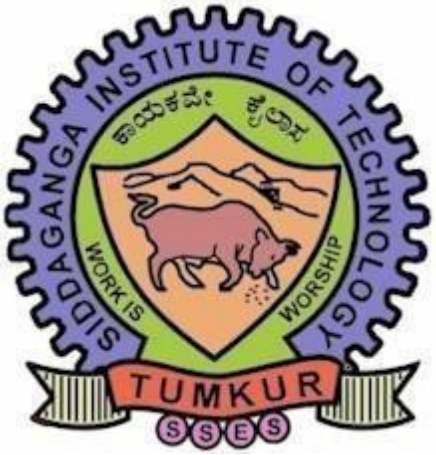
A Report on Open Ended Problem

Submitted for partial fulfillment of v semester analysis and design of algorithms lab

Submitted by
Kaisar Shabir 1SI17CS045



Department of Computer Science and Engineering
Siddaganga Institute of Technology, Tumakuru – 572103
(An Autonomous Institution, Affiliated to VTU, Belagavi & Recognized by AICTE, New Delhi)
2019 -2020

SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMAKURU-3

(An Autonomous Institution, Affiliated to VTU, Belagavi & Recognized by AICTE, New Delhi)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Name of student	USN	Marks
Kaisar Shabir	1SI17CS045	

Signature of In charge Faculty

Department of Computer Science & Engineering

Siddaganga Institute of Technology, Tumakuru-3

(An Autonomous Institute affiliated to VTU,Accredited by NBA)**2019-20**

TABLE OF CONTENTS

INTRODUCTION

Knapsack:

The **knapsack problem** or **rucksack problem** is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorics, computer science, complexity theory, cryptography and applied mathematics.

IMPLEMENTATION

Question No.:1

Alia is planning for a trekking expedition with a backpack that can hold 7kg. She needs to select the most valuable items from the following list that can be accommodated within the backpack. Design and implement an algorithm that displays the most valuable items that can be carried by him using Dynamic Programming Principles.

Items	Weight	Value
1	3	10
2	5	04
3	6	09
4	2	11

Algorithm:

```
//Input:
        // Values (stored in array v)
        // Weights (stored in array w)
        // Knapsack capacity (W)
        //NOTE: The array "v" and array "w" are assumed to store all relevant values starting at
        //index 1.
//Output: optimal solution for the number of items to be selected

for j from 0 to W do:
        m[0, j] := 0
for i from 1 to n do:
        for j from 0 to W do:
                if w[i] > j then:
                        m[i, j] := m[i-1, j]
                else
                        m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])
```

Design technique:

The design technique used is Dynamic Programming. Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of sub problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

Time complexity:

The time efficiency and space efficiency of this algorithm are both in θ(nW).
The time needed to find the composition of an optimal solution is in O(n + W).
Basic Operation: The basic operation is computing the value of V[i, j]. The number of times this is being executed can be calculated as shown below:

$$\sum_0^N \sum_0^M 1 = (M - 0 + 1) \sum_0^N 1$$
$$= (M + 1) \sum_0^N 1$$
$$= (M + 1)(N + 1)$$
$$= MN + N + 1$$
$$\approx MN \text{ (for very large values of M and N)}$$

Hence the time complexity of dynamic knapsack algorithm is given by : **θ**(MN)

Applications:

1) Home Energy Management
2) Cognitive Radio Networks
3) Resource management in software

C code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>


const int MAX = 10;


int max(int a, int b)
{
	return a>b ? a : b;
}


void printWeightValue(int w[MAX], int p[MAX], int num){
	int i;
	printf("Weight\tValue\n");
	for(i = 1; i <= num; i++){
		printf("%d\t%d\n", w[i], p[i]);
	}
}
```

```
void printProfitTable(int capacity, int table[MAX][MAX], int num){
	int i, j;
	for (i=0; i<=num; i++)
	{
		for(j=0; j<=capacity; j++)
		{
			printf("\t%d",table[i][j]);
		}
		printf("\n");
	}
}
void fnPrintProfit(int num, int loaded[MAX], int value[MAX]){
	int totalProfit = 0;
	int i, j;
	for (i=1; i<=num; i++)
	{
		if (loaded[i])
		{
			printf("%d ",i);
			totalProfit += value[i];
		}
	}
	printf("}\n");
	printf("\nTotal Profit : %d\n",totalProfit);
}

void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX][MAX])
{
	int i,j;
	for (j=0; j<=c; j++)
		t[0][j] = 0;

	for (i=0; i<=n; i++)
		t[i][0] = 0;

	for (i=1; i<=n; i++)
	{
		for (j=1; j<=c; j++)
		{
			if (j-w[i] < 0)
				t[i][j] = t[i-1][j];
			else
				t[i][j] = max( t[i-1][j], p[i] + t[i-1][j-w[i]]);
		}
	}
}
```

```
void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int l[MAX])
{
	int i,j;
	i = n;
	j = c;
	while (i >= 1 && j >= 1)
	{
		if (t[i][j] != t[i-1][j])
		{
			l[i] = 1;
			j = j - w[i];
			i--;
		}
		else
			i--;
	}
}

int main(void)
{

	FILE *fp;
	char fName[100];

	int i1;
	int i, j;
	int num = 0;

	int weight[MAX];
	int value[MAX];
	int capacity;
	int loaded[MAX];
	int table[MAX][MAX];
	int choice;
	bool flag = false; //no need for flag here though

	weight[0] = 0;
	value[0] = 0;

	while(1){
		printf("1. Manual Input\n2. Take Input from file\n3. Exit\n");
		printf("Enter your choice:\t");
		scanf("%d", &choice);
		system("clear");
```

```
switch(choice){
        case 1:printf("Enter the number of Items:\t");
                scanf("%d", &num);

                system("clear");

                printf("Enter the weights : \n");
                for (i=1; i<=num; i++)
                {
                        printf("\nWeight %d :",i);
                        scanf("%d",&weight[i]);
                }

                system("clear");

                printf("\nEnter the profits : \n");
                for (i=1; i<=num; i++)
                {
                        printf("\nValue %d :",i);
                        scanf("%d",&value[i]);
                }
                system("clear");
                flag = true;
                break;
        case 2:
                lable1:
                printf("Enter the name of file:\t");
                scanf("%s", fName);

                system("clear");

                fp = fopen(fName, "r");

                if(fp == NULL)
                {
                        printf("%s file does not exist. Try angain\n", fName);
                        goto lable1;
                }

                fclose(fp);
                system("clear");

                fp = fopen(fName, "r");
                char c = getc(fp);
                while(c != EOF){
                        if( c == '\n')
```

```
                                num++;
                        c = getc(fp);
                }
                fclose(fp);

                fp = fopen(fName, "r");

                for(i = 1; i <= num; i++) {
                        fscanf(fp, "%d %d %d", &i1, &weight[i], &value[i]);
                }
                fclose(fp);

                flag = true;
                break;
        case 3:
                exit(0);
        default:
                printf("Invalid input\n");
    }

    if(flag){
        printWeightValue(weight, value, num);

        printf("\nEnter the maximum capacity : ");
        scanf("%d",&capacity);

        for( i=1; i<=num; i++)
                loaded[i] = 0;

        fnProfitTable(weight,value,num,capacity,table);
        fnSelectItems(num,capacity,table,weight,loaded);

printf("============================================================\n");

        printf("Profit Matrix\n");

        printProfitTable(capacity, table, num);

printf("============================================================\n");

        printf("\nItem numbers which can be carried by Alia : \n{ ");

        fnPrintProfit(num, loaded, value);
    }
    flag = false;
    printf("Window will refresh after 10 sec\n");
```

```
            sleep(10);
            system("clear");
        }

        return 0;
}
```

OUTPUT

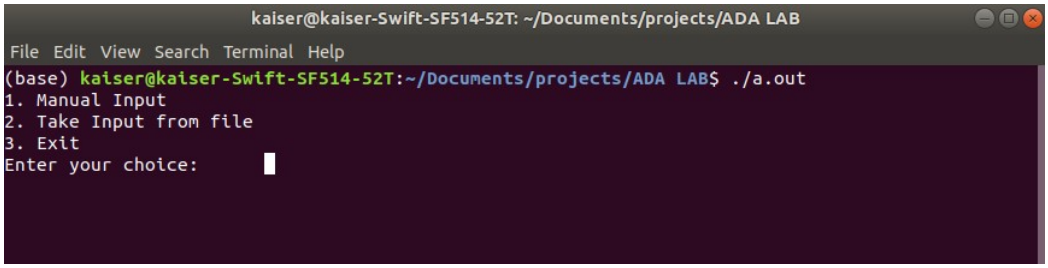


Figure 1: Menu.

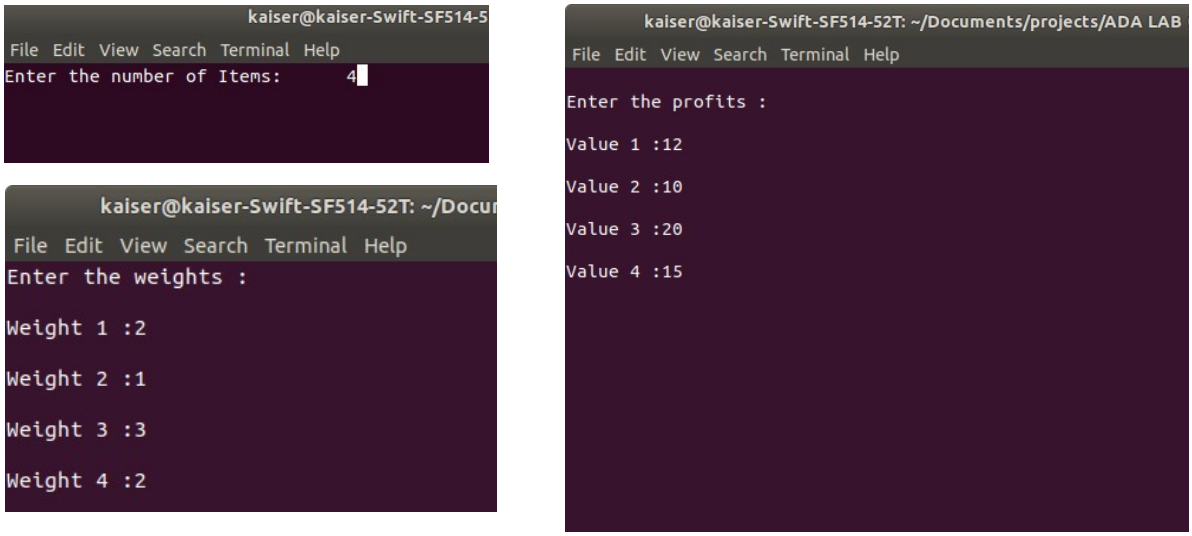


Figure 2, 3 and 4: Option 1 input.

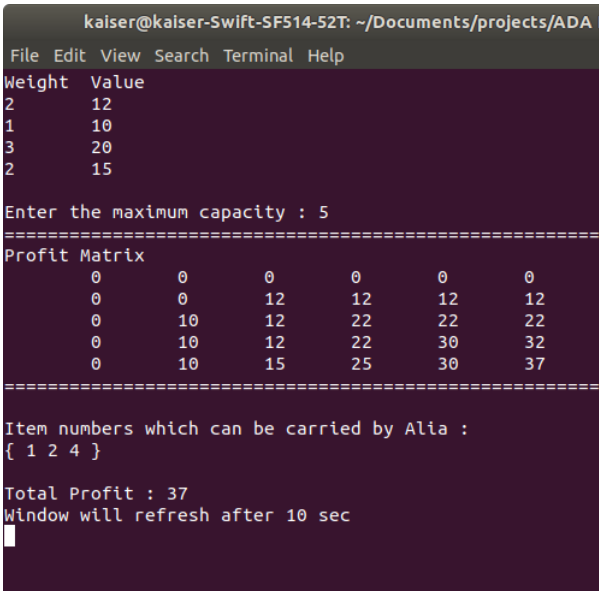


Figure 5: Option 1 output.

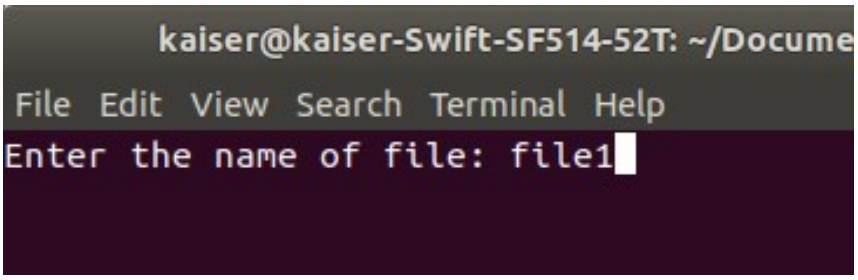


Figure 6: Option 2 input

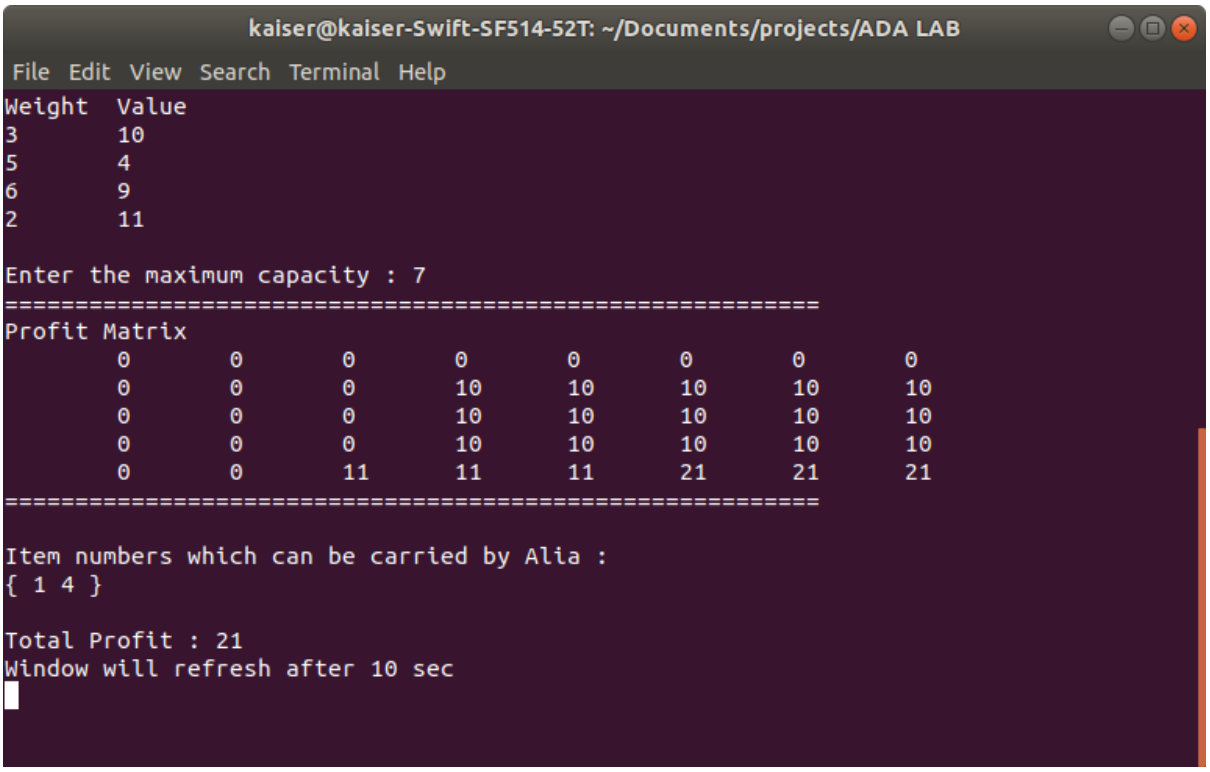


Figure 7: Option 2 output.