LAB 13

Implementation of 0-1 Knapsack Algorithm

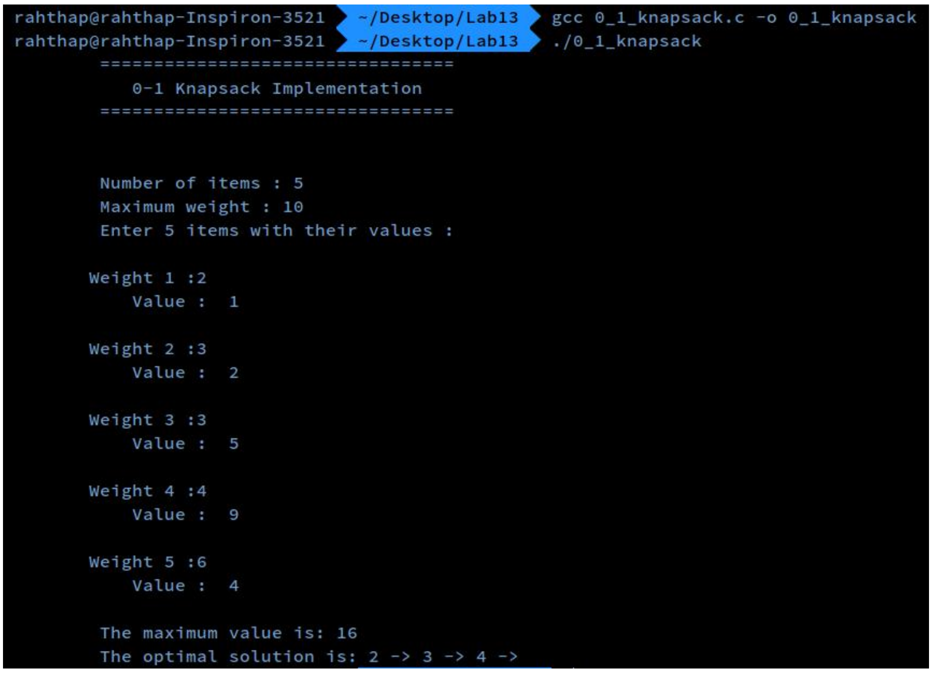
Name : Rahul Thapar

ID : 1410110321

CODE :

```
/*
    @author : Rahul Thapar
    ID          : 1410110321

    Implementation of 0-1 Knapsack
    ------------------------------


*/
#include <stdio.h>
#include <stdlib.h>

int solution[10][10];
int keep[10][10];

int compute_max_val(int weight[10], int value[10],
        int no_items, int max_weight);
void print_optimal(int weight[10], int no_items, int max_weight);

int main(int argc, char *argv[])
{
    int i, no_items, max_weight, max_val;
    int weight[10], value[10];

    printf("\t =================================\n");
    printf("\t      0-1 Knapsack Implementation\n");
    printf("\t =================================\n\n");
    printf("\n\t Number of items : ");
    scanf("%d",&no_items);

    printf("\t Maximum weight : ");
    scanf("%d",&max_weight);

    printf("\t Enter %d items with their values : \n\n", no_items);
    for (i = 1; i <= no_items; i++) {
        printf("\tWeight %d :",i);
        scanf("%d",&weight[i]);
        printf("\t      Value :    ");
        scanf("%d",&value[i]);
        printf("\n");
    }

    max_val = compute_max_val(weight, value, no_items, max_weight);
    printf("\t The maximum value is: %d\n", max_val);
    print_optimal(weight, no_items, max_weight);
}

int compute_max_val(int weight[10], int value[10],
        int no_items, int max_weight)
{
    int i, j, val1, val2;
    for (i = 0; i <= max_weight; i++) {
        solution[0][i] = 0;
        keep[0][i] = 0;
    }

    for (i = 0; i <= no_items; i++) {
        keep[i][0] = 0;
        solution[i][0] = 0;
    }

    /* i - no of items, j - weight */
    for (i = 1; i <= no_items; i++) {
        for (j = 1; j <= max_weight; j++) {
            if (weight[i] > j) {
                solution[i][j] = solution[i-1][j];
                keep[i][j] = 0;
            } else {
                val1 = solution[i-1][j];        // Not choosing the i-th element
                val2 = solution[i-1][j-weight[i]] + value[i];    // Choosing the i-th
                                                          // element
                if (val1 >= val2) {
                    solution[i][j] = val1;
                    keep[i][j] = 0;
                } else {
                    solution[i][j] = val2;
                    keep[i][j] = 1;
                }
            }
            /* printf("solution[%d][%d] = %d\n", i, j, solution[i][j]); */
        }
    }
    return solution[no_items][max_weight];
}

void print_optimal(int weight[], int no_items, int max_weight)
{
    int i = no_items, w = max_weight;     // i - item no, w - weight
    int optimal_soln[10] = {0};

    while (i > 0) {
        if (keep[i][w] == 1) {
            optimal_soln[i] = 1;
            w = w - weight[i];
        }
        i--;
    }

    printf("\t The optimal solution is: ");
    for (i = 0; i <= no_items; i++) {
        if (optimal_soln[i] == 1)
            printf("%d -> ", i);
    }
    printf("\n");
}
```

SCEENSHOTS:



```
rahthap@rahthap-Inspiron-3521  ~/Desktop/Lab13  gcc 0_1_knapsack.c -o 0_1_knapsack
rahthap@rahthap-Inspiron-3521  ~/Desktop/Lab13  ./0_1_knapsack
        =================================
           0-1 Knapsack Implementation
        =================================


        Number of items : 5
        Maximum weight : 10
        Enter 5 items with their values :

       Weight 1 :2
           Value :  1

       Weight 2 :3
           Value :  2

       Weight 3 :3
           Value :  5

       Weight 4 :4
           Value :  9

       Weight 5 :6
           Value :  4

        The maximum value is: 16
        The optimal solution is: 2 -> 3 -> 4 ->
```