



Department of
Computer Science and Engineering

Title: Implement 0-1 Knapsack Problem Using Dynamic Programming (DP)

Algorithms Lab

CSE 206

Submitted by

Shamim Ahmed

ID:201902067



Green University of Bangladesh

Objective(s)

- Understand the basic of dynamic programming
- Apply dynamic programming to solve real-life optimal decision making

Problem Analysis

Suppose a thief is going to steal a store. He has a knapsack to carry goods and maximal weight of W is possible to carry. There are n items available in the store and weight of i -th item is w_i and its profit is p_i . What items should the thief take? Problem is he have to take the item entirely or left it behind which is denoted by $x_i = 0, 1$. Therefore, the items should be selected in such a way that the thief will carry those items for which he will gain maximum profit. Hence, the objective of the thief is to maximize the profit –

$$\max \sum_{i=1}^N x_i p_i \quad (1)$$

In addition, the constraint is

$$\sum_{i=1}^N x_i w_i \leq W \quad (2)$$

Solution Steps

- Take input of list of items, and weights using array
- Construct a DP table $P[n][W]$, where $P[i][w]$ indicates the maximum profit that can be obtained from items 1 to i , if the knapsack has size w
- Case 1: taking the item i , in that case
 $P[i][w] = v_i + P[i-1][w - w_i]$
- Case 2: not taking the item i , in that case
 $P[i][w] = P[i-1][w]$
- The final recurrence relation is
 $P[i][w] = \max\{v_i + P[i-1][w - w_i], P[i-1][w]\}$

We can understand the problem more clearly by the following example

Item	Weight	Value
1	2	12
2	1	10
3	3	20
4	2	15

Figure 1: Weight and profit of each items

$i \backslash w$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	12	12	12	12	12
2	0					
3	0					
4	0					

$P[1][1] = P[0][1] = 0$
 $P[1][2] = \max\{12+0, 0\} = 12$
 $P[1][3] = \max\{12+0, 0\} = 12$
 $P[1][4] = \max\{12+0, 0\} = 12$
 $P[1][5] = \max\{12+0, 0\} = 12$

Figure 2: Iteration 1

$i \backslash w$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	12	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

$P[2][1] = \max\{10+0, 0\} = 10$
 $P[2][2] = \max\{10+0, 12\} = 12$
 $P[2][3] = \max\{10+12, 12\} = 22$
 $P[2][4] = \max\{10+12, 12\} = 22$
 $P[2][5] = \max\{10+12, 12\} = 22$

Figure 3: Iteration 2

Calculate the Iteration 3 by yourself

$i \backslash w$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	12	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

$P[4][1] = P[3][1] = 10$
 $P[4][2] = \max\{15+0, 12\} = 15$
 $P[4][3] = \max\{15+10, 22\} = 25$
 $P[4][4] = \max\{15+12, 30\} = 30$
 $P[4][5] = \max\{15+22, 32\} = 37$

Figure 4: Iteration 4

Time Complexity

Time complexity of 0-1 Knapsack problem is $O(nW)$ where, n is the number of items and W is the capacity of knapsack.

Algorithm

Algorithm 1: Dynamic 0-1 Knapsack

Input: Weights, Values

Output: $P[n, W]$

```

1 for  $w = 0$  to  $W$  do
2    $P[0, w] = 0$ 
3 end
4 for  $i = 1$  to  $n$  do
5    $P[i, 0] = 0$ 
6   for  $w = 1$  to  $W$  do
7     if  $w_i \leq w$  then
8        $P[i, w] = \max\{v_i + P[i-1, w-w_i], P[i-1, w]\}$ 
9     end
10  else
11     $P[i, w] = P[i-1, w]$ 
12  end
13 end
14 end

```

Implementation in C++

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
void knapsack (int v[], int w[],int n, int weight)
```

```
{
```

```
    int k[20][20],i,j;
```

```
    for(i=0; i<=n ;i++)
```

```
    {
```

```
        for(j=0; j<=weight ; j++)
```

```
        {
```

```
            if(i==0 || j==0)
```

```
            {
```

```
                k[i][j]=0;
```

```
            }
```

```
            else if (j<w[i])
```

```
            {
```

```
                k[i][j]=k[i-1][j];
```

```
            }
```

```

else
{
    if (k[i-1][j]>k[i-1][j-w[i]]+v[i])
    {
        k[i][j]=k[i-1][j];
    }
    else
    {
        k[i][j]=k[i-1][j-w[i]]+v[i];
    }
}
}
}

```

```

for (i=0 ;i<=n; i++)
{
    for (j=0; j<=weight; j++)
    {

```

```

        cout<<k[i][j]<<" ";

    }

    cout<<endl;

}

cout<<"\nThe max profit "<<k[n][weight];

}

```

```

int main()

{

    int v[20],w[20],i,j,n,weight;

    //void knapsack(int [], int [], int , int );

    cout<<"Number of object ";

    cin>>n;

    cout<<"\nCapacity of knapsack ";

    cin>>weight;

    for ( i=1; i<=n; i++)

    {

        cout<<"Enter the wiegh and value of "<<i<<" = " ;

```

```
cin>>w[i];
```

```
cin>>v[i];
```

```
cout<<endl;
```

```
}
```

```
knapsack(v,w,n,weight);
```

```
}
```

Sample Input/Output (Compilation, Debugging & Testing)

Output:

Enter No. of Items

4

Enter size of Knapsack

5

Enter the values of items

12 10 20 15

Enter the weights of items

2 1 3 2

Maximum total profit = 37

The screenshot shows a C++ IDE with a file named '0-1 Knapsack.cpp'. The code implements a dynamic programming solution for the 0-1 knapsack problem. It defines a 2D array 'k' to store the maximum profit for subproblems. The main function prompts the user for the number of objects, knapsack capacity, and then for each object's weight and value. The output window shows the execution results: 4 objects, capacity 5, and a maximum profit of 37.

```
33 }
34
35 for (i=0 ;i<=n; i++)
36 {
37     for (j=0; j<=weight; j++)
38     {
39         cout<<k[i][j]<<" ";
40     }
41     cout<<endl;
42 }
43 cout<<"\nThe max profit "<<k[n][weight];
44
45
46
47 int main()
48 {
49     int v[20],w[20],i,j,n,weight;
50     //void knapsack(int [], int [], int , int );
51     cout<<"Number of object ";
52     cin>>n;
53     cout<<"\nCapacity of knapsack ";
54     cin>>weight;
55     for ( i=1; i<=n; i++)
56     {
57         cout<<"Enter the wiegh and value of "<<i<<" = " ;
58         cin>>w[i];
59         cin>>v[i];
60         cout<<endl;
61     }
62
63     knapsack(v,w,n,weight);
64
65
66
67 }
```

Output:

```
Number of object 4
Capacity of knapsack 5
Enter the wiegh and value of 1 = 2 12
Enter the wiegh and value of 2 = 1 10
Enter the wiegh and value of 3 = 3 20
Enter the wiegh and value of 4 = 2 15

0 0 0 0 0
0 0 12 12 12 12
0 10 12 22 22 22
0 10 12 22 30 32
0 10 15 25 30 37

The max profit 37
Process returned 0 (0x0)    execution time : 53.837 s
Press ENTER to continue.
```

Lab Task

Which one has taken in 0 1 knapsack

```
#include <iostream>
```

```
#define MAX 200
```

```
using namespace std;
```

```
int weight[] = {0, 2, 1, 3, 2};
```

```
int value[] = {0, 12, 10, 20, 15};
```

```
int knapsack_weight = 5;
```

```
int n = 4;
```

```
class KnapsackDP
```

```
{
```

```
public:
```



```
int **memoTable;
```

```
KnapsackDP()
```

```
{  
    this->memoTable = new int*[n+1];  
    for(int i=0; i<n+1; i++)  
    {  
        this->memoTable[i]= new int[knapsack_weight+1] {0};  
    }  
}
```

```
int solve()
```

```
{  
  
    for(int i=1; i< (n + 1); i++)  
    {  
        for(int j=1; j<(knapsack_weight + 1); j++)  
        {  
  
            int not_taking_item = memoTable[i-1][j];  
            int taking_item = 0;  
            if(weight[i] <= knapsack_weight)  
            {  
                if(j-weight[i] < 0)  
                    taking_item = memoTable[i-1][j];  
                else  
                    taking_item = value[i] + memoTable[i-1][j-weight[i]];  
            }  
  
            memoTable[i][j] = max(not_taking_item, taking_item);  
        }  
    }  
}
```

```

    }
}

return memoTable[n][knapsack_weight];
}

void selected_items()
{
    for(int i=n, j= knapsack_weight; i>0; i--)
    {
        if(memoTable[i][j] != memoTable[i-1][j])
        {
            cout << "Item: "<<i <<" Selected \n";
            j = j- weight[i];
        }
    }
}
};

int main()
{
    KnapsackDP kdp;
    cout << "Total Benefit: "<< kdp.solve() <<endl;

    //To print which items are included
    kdp.selected_items();
    return 0;
}

```

Output

Terminal

0 1 knapsack which taken.cpp - Code::Blocks 20.03

File Edit View Search Project Build Debug Tools Plugins Settings Help

Start here 0 1 knapsack which taken.cpp 0-1 Knapsack.cpp *Longest-Increasing-Subsequence.cpp *print-sub-sequence.cpp

```
1 #include <iostream>
2 #define MAX 200
3 using namespace std;
4
5
6 int weight[] = {0, 2, 1, 3, 2};
7
8 int value[] = {0, 12, 10, 20, 15};
9
10 int knapsack_weight = 5;
11 int n = 4;
12
13 class KnapsackDP
14 {
15 public:
16     int **memoTable;
17
18     KnapsackDP()
19     {
20         this->memoTable = new int*[n+1];
21         for(int i=0; i<n+1; i++)
22         {
23             this->memoTable[i] = new int[knapsack_weight+1] {0};
24         }
25     }
26
27     int solve()
28     {
29
30         for(int i=1; i< (n + 1); i++)
31         {
32             for(int j=1; j<(knapsack_weight + 1); j++)
33             {
34
35                 int not_taking_item = memoTable[i-1][j];
36                 int taking_item =
```

/home/shamim/Desktop/0 1 knapsack which taken

Total Benefit: 37
Item: 4 Selected
Item: 2 Selected
Item: 1 Selected

Process returned 0 (0x0) execution time : 0.004 s
Press ENTER to continue.

/h6f6@shamim/Desktop/0 1 knapsack which