DP
Lesson 5

Knapsack with Duplicate Items

Given a set of **N** items, each with a weight and a value, represented by the array **w[]** and **val[]** respectively. Also, a knapsack with weight limit **W**.

The task is to fill the knapsack in such a way that we can get the maximum profit. Return the maximum profit.

Note: Each item can be taken any number of times.

Example 1:

Input: N = 2, W = 3
val[] = {1, 1}
wt[] = {2, 1}
Output: 3
Explanation:
1.Pick the 2nd element thrice.
2.Total profit = 1 + 1 + 1 = 3. Also the total
weight = 1 + 1 + 1 = 3 which is <= W.

Example 2:

Input: N = 4, W = 8
val[] = {1, 4, 5, 7}
wt[] = {1, 3, 4, 5}
Output: 11
Explanation: The optimal choice is to
pick the 2nd and 4th element.

Expected Time Complexity: O(N*W)

Expected Auxiliary Space: O(W)

Constraints:

$1 \leq N, W \leq 1000$

$1 \leq val[i], wt[i] \leq 100$

```
int t[1001][1001];
class Solution{
public:

  int knapsack(int W,int wt[] , int val[] , int N)
  {
     if(W==0|| N==0)
        return 0;
     if(t[N][W]!=-1)
        return t[N][W];
     else if(wt[N-1]<=W)
        return t[N][W]=max(knapsack(W ,wt,val, N-1),
                              val[N-1]+knapsack(W-wt[N-1] ,wt,val, N));
     else
        return t[N][W]=knapsack(W ,wt,val, N-1);
  }
  int knapSack(int N, int W, int val[], int wt[])
  {
     memset(t,-1,sizeof(t));
     return knapsack(W, wt, val, N);
     // code here
  }
};
```

Rod Cutting

Given a rod of length **N** inches and an array of prices, **price[]** that contains prices of all pieces of size smaller than **N**. Determine the maximum value obtainable by cutting up the rod and selling the pieces.

Example 1:

Input:

N = 8

Price[] = {1, 5, 8, 9, 10, 17, 17, 20}

Output:

22

Explanation:

The maximum obtainable value is 22 by cutting in two pieces of lengths 2 and 6, i.e., 5+17=22.

Example 2:

Input:

N=8

Price[] = {3, 5, 8, 9, 10, 17, 17, 20}

Output: 24

Explanation:

The maximum obtainable value is 24 by cutting the rod into 8 pieces of length 1, i.e, 8*3=24.

Expected Time Complexity: O(N2)

Expected Auxiliary Space: O(N)

Constraints:

$1 \leq N \leq 1000$

$1 \leq Ai \leq 105$

```
int t[1001][1001];
 int knapsack(int W,int wt[] , int val[] , int N)
  {
     if(W==0|| N==0)
        return 0;
     if(t[N][W]!=-1)
        return t[N][W];
     else if(wt[N-1]<=W)
        return t[N][W]=max(val[N-1]+knapsack(W-wt[N-1] ,wt,val, N),knapsack(W ,wt,val, N-1));
     else
        return t[N][W]=knapsack(W ,wt,val, N-1);
  }
  int cutRod(int price[], int n)
  {
     int wt[n];
     for(int i=1;i<=n;i++)
     wt[i-1]=i;
     memset(t,-1,sizeof(t));
     return knapsack(n, wt, price, n);
     //code here
  }
};
```

Coin Change

Given a value N, find the number of ways to make change for N cents, if we have infinite supply of each of S = { S1, S2, .. , SM } valued coins.

Example 1:

Input:

n = 4 , m = 3

S[] = {1,2,3}

Output: 4

Explanation: Four Possible ways are:

{1,1,1,1},{1,1,2},{2,2},{1,3}.

Example 2:

Input:

n = 10 , m = 4

S[] ={2,5,3,6}

Output: 5

Explanation: Five Possible ways are:

{2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}.

Expected Time Complexity: O(m*n).

Expected Auxiliary Space: O(n).

Constraints:

1 <= n,m <= 103

```
long long int count(int S[], int m, int n)
  {
     long long int t[m+1][n+1];

     for(int i=0;i<=n;i++)
     {
        t[0][i]=0;
     }
     for(int i=0;i<=m;i++)
     {
        t[i][0]=1;
     }

     for(int i=1;i<=m;i++)
     {
        for(int j=1;j<=n;j++)
        {
           if(S[i-1]<=j)
           {
              t[i][j] = t[i-1][j]+t[i][j-S[i-1]];
           }
           else
              t[i][j]=t[i-1][j];
        }
     }
     return t[m][n];
     // code here.
  }
```

Number of Coins

Given a value **V** and array **coins[]** of size **M**, the task is to make the change for **V** cents, given that you have an infinite supply of each of coins{coins1, coins2, ..., coinsm} valued coins. Find the minimum number of coins to make the change. If not possible to make change then return -1.

Example 1:

Input: V = 30, M = 3, coins[] = {25, 10, 5}

Output: 2

Explanation: Use one 25 cent coin and one 5 cent coin

Example 2:

Input: V = 11, M = 4,coins[] = {9, 6, 5, 1}

Output: 2

Explanation: Use one 6 cent coin and one 5 cent coin

Expected Time Complexity: O(V*M)

Expected Auxiliary Space: O(V)

Constraints:

1 ≤ **V*M** ≤ 106

```
int minCoins(int coins[], int M, int N)
        {
            int t[M+1][N+1];
            for(int i=0;i<=M;i++)
                t[i][0]=0;
            for(int i=1;i<=N;i++)
                t[0][i]=INT_MAX-1;

            for(int j=1;j<=N;j++)
            {
                if(j%coins[0]==0)
                    t[1][j]=j/coins[0];
                else
                    t[1][j]=INT_MAX-1;

            }

            for(int i=2;i<=M;i++)
            {
                for(int j=1;j<=N;j++)
                {
                    if(coins[i-1]<=j)
                        t[i][j]=min(t[i][j-coins[i-1]] + 1 , t[i-1][j]);
                    else
                        t[i][j]=t[i-1][j];
                }
            }
            if (t[M][N] == INT_MAX - 1)
    return -1;

            return t[M][N];
            // Your code goes here
        }
```

Reach a given score

Consider a game where a player can score **3** or **5** or **10** points in a move. Given a total score **n**, find number of distinct combinations to reach the given score.

Example:

Input

3

8

20

13

Output

1

4

2

Explanation

For 1st example when n = 8{ 3, 5 } and {5, 3} are the two possible permutations but these represent the same combination. Hence output is 1.

Constraints:

1 ≤ **T** ≤ 100

1 ≤ **n** ≤ 1000