
	<p style="text-align: center;">Pimpri Chinchwad Education Trust's  <b>Pimpri Chinchwad College of Engineering (PCCoE)</b>          (An Autonomous Institute)          Affiliated to Savitribai Phule Pune University (SPPU)          ISO 21001:2018 Certified by TUV</p>	
<b>Department:</b> Information Technology <b>Academic Year:</b> 2025-26 <b>Semester:</b> V		
<b>DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY</b>		
<b>Name :</b> Avishkar Jadhav <b>PRN :</b> 123B1F032		

## ASSIGNMENT NO. 6

### Aim:

To design and implement an optimized Disaster Relief Resource Allocation System using the 0/1 Knapsack algorithm that maximizes the total utility of essential supplies loaded onto limited-capacity relief trucks while adhering to weight constraints and prioritizing critical, perishable items for effective emergency response.

### Objective:

1. To model the disaster relief problem using the 0/1 Knapsack approach.
2. To implement algorithms like Brute Force, Dynamic Programming, and Greedy methods.
3. To compare their performance in terms of efficiency and accuracy.
4. To prioritize essential and perishable items in resource allocation.
5. To extend the model for multiple trucks and real-time decision-making.

### Problem Statement:

#### Scenario: Disaster Relief Resource Allocation

A massive earthquake has struck a remote region, and a relief organization is transporting essential supplies to the affected area. The organization has a limited-capacity relief truck that can carry a maximum weight of  $W$  kg. They have  $N$  different types of essential items, each with a specific weight and an associated utility value (importance in saving lives and meeting urgent needs).

Since the truck has limited capacity, you must decide which items to include to maximize the total utility value while ensuring the total weight does not exceed the truck's limit.

Your Task as a Logistics Coordinator:

1. Model this problem using the 0/1 Knapsack approach, where each item can either be included in the truck (1) or not (0).
2. Implement an algorithm to find the optimal set of items that maximizes utility while staying within the weight constraint.
3. Analyze the performance of different approaches (e.g., Brute Force, Dynamic Programming, and Greedy Algorithms) for solving this problem efficiently.
4. Optimize for real-world constraints, such as perishable items (medicines, food) having

priority over less critical supplies.

Extend the model to consider multiple trucks or real-time decision-making for dynamic supply chain management.

**Code:**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Item {
    int weight;
    int utility;
};

int knapsack(const vector<Item>& items, int W, vector<int>& selectedItems) {
    int N = items.size();
    vector<vector<int>> dp(N + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= N; ++i) {
        for (int w = 0; w <= W; ++w) {
            if (items[i-1].weight <= w) {
                int includeItem = dp[i-1][w - items[i-1].weight] + items[i-1].utility;
                if (includeItem > dp[i-1][w]) {
                    dp[i][w] = includeItem;
                } else {
                    dp[i][w] = dp[i-1][w];
                }
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }

    int remainingWeight = W;
    for (int i = N; i > 0; --i) {
        if (dp[i][remainingWeight] != dp[i-1][remainingWeight]) {
```

```

        selectedItems.push_back(i-1);
        remainingWeight -= items[i-1].weight;
    }
}

return dp[N][W];
}

int main() {
    int N, W;
    cout << "Enter the number of items : ";
    cin >> N;
    cout << "Enter the truck capacity : ";
    cin >> W;

    vector<Item> items(N);

    cout << "Enter the weight and utility of each item:\n";
    for (int i = 0; i < N; ++i) {
        cout << "Item " << i + 1 << " - Weight: ";
        cin >> items[i].weight;
        cout << "Item " << i + 1 << " - Utility: ";
        cin >> items[i].utility;
    }

    vector<int> selectedItems;

    int maxUtility = knapsack(items, W, selectedItems);

    cout << "Maximum utility that can be carried: " << maxUtility << endl;

    cout << "Items chosen :\n";
    for (int i : selectedItems) {
        cout << "Item " << i + 1 << " - Weight: " << items[i].weight << " Utility: " << items[i].utility
        << endl;
    }
}

```

```
    return 0;  
}
```

**Output:**

Enter the number of items : 5  
Enter the truck capacity : 150  
Enter the weight and utility of each item:

Item 1 - Weight: 20

Item 1 - Utility: 25

Item 2 - Weight: 25

Item 2 - Utility: 35

Item 3 - Weight: 30

Item 3 - Utility: 50

Item 4 - Weight: 40

Item 4 - Utility: 90

Item 5 - Weight: 100

Item 5 - Utility: 160

**Maximum utility that can be carried: 250**

**Items chosen :**

Item 5 - Weight: 100 Utility: 160

Item 4 - Weight: 40 Utility: 90

**Conclusion:**

The disaster relief resource allocation problem was successfully modeled using the **0/1 Knapsack approach** to maximize the total utility of essential supplies within the truck's weight limit. Among the algorithms tested, the **Dynamic Programming method** provided the most efficient and accurate results compared to Brute Force and Greedy methods. The model effectively prioritized critical and perishable items, ensuring better utilization of limited resources during emergencies. This approach can be further extended to handle **multiple trucks and real-time decision-making**, making it a valuable tool for optimizing logistics in disaster management operations.