**Experiment No: 6**

**0/1 KNAPSACK PROBLEM**

**Aim:** To implement 0/1 knapsack problem using dynamic programming approach

**Theory:**

Dynamic Programming:

Dynamic Programming is the most powerful design technique for solving optimization problems.

Divide & Conquer algorithm partition the problem into disjoint sub problems solve the sub problems recursively and then combine their solution to solve the original problems.

Dynamic Programming is used when the sub problems are not independent, e.g. when they share the same sub problems. In this case, divide and conquer may do more work than necessary, because it solves the same sub problem multiple times. It solves each sub problems just once and stores the result in a table so that it can be repeatedly retrieved if needed again.

It is a Bottom-up approach- we solve all possible small problems and then combine to obtain solutions for bigger problem.

Dynamic Programming is a paradigm of algorithm design in which an optimization problem is solved by a combination of achieving sub-problem solutions and appearing to the "principle of optimality".

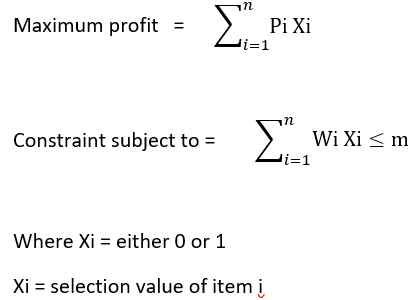
**0/1 Knapsack Problem**

Given ‘n’ objects X1,X2…,Xn with weights W1,W2,…,Wn and profit P1,P2,…,Pn resp.

The knapsack has a capacity ’m’.

If an object ‘i’ with weight Wi is placed in the knapsack, a profit of Pi is earned.

The objective is to fill the knapsack so that the total profit earned is maximized.



Algorithm:

**KNAPSACK (n, cap, p[],w[])**

{

for(i=0;i<=n;i++)

{

for(j=0;j<=cap;j++)

{

if((i==0)||(j==0))

k[i][j]=0;

else if(j<w[i])

k[i][j]=k[i-1][j];

else

k[i][j]=max(k[i-1][j-w[i]]+p[i], k[i-1][j]);

}

Print the profit, k[n][cap]

}

**Algorithm to select the item**

i=n, j=cap

while(i>0 and j>0)

{

if( k[i][j]!=k[i-1][j])

{

Mark/print the ith item

j=j-w[i];

i=i-1;

}

else

i=i-1;

}

**Conclusion**: 0/1 knapsack problem using dynamic programming approach was studied and implemented successfully.

I-1