|  |
| --- |
| **Experiment No. 6** |
| **Fraction Knapsack** |
| Date of Performance:05/04/23 |
| Date of Submission:12/04/23 |

## Experiment No. 6

**Title:** Fractional Knapsack

**Aim:** To study and implement Fractional Knapsack Algorithm

**Objective:** To introduce Greedy based algorithms

#### Theory:

Greedy method or technique is used to solve Optimization problems. A solution that can be maximized or minimized is called Optimal Solution.

The knapsack problem states that − given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that, the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum.

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed size knapsack and must fill it with the most valuable items. The most common problem being solved is the 0-1 knapsack problem, which restricts the number xi of copies of each kind of item to zero or one.

In Knapsack problem we are given:

1. n objects
2. Knapsack with capacity m.
3. An object i is associated with profit Wi.
4. Object i is associated with profit Pi.
5. Object i is placed in knapsack we get profit Pi Xi .

Here objects can be broken into pieces (Xi Values) The Objective of Knapsack problem is to maximize the profit.

**Example:**

Find an optimal solution for fractional Knapsack problem.

Where,

Number of objects = 7

Capacity of Knapsack = 15

P1,P2,P3,P4,P5,P6,P7 = (10,5,15,7,6,18,3)

W1,W2,W3,W4,W5,W6,W7 = (2,3,5,7,1,4,1)

**Solution:**

Arrange the objects in decreasing order of Pi/Wi ratio.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Object** | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **Pi** | 10 | 5 | 15 | 7 | 6 | 18 | 3 |
| **Wi** | 2 | 3 | 5 | 7 | 1 | 4 | 1 |
| **Pi/Wi** | 5 | 1.67 | 3 | 1 | 6 | 4.5 | 3 |

Select the objects with maximum Pi/Wi ratio:

|  |  |  |  |
| --- | --- | --- | --- |
| **Object** | **Profit (Pi)** | **Weight (Wi)** | **Remaining Weight** |
| - | - | - | 15 |
| 5 | 6 | 1 | 14 |
| 1 | 10 | 2 | 12 |
| 6 | 18 | 4 | 8 |
| 3 | 15 | 5 | 3 |
| 7 | 3 | 1 | 2 |
| 2 | 3.33 | 2 | 0 |
| **Total** | **55.33** | | |

**So, the maximum profit is 55.33 units.**

**Algorithm:**

Fractional Knapsack Problem:

Here,

N- Total No. of Objects

M- Capacity of Knapsack

P- Initial profit. P=0

Pi- Profit of ith object

Wi- Weight of ith Object

**Step 1:**

For i=1 to N

**O(n)**

**O(n)**

**O(n.logn)**

Calculate Profit / Weight Ratio (i.e. Pi/Wi)

**Step 2:**

Sort objects in decreasing order of Profit / Weight Ratio

**Step 3: // Add all the profit by considering the weight capacity of fractional knapsack.**

For i=1 to N

if M > 0 AND Wi <= M

M = M –Wi

P = P + Pi

else

break

if M > 0 Then

P = P + Pi \* (M/Wi)

**Step 4:**

Display Total Profit

**Time Complexity** = O(n) + O(n.logn) + O(n)

= Max(O(n),O(n.logn),O(n))

= O(n.logn)

**Code:**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Object {**

**int weight;**

**int profit;**

**float ratio; // Profit/Weight ratio**

**};**

**int compare(const void \*a, const void \*b) {**

**struct Object \*obj1 = (struct Object \*)a;**

**struct Object \*obj2 = (struct Object \*)b;**

**return (obj2->ratio - obj1->ratio);**

**}**

**float knapsack(int N, int M, struct Object items[]) {**

**qsort(items, N, sizeof(items[0]), compare); // Sort items by ratio**

**float totalProfit = 0.0;**

**for (int i = 0; i < N; i++) {**

**if (M > 0 && items[i].weight <= M) {**

**M -= items[i].weight;**

**totalProfit += items[i].profit;**

**} else {**

**break;**

**}**

**}**

**if (M > 0) {**

**totalProfit += items[i].ratio \* M; // Fractional part**

**}**

**return totalProfit;**

**}**

**int main() {**

**int N, M; // N: Total number of objects, M: Capacity of Knapsack**

**printf("Enter the total number of objects: ");**

**scanf("%d", &N);**

**printf("Enter the capacity of knapsack: ");**

**scanf("%d", &M);**

**struct Object items[N];**

**for (int i = 0; i < N; i++) {**

**printf("Enter weight and profit for object %d: ", i + 1);**

**scanf("%d %d", &items[i].weight, &items[i].profit);**

**items[i].ratio = (float)items[i].profit / items[i].weight; // Calculate profit/weight ratio**

**}**

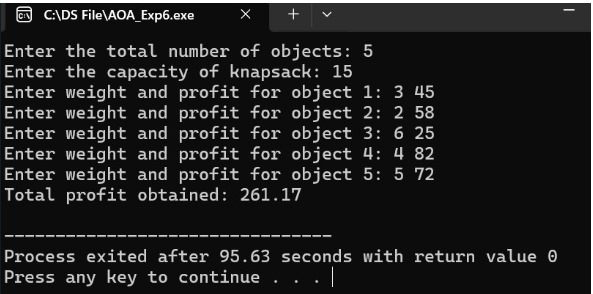
**float totalProfit = knapsack(N, M, items);**

**printf("Total profit obtained: %.2f\n", totalProfit);**

**return 0;**

**}**

**Output:**

****

**Conclusion:**

**The Fractional Knapsack Algorithm, a prime example of a greedy approach, efficiently maximizes profit within knapsack constraints. By prioritizing items based on their profit-to-weight ratios and selecting fractions as needed, it achieves optimal solutions. Unlike the 0-1 knapsack problem, it allows fractional selections, enhancing flexibility and often leading to higher profits. Overall, this algorithm showcases the power of greedy strategies in solving real-world optimization challenges with simplicity and effectiveness.**