Experiment No : 6

Name: Shawn Louis

Roll No: 31

Batch: B

Problem Statement:

To implement 0-1 Knapsack

1) Using Dynamic Programming
2) Show items added to the knapsack and the total profit.

Objective:

- To be able to implement a problem using dynamic programming

Expected Outcome:

- **Ability to understand a given problem statement and build logic as per dynamic programming.**
- **Ability to write effficient code.**

Theory:

```
Dynamic-0-1-knapsack (v, w, n, W)
for w = 0 to W do
   c[0, w] = 0
for i = 1 to n do
   c[i, 0] = 0
   for w = 1 to W do
      if $w_i \le w$ then
         if $v_i$ + c[i-1, $w-w_i$] then
            c[i, w] = $v_i$ + c[i-1, $w-w_i$]
         else c[i, w] = c[i-1, w]
      else
         c[i, w] = c[i-1, w]
```

Algorithm:

Let ***i*** be the highest-numbered item in an optimal solution **S** for **W** dollars. Then ***S'* = *S* - *{i}*** is an optimal solution for ***W* - *w*$_i$** dollars and the value to the solution ***S*** is ***V*$_i$** plus the value of the sub-problem.

We can express this fact in the following formula: define **c[i, w]** to be the solution for items **1,2, … , i** and the max$_i$mum weight **w**.

The algorithm takes the following inputs

- The max$_i$mum weight **W**

- The number of items **n**
- The two sequences **v = <v_1, v_2, …, v_n>** and **w = <w_1, w_2, …, w_n>**

The set of items to take can be deduced from the table, starting at **c[n, w]** and tracing backwards where the optimal values came from.

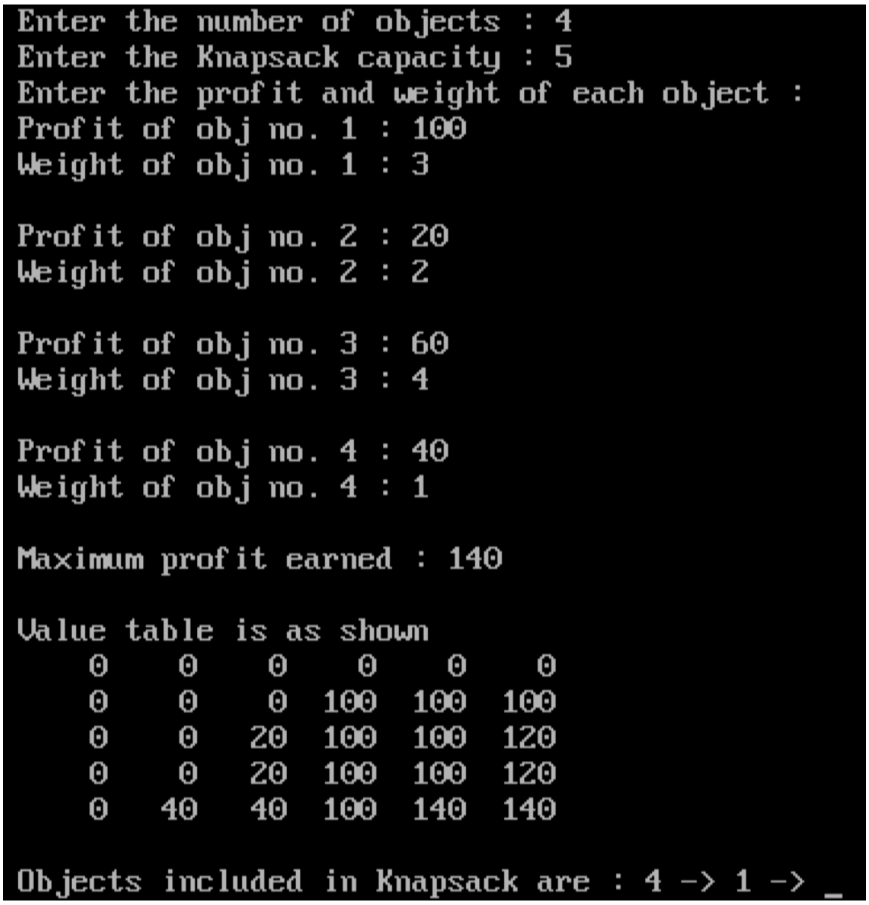
If *c[i, w] = c[i-1, w]*, then item ***i*** is not part of the solution, and we continue tracing with **c[i-1, w]**. Otherwise, item ***i*** is part of the solution, and we continue tracing with **c[i-1, w-W]**.

Program Code:

```
#include<stdio.h>
#include<conio.h>

int max(int x, int y)
{
        if(x > y)
                return x;
        else
                return y;
}

void main()
{
        int i, j, n, m;
        int p[50], w[50], v[50][50];
        clrscr();

        printf("Enter the number of objects : ");
        scanf("%d", &n);

        printf("Enter the Knapsack capacity : ");
        scanf("%d", &m);

        printf("\nEnter the profit and weight of each object : ");
        for(i = 1; i <= n; i++)
        {
                printf("Profit of obj no. %d : ", i);
                scanf("%d", &p[i]);

                printf("Weight of obj no. %d : ", i);
                scanf("%d", &w[i]);

                printf("\n");
        }
```

```
    for (i = 0; i <= n; i++)
    {
        v[i][0] = 0;
    }

    for (j = 0; j <= m; j++)
    {
        v[0][j] = 0;
    }

    for(i = 1; i <= n; i++)
        for(j = 1; j <= m; j++)
            if(i == 1)
                if(j < w[1])
                    v[1][j] = 0;
                else
                    v[1][j] = p[1];
            else if(j < w[i])
                v[i][j] = v[i-1][j];
            else
                v[i][j] = max(v[i-1][j], v[i-1][j-w[i]] +
p[i]);

    printf("\n Maximum profit earned : %d\n", v[n][m]);

    printf("\nValue table is as shown \n");
    for(i = 0; i <= n; i++)
    {
        for(j = 0; j <= m; j++)
            printf("%5d", v[i][j]);
        printf("\n");
    }

    printf("\nObjects included in Knapsack are : ");
    i = n;
    j = m;
    while(i > 0 && j > 0)
    {
        if(v[i][j] != v[i-1][j])
        {
            printf("%d -> ", i);
            j = j - w[i];
            i = i - 1;
        }
        else
            i = i -1;
    }
    getch();
}
```

Output Snapshot:



```
Enter the number of objects : 4
Enter the Knapsack capacity : 5
Enter the profit and weight of each object :
Profit of obj no. 1 : 100
Weight of obj no. 1 : 3

Profit of obj no. 2 : 20
Weight of obj no. 2 : 2

Profit of obj no. 3 : 60
Weight of obj no. 3 : 4

Profit of obj no. 4 : 40
Weight of obj no. 4 : 1

Maximum profit earned : 140

Value table is as shown
    0    0    0    0    0    0
    0    0    0  100  100  100
    0    0   20  100  100  120
    0    0   20  100  100  120
    0   40   40  100  140  140

Objects included in Knapsack are : 4 -> 1 -> _
```

Application

- Finding the least wasteful way to cut raw materials
- Selection of investments and portfolios
- Knapsack Cryptosystems

Outcome:

Successfully analysed and implemented 0/1 Knapsack problem using Dynamic Programming in C.