|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2025 - 26** | **Experiment No.** | **5** |
| **Course & Semester** | **S.E. – Sem. III** | **Subject Name** | **Analysis of Algorithm** |
| **Experiment Type** | **Software Performance** | **Subject Code** | **25PCC12CS05** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of Student:** | Atharva Dharmendra Jagtap | **Roll No.:** | 10937 |
| **Date of Performance:** |  | **Date of Submission:** |  |
| **LO Mapping** | 25PCC12CS05.1: Analyze the time and space complexity of algorithms.  25PCC12CS05.3: Apply greedy strategy to solve optimization problem. | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Indicator** | **Poor** | **Average** | **Good** | | Timeline Maintains submission deadline (3) | Submission not done (0) | One or More than One week late (1-2) | Maintains deadline (3) | | Completion and Organization (3) | N/A | Document is just acceptable (1-2) | Completed whole document and neatly organized (3) | | Program Performance (2) | Could not perform at all (0) | Implemented few parts (1) | Full implementation (2) | | Knowledge In depth knowledge of the Experiment (2) | Unable to answer questions (0) | Unable to answer few questions (1) | Able to answer all questions (2) | |
| **Assessment Marks:**   |  |  | | --- | --- | | Timeline |  | | Completion and Organization |  | | Program Performance |  | | Knowledge |  | |
| Total: (Out of 10) |
| Teacher’s Sign: Student Sign: |

**Experiment No. 5**

**AIM:** To Identify and implement an algorithm to be used to solve the challenge faced by airline and shipping companies of maximizing revenue while adhering to weight and space constraints when loading cargo onto airplanes or ships. determine the optimal selection and allocation of cargo items based on their values (revenue) and weights, ensuring efficient use of cargo space.

**THEORY:**

Airlines and shipping companies face the challenge of efficiently utilizing their cargo capacity. The problem involves determining the most valuable combination of items that can be loaded onto a vessel or an aircraft while adhering to constraints such as the total weight or volume of the cargo space. This problem is a variation of the well-known Knapsack Problem, which is a classic optimization problem in computer science.

The Knapsack Problem involves selecting a subset of items, each with a specific weight and value, to include in a knapsack with a fixed capacity. The objective is to maximize the total value (revenue) of the selected items without exceeding the knapsack’s weight (or space) limit. This type of problem is particularly relevant to logistics, transportation, and supply chain management, where companies need to optimize their operations and maximize profitability while managing limited resources.

Knapsack Problem Types:

1. 0/1 Knapsack Problem: Each item can either be included or excluded from the knapsack. The goal is to maximize the total value without exceeding the weight capacity.
2. Fractional Knapsack Problem: Items can be broken into smaller parts, and fractions of the items can be taken. This allows for a more flexible approach to maximizing value.

In the context of airline or shipping cargo, the 0/1 Knapsack Problem is more applicable since cargo items usually cannot be divided into smaller portions. They must be taken whole or left out entirely.

**ALGORITHM:**

 **Input**:

* Read the maximum capacity of the knapsack.
* Read the number of items (n).
* Read the profit array p[] (profits of each item).
* Read the weight array w[] (weights of each item).

 **Compute Profit-to-Weight Ratio**:

* For each item, calculate the ratio profit-to-weight\text{profit-to-weight}profit-to-weight as p[i]w[i]\frac{p[i]}{w[i]}w[i]p[i]​.

 **Sort Items by Ratio**:

* Sort the items in descending order based on their profit-to-weight ratio using a sorting function.

 **Initialize Variables**:

* Set total\_profit to 0 (to store the total profit).
* Set current\_weight to 0 (to track the total weight used).

 **Select Items for Knapsack**:

* Iterate over the sorted list of items:
  + If adding the whole item doesn't exceed the capacity, add the entire item to the knapsack.
  + If adding the item exceeds the capacity, take the fractional part of the item that fits in the remaining capacity.

 **Output**:

* Print the total profit obtained from the selected items.

**CODE (0/1 Knapsack):**

#include <stdio.h>

#include <string.h>

#define MAX 100

struct Knapsack {

    int p;           // profit

    int w;           // weight

    char name[20];   // item name

};

struct Knapsack knapsack[MAX];

int cap = 0, noe = 0;

void sort(char element[][20], int p[], int w[], float ptwr[]) {

    int i, j, temp;

    char name[20];

    float tempRatio;

    for (i = 0; i < noe - 1; i++) {

        for (j = 0; j < noe - i - 1; j++) {

            if (ptwr[j] < ptwr[j + 1]) {

                strcpy(name, element[j]);

                strcpy(element[j], element[j + 1]);

                strcpy(element[j + 1], name);

                temp = p[j];

                p[j] = p[j + 1];

                p[j + 1] = temp;

                temp = w[j];

                w[j] = w[j + 1];

                w[j + 1] = temp;

                tempRatio = ptwr[j];

                ptwr[j] = ptwr[j + 1];

                ptwr[j + 1] = tempRatio;

            }

        }

    }

}

int main() {

    int weight = 0, profit = 0, i;

    printf("Enter knapsack capacity: ");

    scanf("%d", &cap);

    printf("Enter number of elements: ");

    scanf("%d", &noe);

    char elements[noe][20];

    int p[noe];

    int w[noe];

    float ptwr[noe];

    for (i = 0; i < noe; i++) {

        printf("Enter profit, weight and name of element %d: ", i + 1);

        scanf("%d %d %s", &p[i], &w[i], elements[i]);

        ptwr[i] = (float)p[i] / w[i];

    }

    sort(elements, p, w, ptwr);

    printf("\nItems in Knapsack:\n");

    printf("Name\tProfit\tWeight\n");

    for (i = 0; i < noe; i++) {

        if(weight + w[i] <= cap)

        {

        knapsack[i].p = p[i];

        knapsack[i].w = w[i];

        strcpy(knapsack[i].name, elements[i]);

        weight += w[i];

        profit += p[i];

        printf("%s\t%d\t%d\n", elements[i], p[i], w[i]);

        }

    }

    printf("\nTotal Profit: %d\n", profit);

    printf("Total Weight: %d\n", weight);

    return 0;

}

**OUTPUT:**

****

**CODE(Fractional Knapsack**):

#include <stdio.h>

#include <string.h>

#define MAX 100

struct Knapsack {

    float p;         // profit

    float w;         // weight

    char name[20];   // item name

};

struct Knapsack knapsack[MAX];

int cap = 0, noe = 0;

void sort(char element[][20], int p[], int w[], float ptwr[]) {

    int i, j, temp;

    char name[20];

    float tempRatio;

    for (i = 0; i < noe - 1; i++) {

        for (j = 0; j < noe - i - 1; j++) {

            if (ptwr[j] < ptwr[j + 1]) {

                // Swap names

                strcpy(name, element[j]);

                strcpy(element[j], element[j + 1]);

                strcpy(element[j + 1], name);

                // Swap profits

                temp = p[j];

                p[j] = p[j + 1];

                p[j + 1] = temp;

                // Swap weights

                temp = w[j];

                w[j] = w[j + 1];

                w[j + 1] = temp;

                // Swap ratios

                tempRatio = ptwr[j];

                ptwr[j] = ptwr[j + 1];

                ptwr[j + 1] = tempRatio;

            }

        }

    }

}

int main() {

    float weight = 0, profit = 0;

    int i;

    printf("Enter knapsack capacity: ");

    scanf("%d", &cap);

    printf("Enter number of elements: ");

    scanf("%d", &noe);

    char elements[noe][20];

    int p[noe];

    int w[noe];

    float ptwr[noe];

    for (i = 0; i < noe; i++) {

        printf("Enter profit, weight and name of element %d: ", i + 1);

        scanf("%d %d %s", &p[i], &w[i], elements[i]);

        ptwr[i] = (float)p[i] / w[i];

    }

    sort(elements, p, w, ptwr);

    printf("\nItems in Knapsack:\n");

    printf("Name\tProfit\tWeight (Taken)\n");

    for (i = 0; i < noe; i++) {

        if (weight + w[i] <= cap) {

            // Take full item

            knapsack[i].p = p[i];

            knapsack[i].w = w[i];

            strcpy(knapsack[i].name, elements[i]);

            weight += w[i];

            profit += p[i];

            printf("%s\t%d\t%d (full)\n", elements[i], p[i], w[i]);

        } else {

            // Take fractional part

            float remain = cap - weight;

            if (remain > 0) {

                float fraction = remain / w[i];

                knapsack[i].p = p[i] \* fraction;

                knapsack[i].w = remain;

                strcpy(knapsack[i].name, elements[i]);

                weight += remain;

                profit += p[i] \* fraction;

                printf("%s\t%.2f\t%.2f (fractional)\n", elements[i], p[i] \* fraction, remain);

            }

            break; // Knapsack is full

        }

    }

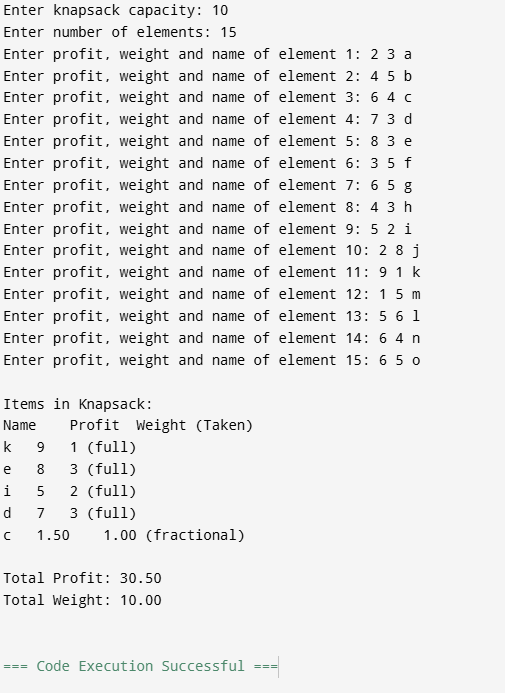
    printf("\nTotal Profit: %.2f\n", profit);

    printf("Total Weight: %.2f\n", weight);

    return 0;

}

**OUTPUT:**

****

**POST LAB QUESTIONS**

**1. Explain how the profit-to-weight ratio influences the selection of cargo items. Why is sorting by this ratio useful in greedy approaches?**

* The profit-to-weight ratio (P/W) indicates the value gained per unit of weight.
* Items with a higher ratio provide more profit for less weight.
* Sorting by this ratio ensures the greedy algorithm always picks the most efficient items first, maximizing profit quickly.

**2. In your implementation, how did you ensure that the total weight of selected items did not exceed the knapsack capacity?**

* The selection loop checks:
* if (weight + w[i] <= cap)
* This condition ensures that adding an item will not exceed the knapsack’s maximum capacity before including it.

**3. If two items have the same profit-to-weight ratio, how should the algorithm decide which one to include? Justify your answer.**

* Choose the item with lower weight first.
* This allows for better space utilization in the knapsack, potentially leaving room to include additional items and increase overall profit.

**CONCLUSION:**

Using the profit-to-weight ratio as the primary selection criterion ensures the greedy knapsack algorithm prioritizes the most valuable items per unit weight, maximizing efficiency. By continuously checking the cumulative weight against the capacity, the implementation guarantees no overflow. In cases of identical ratios, prioritizing lighter items enhances optimal utilization of available space, leading to a higher total profit within the knapsack's constraints.