0-1 Knapsack Problem

We have given ‘n’ items each item has its associated weight and values. Here value means the profit that we will get if we own that item. We have also given a knapsack of capacity W.

The task is to place n items in the knapsack in such a way so that sum of the profits of all the items present in knapsack is maximum and total weight of items should be less than or equal to knapsack capacity.

Here 0-1 means that we can’t put the items in the knapsack in fraction. Either we can place complete item in the knapsack or we have to simply avoid to put that item in knapsack.0-1 knapsack problem is solved by dynamic programming.

Lets n items are (i1,i2,i3,i4…..in), their weight(w1,w2,w3,w4……wn) and profit/value of n items(v1,v2,v3,v4,vn).

Also assume that knapsack capacity=W

So we have to choose subset of n items such that sum of value/profit is maximum and sum of their weights<=W.

Example:n=3

v[]=[6,10,12]

w[]=[10,20,30]

W=50.

Total profit=22 ie,by selecting item 2,and item 3 total profit=22

Recursive formula to solve the above problem.

if (n == 0 || W == 0) i.e. if no item is present for selection or Weight of knapsack is zero then, nothing can be placed in knapsack therefore profit in this case=0.

profit associated with knapsack=0;

if (wt[n - 1] > W)

If weight of the nth item is more than capacity of Knapsack then item cannot be included in the solution therefore return knapSack(W, wt, val, n - 1);

```
    else

        return max(
            val[n - 1] + knapSack(W - wt[n - 1],
                                  wt, val, n - 1),
            knapSack(W, wt, val, n - 1));
}
```

In Dynamic programming we consider the same cases that are mentioned in the recursive approach.

In Dynamic programming we create a table and consider all possible values of weights ie from 1 to W as coloums and all items as rows. The entry table[i][j] denotes maximum value of 'j-weight' considering all values from '1 to ith item'. So if we consider 'wi' (weight in 'ith' row) we can fill it in all columns which have 'weight values > wi'.

Now two possibilities can take place:

Fill 'wi' in the current column.

Do not fill 'wi' in the current column.

we have to choose maximum of these two possibilities, ie. if we do not fill 'ith' weight in 'jth' column then table[i][j] state will be same as table[i-1][j] but if we fill the weight,table[i][j] will be equal to the value of 'wi'+ value of the column weighing 'j-wi' in the previous row. So we will take max of these two possible values to fill the current state.

Let us take an Example:

Total items:3

I[]={i1,i2,i3}

W[] = {1, 2, 3}//Weight array

V[]= {10, 15, 40}//profit array

W=6//Capacity of knapsack

	0	**1**	**2**	**3**	**4**	**5**	**6**
{}	0	0	0	0	0	0	0
{i1}	0	10	10	10	10	10	10
{i1,i2}	0	10	15	25	25	25	25
{i1,i2,i3}	0	10	15	40	50	55	65

Complexity Analysis:

- Time Complexity: O (n*W).
  where ‘n’ is the number of items and ‘W’ is capacity of Knapsack as we have to fill n*W entries to find the maximum profit.
- Space Complexity: O(n*W) as we have used matrix of size n*W