0 – 1 Knapsack Problem

What is 0 – 1 knapsack ?

0 – 1 represents either you choose an item completely or don't choose it and there is no partial choosing or multiple choosing of the same item.

Problem statement

wt[0..n-1] represents weight array, val[0..n-1] represents value array and w represents the total capacity of knapsack. Find the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to w.

0 – 1 Knapsack Recursive

```
int knapsack(int wt[],int val[],int n,int w)
{
        // Base Condition
        if(n==0||w==0)
            return 0;
       // Choice Diagram
      if(wt[n-1]<=w)
            return max(val[n-1]+knapsack(wt,val,n-1,w-wt[n-1]),knapsack(wt,val,n-1,w));
      else
            return knapsack(wt,val,n-1,w);
}
```

0 – 1 Knapsack Memorization or Top Down

```
int knapsack(int wt[],int val[],int n,int w)
{
        // dp can be decleared globlally and initialised with -1
       // Checking weather the element is already present or not
       if(dp[n][w]!=-1)
            return dp[n][w];
```

```
    // Base condition
    if(n==0||w==0)
        dp[n][w]=0;
    // Choice diagram
    if(wt[n-1]<=w)
        dp[n][w]=max(val[n-1]+knapsack(wt,val,n-1,w-wt[n-1]),knapsack(wt,val,n-1,w));
    else
        dp[n][w]=knapsack(wt,val,n-1,w);
    return dp[n][w];
}
```

0 – 1 Knapsack Bottom Up

```
int knapsack(int wt[],int val[],int n,int w)
{
    vector<vector<int>> dp(n+1,vector<int>(w+1,0));
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=w;j++)
        {
            dp[i][j]=dp[i-1][j];
            if(j>=wt[i-1])
                dp[i][j]=max(val[i-1]+dp[i-1][j-wt[i-1]],dp[i-1][j]);
        }
    }
    return dp[n][w];
}
```

Types of 0 – 1 knapsack

1) Subset sum

Problem Statement : We have an arr[0..n-1] and sum return true if we can add some elements of the arr to get sum, otherwise return false (Note 0 – 1 knapsack property must be followed).

Solution :

```
bool solve(int arr[],int n,int sum)
{
        vector<vector<int>> dp(n+1,vector<int>(sum+1,0));
        for(int i=1;i<=n;i++)
        {
                for(int j=1;j<=sum;j++)
                {
                        dp[i][j]=dp[i-1][j];
                        if(j>=arr[i-1])
                                dp[i][j]=max(dp[i-1][j-arr[i-1]],dp[i-1][j]);
                }
        }
        return dp[n][sum];
}
```

2) Equal sum partition

Problem Statement : We have an arr[0..n-1] return true if it is possible to break the array into two halfs such that the sum of values present in the two halfs must be equal, otherwise false.

Solution idea :

```
int sum=0;
for(int i=0;i<n;i++)
        sum+=arr[i];
if(sum&1)
        return false;
else
{
        if(subset_sum(arr,n,sum/2))
                return true;
        else
                return flase;
}
```

3) Count of subsets with a given sum

Problem Statement : We have an arr[0..n-1] and sum find the number of subsets whose sum would be equal to the given sum.

Solution :

```
int solve(int arr[],int n,int sum)
{
    vector<vector<int>> dp(n+1,vector<int>(sum+1,0));
    for(int i=0;i<=sum;i++)
        dp[0][i]=1;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=sum;j++)
        {
            dp[i][j]+=dp[i-1][j];
            if(j>=arr[i-1])
                dp[i][j]+=dp[i-1][j-arr[i-1]];
        }
    }
    return dp[n][sum];
}
```

4) Minimum subset sum difference

Problem Statement : We have an arr[0..n-1] lets break the array into two parts such that the difference between their sums must be minimum, output the minimum value.

Solution Idea :

```
int sum=0;
for(int i=0;i<n;i++)
    sum+=arr[i];
sum=sum/2+1;
while(sum--)
{
    if(subset_sum(arr,n,sum))
        return total_sum-(2*sum);
}
```

5) Number of subsets with given difference

Problem Statement : We have an arr[0..n-1] and diff, output the number of pairs of subsets can be formed whose difference is equal to the given difference.

Solution Idea :

```
int ans=0;
int sum=total_sum+1;
while(sum--)
{
    if(subset_sum(arr,n,sum)&&(total_sum-2*sum)==diff)
        ans++;
}
return ans;
```

6) Target sum

Problem Statement : We have an arr[0..n-1] and target output the number of ways of getting that target where you can add some values and sub the remaining values to get the target.

Solution Idea :

It is same as the count the number of subsets with given diff but here the diff is equal to the target.

```
return count_subset_diff(arr,n,target);
```