = 12

Subset2 = {11}, sum of Subset2 = 11

Example 2:

Input: N = 2, arr[] = {1, 4}

Output: 3

Explanation:

Subset1 = {1}, sum of Subset1 = 1

Subset2 = {4}, sum of Subset2 = 4

Expected Time Complexity: O(**N*|sum of array elements|**)

Expected Auxiliary Space: O(**N*|sum of array elements|**)

Constraints:

1 ≤ **N*|sum of array elements|** ≤ 106

```
vector <vector<int>> ans;
int differ(int arr[], int n, int a, int b)
{
   if(n==0)
   {
      return abs(a-b);
   }
   if(ans[n][a]==-1)
   {
      int x=differ(arr,n-1,a+arr[n-1],b);
      int y=differ(arr,n-1,a,b+arr[n-1]);
      ans[n][a]=min(x,y);
   }
   return ans[n][a];
}
class Solution{

 public:
          int minDifference(int arr[], int n)  {
             int here_sum=0;
             for(int i=0; i<n; i++)
                here_sum+=arr[i];
             ans.assign(n+1,vector<int>(here_sum+1,-1));
             return differ(arr,n,0,0);
           }
};
```