**Exp 4**

**Name : Rais Saaim Ravish**

**Roll No : 24DDS03**

**Batch : 2**

**Aim:**Implement Fractional Knapsack problem

**Theory:**

The Fractional Knapsack Problem is an optimization problem where we aim to maximize the total value of items placed in a knapsack with a given weight capacity. Unlike the 0/1 Knapsack Problem, in the fractional version, we can take fractions of items instead of whole items.

Approach:

Greedy Algorithm: The best approach for solving the fractional knapsack problem is the greedy method.

Steps:

* 1. Calculate the value per unit weight for each item.
  2. Sort the items in descending order based on value per unit weight.
  3. Select items with the highest value/weight ratio first, taking full items if possible.
  4. If the full item cannot be taken, take the fraction of the remaining weight.
  5. Continue this process until the knapsack is full.

Time Complexity:

Best Case: O(n)O(n)O(n) (If items are already sorted in decreasing order of value/weight).

Average Case: O(nlog⁡n)O(n \log n)O(nlogn) (Sorting step dominates).

Worst Case: O(nlog⁡n)O(n \log n)O(nlogn) (When items need to be sorted).

Space Complexity: O(n)O(n)O(n) (For storing the items and sorting).

**Program:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 30

int comparisons = 0;

float \*fractionalKnapsack(int w[], int p[], int W, int n)

{

float ratio[n];

int indices[n];

float \*x = (float \*)malloc(n \* sizeof(float));

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

{

ratio[i] = (float)p[i] / w[i];

x[i] = 0;

indices[i] = i;

}

for (int i = 0; i < n - 1; i++)

{

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

{

comparisons++;

if (ratio[indices[i]] < ratio[indices[j]])

{

int temp = indices[i];

indices[i] = indices[j];

indices[j] = temp;

}

}

}

int weight = 0;

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

{

int idx = indices[i];

if (weight + w[idx] <= W)

{

x[idx] = 1;

weight += w[idx];

}

else

{

x[idx] = (float)(W - weight) / w[idx];

break;

}

}

return x;

}

int main()

{

int n, W;

printf("Fractional Knapsack: \nEnter Number of Items: ");

scanf("%d", &n);

printf("Enter Total Weight limit for knapsack: ");

scanf("%d", &W);

int weight[n];

int profit[n];

printf("Enter the weight of %d items: ", n);

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

{

scanf("%d", &weight[i]);

}

printf("Enter the profit of %d items: ", n);

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

{

scanf("%d", &profit[i]);

}

comparisons = 0;

float \*sol = fractionalKnapsack(weight, profit, W, n);

printf("\n Optimal Solution: ");

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

{

printf("%.2f ", sol[i]);

}

float totalProfit = 0;

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

{

totalProfit += sol[i] \* profit[i];

}

printf("\n Total Profit: %.2f \n", totalProfit);

printf("\nNumber of Comparisons: %d \n", comparisons);

free(sol);

return 0;

}

**Output:**

Output 1:

Fractional Knapsack:

Enter Number of Items: 5

Enter Total Weight limit for knapsack: 60

Enter the weight of 5 items: 5 10 15 22 25

Enter the profit of 5 items: 30 40 45 77 90

Optimal Solution: 1.00 1.00 0.00 0.91 1.00

Total Profit: 230.00

Number of Comparisons: 10

Output 2:

Fractional Knapsack:

Enter Number of Items: 7

Enter Total Weight limit for knapsack: 15

Enter the weight of 7 items: 2 3 5 7 1 4 1

Enter the profit of 7 items: 10 5 15 7 6 18 3

Optimal Solution: 1.00 0.67 1.00 0.00 1.00 1.00 1.00

Total Profit: 55.33

Number of Comparisons: 21

**Conclusion:**

The Fractional Knapsack Problem is effectively solved using the greedy algorithm, which prioritizes items with the highest value-to-weight ratio. By sorting the items in descending order based on this ratio, we can maximize the total value placed in the knapsack. The implementation efficiently selects full or fractional parts of items until the capacity is exhausted.