0/1 KNAPSACK PROBLEM

Problem Statement:

Given a Knapsack of a maximum capacity of W and N items each with its own value and weight, throw in items inside the Knapsack such that the final contents has the maximum value.

Properties:

1. In 0/1 Knapsack problem, items can be entirely accepted or rejected.
2. Given a knapsack with maximum capacity W, and a set S consisting of n items.
3. Each item i has some weight wi and benefit value bi(all wi and W are integer values).
4. The problem is how to pack the knapsack to achieve maximum total value of packed items.
5. For solving the knapsack problem we can generate the sequence of decisions in order to obtain the optimum selection.
6. Let Xn be the optimum sequence and there are two instances {Xn} and {Xn-1, Xn-2… X1}.
7. So from {Xn-1, Xn-2… X1} we will choose the optimum sequence with respect to Xn.
8. The remaining set should fulfill the condition of filling Knapsack of capacity W with maximum profit.
9. Thus, 0/1 Knapsack problem is solved using the principle of optimality.

To solve this problem using dynamic programming method we will perform following steps:

Steps:

- Let, $f_i\ y_j$ be the value of optimal solution.
- Using formula: $f_i(y_j)$ =max $f_{i-1}(y),f_{i-1}(y–wi)+ p_i$ to solve problem.
- Initially S^0 =(0,0)
- Then $S_1^{\ i}$=(P,W)|(P–p_i,W–w_i)Si
- $S_1^{\ i+1}$can be computed by merging S^i and $S_1^{\ i}$
- This is used for obtaining optimal solution.

EXAMPLE:

Distribution Table

i	p_i	w_i
1	1	2
2	2	3
3	5	4
4	6	5

SOLUTION:

Build sequence of decision S^0,S^1,S^2

Initially S^0 = (0,0)

S_0^1= (1,2)

This means while building S_0^1we select the next i^{th}pair. For S_0^1we have selected first (P, W) pair which is (1, 2).

Now S^1 =Merge S^0 and S_0^1

=(0,0),(1,2)

S_1^1={Select next pair (P, W) and add it with S^1}

=(2,3),(2+0,3+0),(2+1,3+2)

=(2,3),(3,5)

since Repetition of (2, 3) is avoided.

S^2 =Merge S^1 and S_1^1

=(0,0),(1,2),(2,3),(3,5)

S_2^1={Select next pair (P, W) and add it with S^2 }

=(5,4),(6,6),(7,7),(8,9)

S^3 ={Merge S^2 and S_2^1}

S^3 =(0,0),(1,2),(2,3),(5,4),(6,6),(7,7),(8,9)

Note that the pair (3, 5) is purged from S^3 . This is because, let us assume (P_j,W_j)=(3,5)and(P_k,W_k)=(5,4), Here $P_j \leq P_k$ and $W_j > W_k$ is true hence we will eliminate pair (Pj,W_j) i.e (3, 5) from S_3

S_3^1={Select next pair (P, W) and add it with S^3 }

=(6,5),(7,7),(8,8),(11,9),(12,11),(13,12),(14,14)

S^4 =(0,0),(1,2),(2,3),(5,4),(6,6),(7,7),(8,9),(6,5),(7,7),(8,8),(11,9),(12,11),(13,12),(14,14)

Now we are interested in M =8. We get pair (8, 8) in S^4 . Hence we will set X_4 . Now we select next object $(P–P_4)$ and $(W–W_4)$ i.e (8 - 6) and (8 - 5). i.e (2, 3) Pair (2, 3) € S^2 hence set X_2 =1. So we get the final solution as (0, 1, 0, 1)

Implementation in CPP:

GeeksForGeeks

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
   if (n == 0 || W == 0)
       return 0;


   if (wt[n-1] > W)
       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));

 }
```

```
int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int  W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```