Danyl Fernandes
2020012004 (72)
20-04-2021

AOA Experiment 5

Aim:

To implement & analyze the 0/1 knapsack problem. Compare its complexity with Fractional Knapsack:

Implementation:

```
#include<stdio.h>
int max(int a, int b) { return (a > b)? a : b; }
int DP_Knapsack(int W, int wt[], int pt[], int n)
{
int i, w, j;
int K[n+1][W+1];
for (i = 0; i <= n; i++)
{
for (w = 0; w <= W; w++)
{
if (i==0 || w==0)
{
K[i][w] = 0;
}
else if (wt[i-1] <= w){
K[i][w] = max(pt[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
}
else
{
K[i][w] = K[i-1][w];
}
}
}
for(i=0;i<=n;i++)
{
for(j=0;j<=W;j++)
{
printf("%d \t",K[i][j]);
}
printf("\n");
}
i=n, j=W;
while(i>0 && j>0)
{
if(K[i][j]==K[i-1][j])
{
```

```c
i--;
}
else
{
//printf("%d=1",i);
i--;
j=j-wt[i];
}
}
return K[n][W];
}
int main()
{
int n;
printf("\nEnter the number of items: ");
scanf("%d", &n);
int pt[n];
int wt[n];
int i;
printf("\nEnter the details of the object: \n");
printf("\n");
for(i = 0; i < n; i++)
{
printf("Enter the weight of the object %d: ",i+1);
scanf("%d", &wt[i]);
printf("Enter the profit of the object %d: ",i+1);
scanf("%d", &pt[i]);
printf("\n");
}
int W;
printf("\nEnter the capacity of the knapsack : ");
scanf("%d", &W);
printf("TABLE:\n");
printf("\n");
printf("\nMaximum profit in a 0-1 knapsack : %d\n",
DP_Knapsack(W, wt, pt, n));
return 0;
}
```

Output:

C:\Users\Lenovo\Documents\knap01.exe

```
*********************************************
             0/1 KNAPSACK PROBLEM
*********************************************

Enter the number of items:  4

Enter the details of the object:

Enter the weight of the object 1: 2
Enter the profit of the object 1: 3

Enter the weight of the object 2: 3
Enter the profit of the object 2: 4

Enter the weight of the object 3: 4
Enter the profit of the object 3: 5

Enter the weight of the object 4: 5
Enter the profit of the object 4: 6


Enter the capacity of the knapsack : 5
TABLE:

0       0       0       0       0       0
0       0       3       3       3       3
0       0       3       4       4       7
0       0       3       4       5       7
0       0       3       4       5       7

Maximum profit in a 0-1 knapsack : 7

--------------------------------
Process exited after 36.18 seconds with return value 0
Press any key to continue . . .
```

Danyl Fernandes
2020012004 (72)

Exp 05 :

Theory :

- The knapsack problem is one of the classic problems in combinational optimization that can be solved by diffrent algorithmic strategies.

- There are two variants of this problems 0/1 knapsack and fractional knapsack problem

- Dynamic programming approach enumerates a sequence of decisions on the inclusion or the exclusion of each item to get an optimal solution.

Approach :

- In the dynamic programming, considering the same cases as mentioned in the recursive approach

- In a DP[ ][ ] table, consider all the possible weights from '1' to 'W' as the coloumn and weights that can kept as the rows.

- The state DP[i][j] will denote maximum value of 'j-weights' considering all the values from '1 to ith', so if we consider 'wi' (weight in ith row) we can fill it all coloumns

Daryl Fernandes
2020012004(72)

which have 'weight values' > 'wi'

Two possiblities can be considered:

- Fill 'wi' in given coloumn
- Do not fill 'wi' the the given coloumn

We take the maximum of these two possibilities, formally if we do not fill 'ith' weight in the 'jth' coloumn the DP[i][j] ~~sate~~ state will be same as DP[i-1][j] but if we fill the weight, DP[i][j] will be equal to the value of 'wi' + value of the coloumn, weighing 'j-wi' in the previous row.

So we take the maximum of these two possibilities to fill the current state

Algorithm:

```
Dynamic_01_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 wi ≤ w then
        if v1 + c[i-1, w-wi] then
            c[i, w] = v1 + c[i-1, w-w1]
        else c[i, w] = c[i-1, w]
    else
        c[i, w] = c[i-1, w]
```

Daryl Fernandes
2020012004 (72)

Complexity Analysis :

- The no of items = n and the capacity of knapsack = M , decide the complexity of the algorithm knapsack-DP (D, W, n, M)

- The time to compute all table entries is O(nM) & the time to trace the decision sequence on x_i's that gives an optimal solution is O(n)

- Thus the time complexity of knapsack_DP becomes O(NM)

Comparison with Greedy Algorithm :

- In the 0-1 knapsack, we are not break item, we can consider the entire item or not take it at all.

- The time to compute entries for this O(nm)
- In fractional, we can break the item for the maximazing the total value of knapsack

- Any efficient sorting algorithm takes O(nlgn) time to sort the items, hence the complexity is O(nlgn).

- Greedy apprach is faster than the dynamic approach but dynamic apprach promise an optimal solutions.

Daryl Fernandes
2020012004 (72)

- As in some cases greedy algorithm fails to provide an globally optimal solutions but dynamic approach always provides a globally optimal solution.

Example :

$(w_1, w_2, w_3, w_4) = (2, 3, 4, 5)$
$(p_1, p_2, p_3, p_4) = (3, 4, 5, 6)$

$w = 5, \quad n = 4$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

$T(i,j) = \max\{T(i-1), j\}$, value $i + T(1-i, j-w)\}$

$T(1,1) = \max\{T(0,1), 3+T(0,-1)\}$
$T(1,1) = 0$

$T(1,2) = \max\{T(0,2), 3+T(0,0)\}$
$= \max(0,3) = 3$

$T(1,3) = \max\{T(0,3), 3+T(0,1)\}$
$= \max\{0, 3\} = 3$

$T(1,4) = \max\{0, 3+0\} = 3$

Danyl Fernandes
2020012004 (72)

Calculating similarly for all values
So max possible value that can be
put into the knapsack = 7

Conclusion : We were successfully able to
implement & analyze the dynamic programming
approach for 0/1 knapsack.