Rajshahi University of Engineering & Technology

CSE 2202: Sessional Based on CSE 2201

# Lab Report 08

Date: January 28, 2019

Submitted to


## Biprodip Pal

Instructor, CSE 2201 & CSE 2202

Assistant Professor, Dept. of CSE


Submitted by


## Fuad Al Abir

Roll: 1603021

Section: A

Dept. of CSE

## Sessional – Cycle 8 – Problem A

Given $w_i$ and $p_i$ for N objects. Find the maximum profit using 0/1 Knapsack algorithm using Dynamic Programming approach.

## Code:

```
/*------------------------------
    I N T R O D U C T I O N
------------------------------
Author:     Fuad Al Abir
Date:       January 21, 2019
Name:       knapsackDP.cpp
Objective:  Finding the maximum profit for 0/1 Knapsack problem using Dynamic
Programming approach.
*/

#include <iostream>
#include <time.h>
#include <stdlib.h>

using namespace std;

int max2(int a, int b)
{
    if(a >= b) return a;
    else return b;
}

int main()
{
    time_t random_seed;
    time(&random_seed);
    srand(random_seed);

    int n_object;
    cout << "Number of Objects: ";

    cin >> n_object;
    int weight[n_object + 1];
    int profit[n_object + 1];
    cout << endl;
    cout << "Weight\tProfit" << endl;
    for(int i = 1; i <= n_object; i++)
    {
        weight[i] = 1 + rand() % 5;
        profit[i] = 1 + rand() % 10;
        cout << weight[i] << "\t" << profit[i] << endl;
    }
    int knapsack_weight;
    cout << endl;
    cout << "Knapsack Weight: ";
    cin >> knapsack_weight;

    int knapsackDP[n_object + 1][knapsack_weight + 1];

    for(int i = 0; i <= n_object; i++)
    {
        for(int w = 0; w <= knapsack_weight; w++)
        {
            knapsackDP[i][w] = 0;
        }
    }
```

```
    for(int i = 1; i <= n_object; i++)
    {
        for(int w = 1; w <= knapsack_weight; w++)
        {
            if(weight[i] > w) knapsackDP[i][w] = knapsackDP[i - 1][w];
            else knapsackDP[i][w] = max2( profit[i] + knapsackDP[i-1][w-weight[i]],
knapsackDP[i-1][w]);
        }
    }

    for(int i = 0; i <= n_object; i++)
    {
        for(int w = 0; w <= knapsack_weight; w++)
        {
            cout << knapsackDP[i][w] << "\t";
        }
        cout << endl;
    }

    cout << "\nProfit: " << knapsackDP[n_object][knapsack_weight] << endl;
}
```

## Input/Output:

```
Number of Objects: 7

Weight  Profit
5       7
5       2
3       10
5       4
2       2
3       6
3       5

Knapsack Weight: 13
0  0  0   0   0   0   0   0   0   0   0   0   0   0
0  0  0   0   0   7   7   7   7   7   7   7   7   7
0  0  0   0   0   7   7   7   7   7   9   9   9   9
0  0  0  10  10  10  10  10  17  17  17  17  17  19
0  0  0  10  10  10  10  10  17  17  17  17  17  21
0  0  2  10  10  12  12  12  17  17  19  19  19  21
0  0  2  10  10  12  16  16  18  18  19  23  23  25
0  0  2  10  10  12  16  16  18  21  21  23  23  25

Profit: 25
```

## Sessional – Cycle 8 – Problem B

Given a set of numbers i.e. {2, 5, 7} and an integer N (i.e. 9). Find the elements of given set can make the table of N.

## Code:

```
/*------------------------------
    I N T R O D U C T I O N
------------------------------
Author:     Fuad Al Abir
Date:       January 21, 2019
Name:       SumofN.cpp
Objective:   Finding the elements of a set that can sum up to N using Dynamic
Programming approach.
*/

#include <iostream>
#include <time.h>
#include <stdlib.h>

using namespace std;

int main()
{
    time_t random_seed;
    time(&random_seed);
    srand(random_seed);

    cout << "Number of elements: ";
    int element;
    cin >> element;

    int set[element];
    cout << endl;
    cout << "Set: { ";
    for(int i = 1; i <= element; i++)
    {
        set[i] = 1+rand()%5;
        if(i != element) cout << set[i] << ", ";
        else cout << set[i] << " }";
    }

    cout << endl << endl << "N: ";
    int n;
    cin >> n;

    int T[n + 1][element + 1];
    for(int i = 0; i <= n; i++)
    {
        for(int j = 0; j <= element; j++)
        {
            T[i][j] = 0;
        }
    }

    for(int i = 1;i <= n; i++)
    {
        for(int j = 1; j <= element; j++)
        {
            if(i == set[j])
            {
                T[i][j] = 1;
            }
            else if(i < set[j]){
```

```
                T[i][j] = T[i][j-1];
            }
            else{
                if(T[i - set[j]][j - 1])
                    T[i][j] = 1;
                else
                    T[i][j] = 0;
            }
        }
    }

    cout << endl << "  | ";
    for(int i = 0; i <= element; i++) cout << i << " ";
    cout << endl << "--------------------";

    cout << endl;
    for(int i = 0; i <= n; i++)
    {
        cout << i << " | ";
        for(int j = 0; j <= element; j++)
        {
            cout << T[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

## Input/Output:

```
Number of elements: 4

Set: { 5, 3, 1, 2 }

N: 6

  | 0 1 2 3 4
--------------------
0 | 0 0 0 0 0
1 | 0 0 0 1 1
2 | 0 0 0 0 1
3 | 0 0 1 0 1
4 | 0 0 0 1 0
5 | 0 1 0 0 0
6 | 0 0 0 0 1
```

**Discussion:** As we approached to the problems by Dynamic Programming - we used tabulated approach to solve those. Thus, the result lies at the very last element of the table. For the 0/1 knapsack problem, the last value of the last column is the total profit, whereas, for the sum of N problem, if we get the last value of the last column '1', then the set can make the number N by adding themselves, otherwise not.