Danyl Fernandes
2020012004 (72)
18-04-2021

AOA Experiment 3

Aim:

To implement & analyze Fractional Knapsack problem:

Implementation:

```
#include<stdio.h>
#include<conio.h>
void fracKnapsack(float cap, int n, float wt[], float pt[]) {
    float x[20], total_pt,unused_cap;
    int i,j;
    unused_cap=cap;
    total_pt=0;
    for(i=0;i < n;i++)
        x[i]=0.0;
    for(i=0;i<n;i++) {
        if(wt[i] > unused_cap)
            break;
        else {
            x[i]=1.0;
            total_pt=total_pt+pt[i];
            unused_cap=unused_cap-wt[i];
        }
    }
    if(i < n)
        x[i]=unused_cap/wt[i];
    total_pt=total_pt+(x[i]*pt[i]);
    printf("The items added in the knapsack are:-\n ");
    for(i=0;i<n;i++)
        if(x[i]==1.0)
            printf("\nProfit for object with weight %.2f = %.2f ",
pt[i], wt[i]);
        else if(x[i] > 0.0)
            printf("\n%.2f part of Profit %.2f with weight %.2f",
x[i], pt[i], wt[i]);
    printf("\nTotal profit for %d objects with capacity %.2f =
%.2f\n\n", n, cap,total_pt);
    printf("_______________________");
}
```

```c
}
int main() {
    float item_wt[20],item_pt[20],pt_per_wt[20], t1,t2,t3;
    int n;
    float capacity;
    int i,j;
    printf(" Enter the available number of objects:  ");
    scanf("%d", &n);
    printf("\nEnter the total capacity of knapsack: ");
    scanf("%f", &capacity);
    for(i=0;i < n;i++) {
        printf("\nEnter the profit for object %d: ", (i+1));
        scanf("%f", &item_pt[i]);
        printf("Enter the weight for object %d: ", (i+1));
        scanf("%f", &item_wt[i]);
        pt_per_wt[i]=item_pt[i]/item_wt[i];
    }
    for(i=0;i < n;i++)
        for(j=0;j < n;j++) {
            if(pt_per_wt[i] > pt_per_wt[j]) {
                t1=pt_per_wt[i];
                pt_per_wt[i]=pt_per_wt[j];
                pt_per_wt[j]=t1;
                t2=item_wt[i];
                item_wt[i]=item_wt[j];
                item_wt[j]=t2;
                t3=item_pt[i];
                item_pt[i]=item_pt[j];
                item_pt[j]=t3;
            }
        }
    fracKnapsack(capacity,n,item_wt,item_pt);
    return 0;
}
```

Output:

```
C:\Users\thearchhero\Desktop\knapsack.exe
 Enter the available number of objects:  4

Enter the total capacity of knapsack: 10

Enter the profit for object 1: 5
Enter the weight for object 1: 1

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

Enter the profit for object 3: 7
Enter the weight for object 3: 5

Enter the profit for object 4: 8
Enter the weight for object 4: 1
The items added in the knapsack are:-

Profit for object with weight 8.00 = 1.00
Profit for object with weight 5.00 = 1.00
Profit for object with weight 7.00 = 5.00
0.75 part of Profit 3.00 with weight 4.00
Total profit for 4 objects with capacity 10.00 = 22.25


_______________________
Process returned 0 (0x0)   execution time : 35.500 s
Press any key to continue.
```

Daryl Fernandes
2020012004 (72)

Exp 3

Theory :

- The fractional knapsack problem follows the subset paradigm of greedy approach.

- It is also known as continuous knapsack problem. This variant of knapsack problem allows keeping the fractional of any item into a knapsack.

- It is not allowed in 0/1 knapsack problem

Procedure :

- To get the maximum profit by filling a knapsack with objects, the greedy approach tries to select the objects with more profit value

- However, those objects should be of lighter weights, so that more objects can be kept into the knapsack considering its capacity

- This is achieved by checking the ratio of profit to weight of each other. The greedy algorithm arranges all the items in descending order of the ratios of their profits to weight

Danyl Fernandes
2020012004 (72)

- The items are kept in a knapsack as per this order. If no sufficient space is available in a knapsack to add a whole item as per this order, then its fractional part is added into the knapsack to make it full.

Algorithm

GreedyKnapsack (m, n)
// p[1:n] and w[1:n] contain the profits and weights respectively.
// of the n objects ordered such that p[i] / w[i] ≥ p[i+1] / w[i+1]
// m is the knapsack size and x[1:n] is the solution vector

```
for i := 1 to n do x[i] := 0.0;
    u := m;
for i := 1 to n do {
    if (w[i] > u) then break;
    x[i] := 1.0;  u := u - w[i];
}
    if (i ≤ n) then x[i] = v/w[i]
}
```

Analysis:

- Any efficient sorting algorithm takes O(n log n) time to sort n items in descending order of their profit to weight rations

Daryl Fernandes
2020012004 (72)

- If the objects are alredy arranged in the required order, the while loop takes $O(n)$
- Therefore, the total time complexity is considered as $O(n \log n)$

Example :

$p = (18, 5, 9, 10, 12, 17)$
$w = (7, 2, 3, 5, 3, 8)$

$m = 18$, $n = 6$

Let $i1, i2, i3, i4, i5, i6$ be the 6 items with profits
$p(1:6) = \{18, 5, 9, 10, 12, 7\}$
$w(1:6) = \{7, 2, 3, 5, 3, 2\}$

1) Calculating the $\frac{Pi}{Wi}$ ratios, $1 \leq i \leq n$

item	P_i	w_i	P_i/W_i
1	18	7	2.57
2	5	2	2.5
3	9	3	3
4	10	5	2
5	12	3	4
6	7	2	3.5

2) Arranging the items in descending order of pi/wi as $i5$, $i6$, $i3$, i, $i2$ & $i4$

3) $m = 18$, we can add items $i5, i6, i3, i1$

Danyl Fernandes
2020012004

& i2 as whole

$\therefore$ Rem capacity $= 18 - (3+2+3+7+2) = 1$

4) Due to insufficient capacity of a knapsack item added as a fractional part $= \frac{1}{5}$ by giving profit.

$$= \frac{1}{5} \times 10 = 2 \text{ units}$$

$\therefore$ Total profit earned

$$= 12 + 7 + 9 + 18 + 5 + 2$$
$$= 53 \text{ units}$$

5) The fixed size solution vector of 6 items $= (1, 1, 1, \frac{1}{5}, 1, 1)$.

Conclusion: Implemented & analysis of the fractional knapsack was successful & desired output was achieved.