

NAME:	Harshal Chawan
UID:	2021300019
SUBJECT	Design and Analysis of Algorithm
EXPERIMENT NO :	05
DATE OF PERFORMANCE	03/04/2023
DATE OF SUBMISSION	11/04/2023
AIM:	To implement fractional knapsack problem and calculate profit.
PROBLEM STATEMENT 1:	Fractional knapsack problem
ALGORITHM Mand THEORY:	Given the weights and profits of N items, in the form of {profit, weight} put these items in a knapsack of capacity W to get the maximum total profit in the knapsack. In Fractional Knapsack, we can break items for maximizing the total value of the knapsack.
Program:	<pre> #include<stdio.h> #include<stdlib.h> struct Item { int SrNo; float w,profit,ratio; }; void sort(int n,struct Item a[n]) { int i,j; struct Item temp; for(i=0;i<n-1;i++) { for(j=0;j<n-1;j++) { </pre>

```

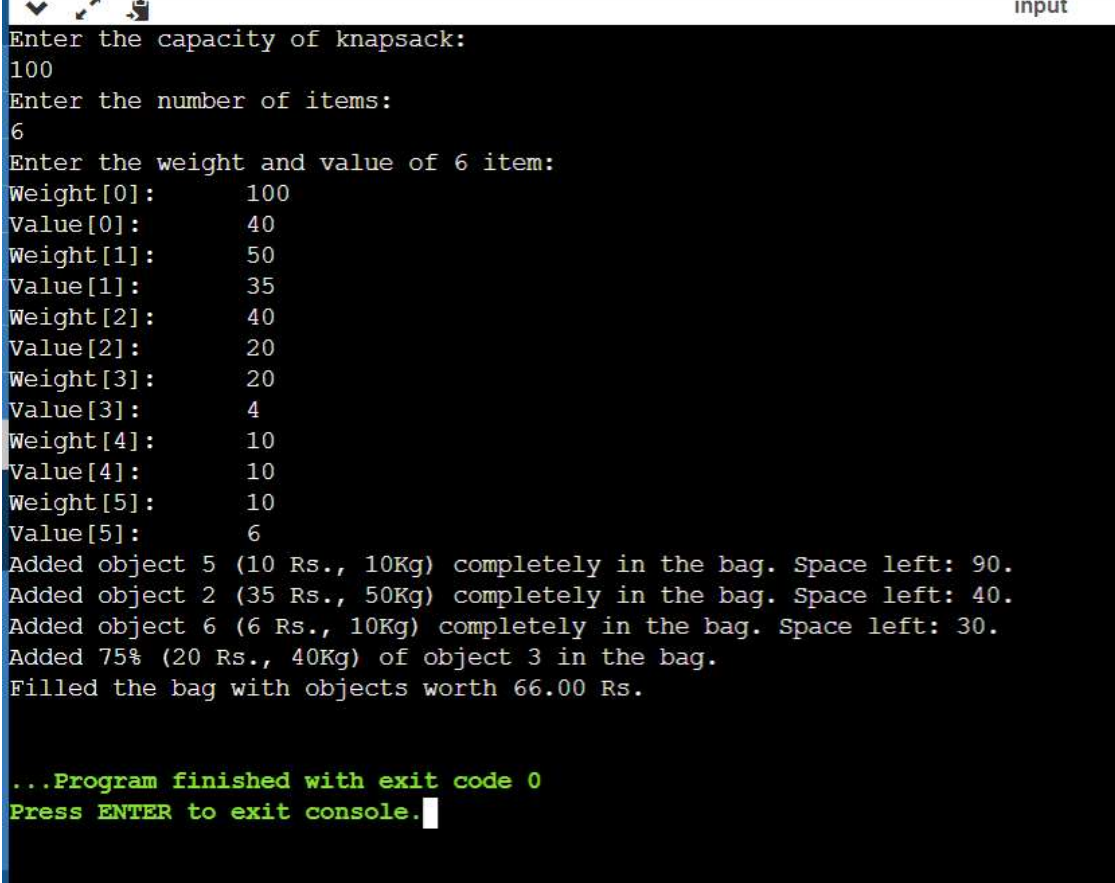
        if(a[j].ratio>a[j+1].ratio)
        {
            temp=a[j];
            a[j]=a[j-1];
            a[j-1]=temp;
        }
    }
}
void main()
{
    int n,i;
    float W,p=0;
    printf("Enter the capacity:");
    scanf("%f",&W);
    printf("Enter the number of elements:");
    scanf("%d",&n);
    struct Item a[n];
    for(i=0;i<n;i++)
    {
        printf("Enter the weight and profit:");
        scanf("%f %f",&a[i].w,&a[i].profit);
        a[i].ratio=a[i].profit/a[i].w;
        a[i].SrNo=i+1;
    }
    printf("\nINITIAL TABLE:\nSr.NO\t\tweight\t\tProfit\t\tP/w");
    for(i=0;i<n;i++)
    {
        printf("\n%d\t\t%f\t\t%f\t\t%f\n",a[i].SrNo,a[i].w,a[i].profit,a[i].ratio);
    }
    sort(n,a);
    printf("\nSORTED TABLE:\nSr.NO\t\tweight\t\tProfit\t\tP/w\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t\t%f\t\t%f\t\t%f\n",a[i].SrNo,a[i].w,a[i].profit,a[i].ratio);
    }

    printf("_____
    _____\n\n");

    printf("Knapsack
    Table:\nSrNo\tElement\t\tweight\t\tProfit\t\tRatio\t\tRemaining
    capacity\t\tTotal Profit\n");

```

	<pre>for(i=0;i<n;i++) { if(W>=a[i].w) { W-=a[i].w; p+=a[i].profit; } else if(W<=a[i].w) { p+=W*a[i].ratio; W=0; } printf("\n%d\t\t%d\t\t%f\t\t%f\t\t%f\t\t%f\t\t%f\n", (i+1), a[i].SrNo, a[i].w, a[i].profit, a[i].ratio, W, p); if(W==0) { break; } } printf("\nTotal Profit: %f", p); }</pre>
OUTPUT:	

	 <pre> input Enter the capacity of knapsack: 100 Enter the number of items: 6 Enter the weight and value of 6 item: Weight[0]: 100 Value[0]: 40 Weight[1]: 50 Value[1]: 35 Weight[2]: 40 Value[2]: 20 Weight[3]: 20 Value[3]: 4 Weight[4]: 10 Value[4]: 10 Weight[5]: 10 Value[5]: 6 Added object 5 (10 Rs., 10Kg) completely in the bag. Space left: 90. Added object 2 (35 Rs., 50Kg) completely in the bag. Space left: 40. Added object 6 (6 Rs., 10Kg) completely in the bag. Space left: 30. Added 75% (20 Rs., 40Kg) of object 3 in the bag. Filled the bag with objects worth 66.00 Rs. ...Program finished with exit code 0 Press ENTER to exit console. </pre>
CONCLUSION:	<p>By performing above experiment I have understood longest common subsequence. This dynamic programming approach reduces time complexity of the calculation of longest common subsequence.</p>

