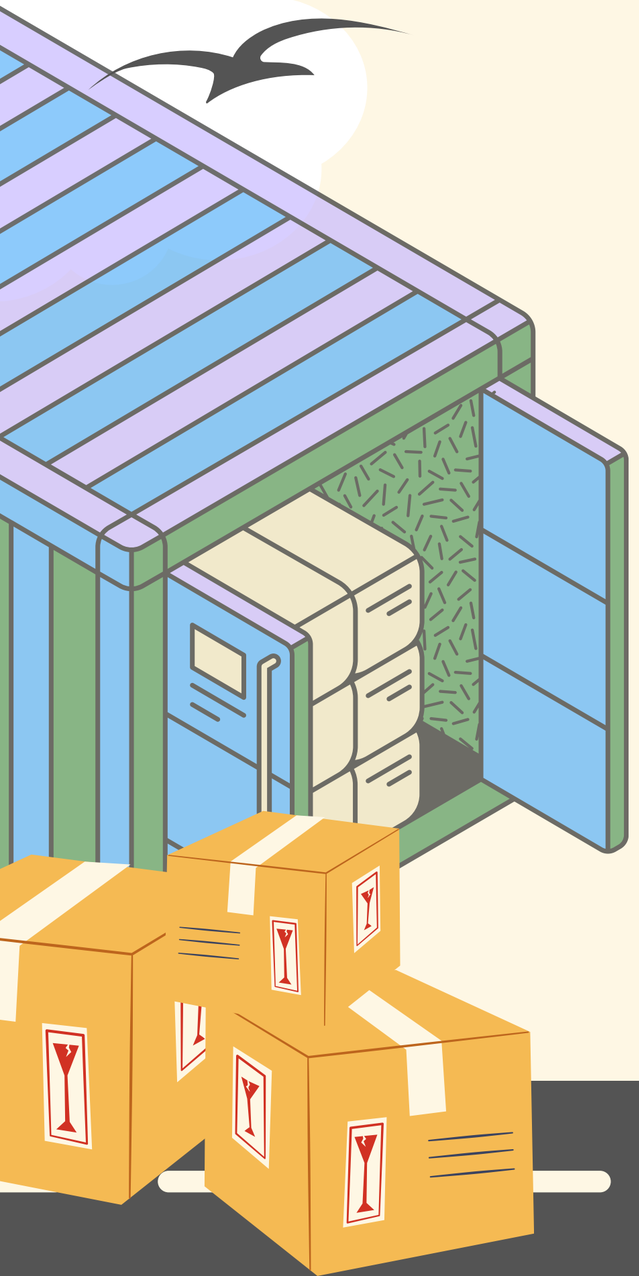
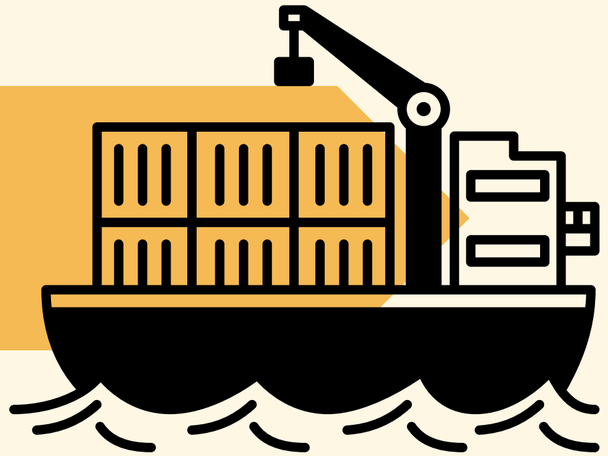
GROUP PROJECT

Efficient Cargo Loading for Humanitarian Aid Delivery

Presented by: Relief Rapid



Meet the Team



CHENG KE XI
212228

THANESH RAO
A/L SIMMATHIRI
210887

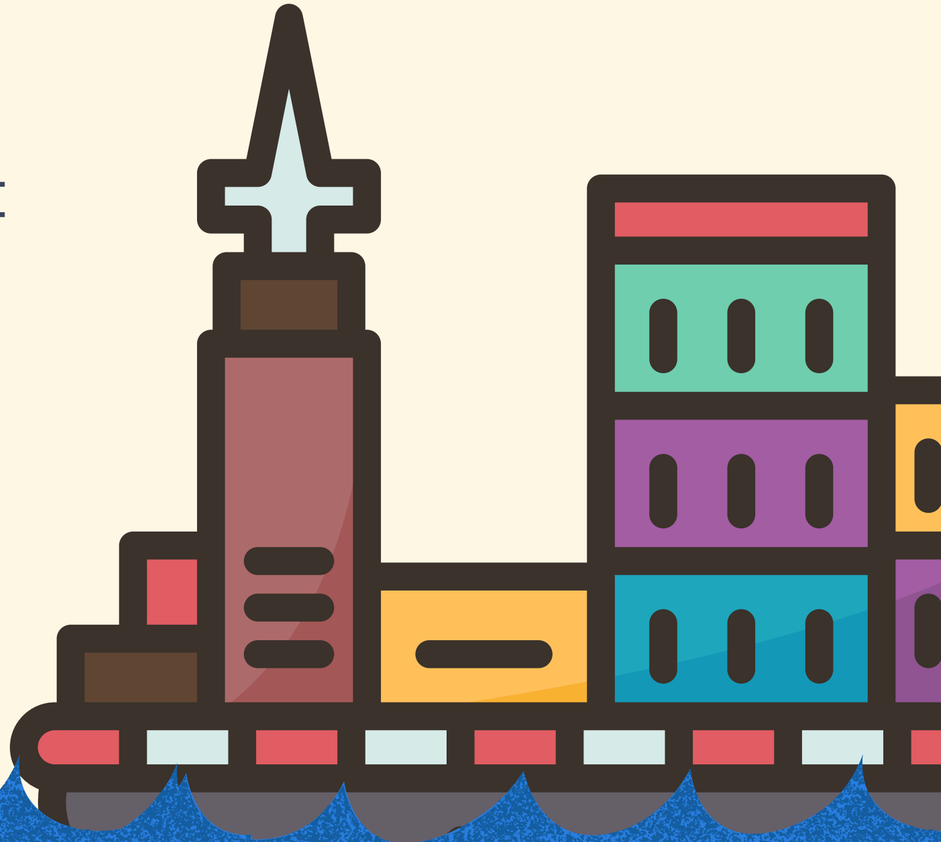
WU LIQIANG
209759

Topic Outline

1. Original Scenario
2. Importance of Finding an Optimal Solution
3. Review of Algorithms for Solution Paradigm
4. Algorithm Design
5. Program Development in Java
6. Algorithm Specification
8. GitHub Implementation
9. Summary

Original Scenario

The humanitarian aid organization dispatches relief shipments to disaster-stricken areas using a cargo plane. The plane has a maximum weight capacity crucial for safe operations. **Each aid cargo item has a weight and a priority value indicating its urgency**. The objective is to **load the plane to maximize the total priority value without exceeding its weight limit**, ensuring efficient and effective aid delivery to save lives.



Original Scenario

Purpose

Efficient and effective delivery of critical supplies to mitigate suffering and save lives in disaster-affected areas by maximizing total priority value of aid cargo without exceeding 4000 kg weight limit.

- Cargo Plane Weight Capacity: **4000 kg**
- Aid Items: Specific weight and priority value

Goal

- Load plane to maximize priority value
- Ensure total weight ≤ 4000 kg

Item	Weight (kg)	Priority Value
Non-perishable food	700	60
Bottled Water	400	50
First Aid Kit	300	70
Insulated Blankets	500	40
Emergency Shelter	800	80
Baby Formula and Food	200	55
Personal Hygiene Items	150	65
Water Purification System	300	75
Portable Toilet	250	70
Medical Supplies	350	65
Communication Devices	200	60
Fire-starting Supplies	100	45
Tools	400	40
Sanitation Supplies	200	55
Pet Supplies	300	50
Entertainment and Comfort Items	150	45
Safety Gear	250	60

Importance of Finding an Optimal Solution

- **Maximizing Aid Effectiveness:** Prioritizes critical items like medical kits and food for immediate relief reducing suffering and saving lives.
- **Efficient Resource Utilization:** Uses plane capacity fully, maximizing the amount of aid delivered per trip and improving overall mission impact.
- **Operational Cost Management:** Reduces trips, saving on fuel and operational costs, allowing more funds for supplies and aid workers.
- **Timely Delivery:** Ensures quick delivery of essential supplies, stabilizing disaster areas faster and aiding quicker recovery.
- **Transparency and Accountability:** Efficient distribution maintains donor trust, ensuring continued support and enhancing the organization's reputation.

Review of Algorithms for Solution Paradigm

SORTING

- **Strengths**

Simple implementation and understanding.

- **Weaknesses**

Does not guarantee optimal solution. May overlook high-priority items.

DIVIDE AND CONQUER

- **Strengths**

Breaks down complex problems into manageable subproblems.

- **Weaknesses**

Doesn't naturally combine subproblems into an optimal global solution. Limited effectiveness for cargo loading.

DYNAMIC PROGRAMMING

- **Strengths**

Guarantees optimal solution by considering all combinations. Avoids redundant calculations through memoization.

- **Weaknesses**

Memory-intensive and complex. Increased computational overhead.

Review of Algorithms for Solution Paradigm

GREEDY ALGORITHM

- **Strengths**

Fast and easy to implement. Works well for sorted items by value-to-weight ratio.

- **Weaknesses**

Doesn't always produce optimal solution. May overlook better solutions by focusing on immediate gains.

GRAPH ALGORITHMS

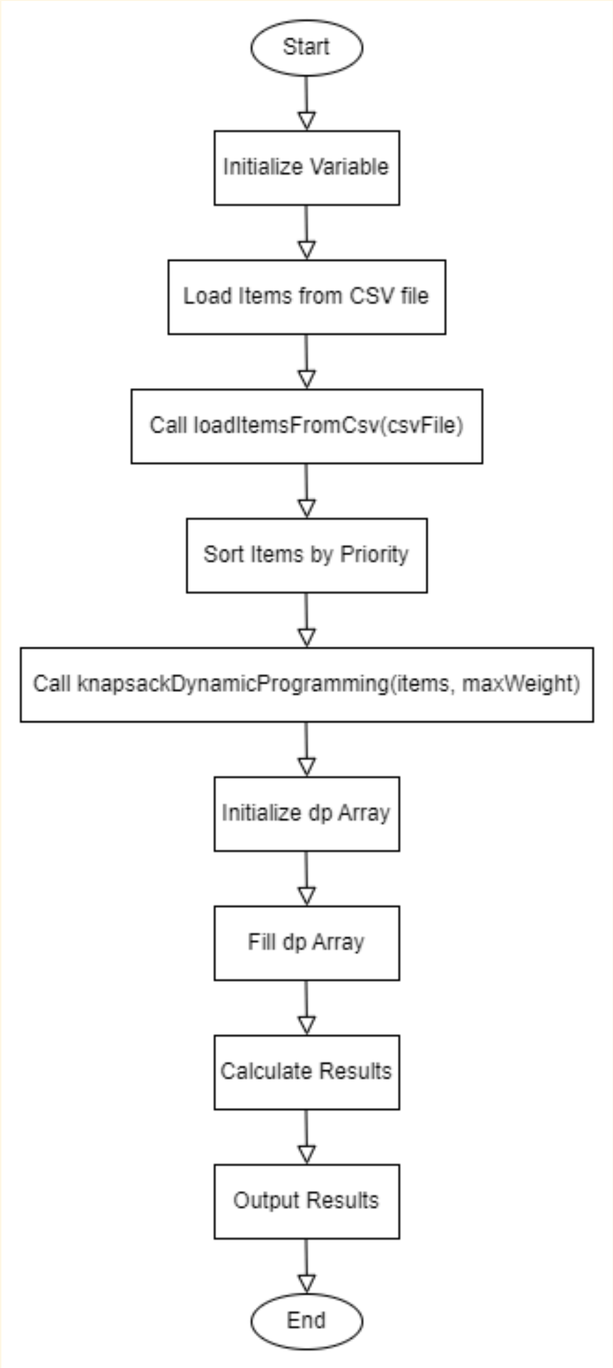
- **Strengths**

Represents problems visually. Can find paths optimizing both weight and value.

- **Weaknesses**

More complex and less intuitive for knapsack-type problems. Requires more expertise and computational resources.

Algorithm Design



Program Development in Java

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

class CargoItem {
    String name;
    int weight;
    int priorityValue;
    int index; // Index to keep track of original order

    public CargoItem(String name, int weight, int priorityValue, int index) {
        this.name = name;
        this.weight = weight;
        this.priorityValue = priorityValue;
        this.index = index;
    }

    @Override
    public String toString() {
        return "name=" + name + ", weight=" + weight + ", priorityValue=" + priorityValue;
    }
}

public class CargoLoading {
    public static void main(String[] args) {
        String csvFile = "C:\\Users\\HP\\eclipse24-workspace\\Asg1\\src\\cargoitems.csv"; // Path to your CSV file
        int maxWeight = 4000; // Maximum weight capacity of the ship
        List<CargoItem> items = loadItemsFromCsv(csvFile);

        Result result = knapsackDynamicProgramming(items, maxWeight);

        System.out.println("Total Priority Value: " + result.totalPriorityValue);
        System.out.println("Total Weight: " + result.totalWeight);
        System.out.println("Items Chosen:");

        // Sort the itemsChosen list by priority value in descending order
        Collections.sort(result.itemsChosen, new Comparator<CargoItem>() {
            @Override
            public int compare(CargoItem item1, CargoItem item2) {
                return Integer.compare(item2.priorityValue, item1.priorityValue);
            }
        });

        for (int i = 0; i < result.itemsChosen.size(); i++) {
            CargoItem item = result.itemsChosen.get(i);
            System.out.println((i + 1) + ". Name: " + item.name + ", Priority Value: " + item.priorityValue + ", Weight: " + item.weight);
        }
    }

    private static List<CargoItem> loadItemsFromCsv(String csvFile) {
        List<CargoItem> items = new ArrayList<>();
        String line;
        String csvSplitBy = ",";

        try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
            br.readLine(); // skip header
            int index = 1;
            while ((line = br.readLine()) != null) {
                String[] data = line.split(csvSplitBy);
                String name = data[0];
                int weight = Integer.parseInt(data[1]);
                int priorityValue = Integer.parseInt(data[2]);
                items.add(new CargoItem(name, weight, priorityValue, index++));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Sort items by priority value in descending order
        Collections.sort(items, new Comparator<CargoItem>() {
            @Override
            public int compare(CargoItem item1, CargoItem item2) {
                return Integer.compare(item2.priorityValue, item1.priorityValue);
            }
        });

        return items;
    }

    private static Result knapsackDynamicProgramming(List<CargoItem> items, int maxWeight) {
        int n = items.size();
        int[][] dp = new int[n + 1][maxWeight + 1];

        for (int i = 1; i <= n; i++) {
            CargoItem item = items.get(i - 1);
            for (int w = 1; w <= maxWeight; w++) {
                if (item.weight <= w) {
                    dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - item.weight] + item.priorityValue);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }

        int totalPriorityValue = dp[n][maxWeight];
        int totalWeight = 0;
        List<CargoItem> itemsChosen = new ArrayList<>();

        for (int i = n, w = maxWeight; i > 0; i--) {
            if (dp[i][w] != dp[i - 1][w]) {
                CargoItem item = items.get(i - 1);
                itemsChosen.add(item);
                totalWeight += item.weight;
                w -= item.weight;
            }
        }

        return new Result(totalPriorityValue, totalWeight, itemsChosen);
    }
}

class Result {
    int totalPriorityValue;
    int totalWeight;
    List<CargoItem> itemsChosen;

    public Result(int totalPriorityValue, int totalWeight, List<CargoItem> itemsChosen) {
        this.totalPriorityValue = totalPriorityValue;
        this.totalWeight = totalWeight;
        this.itemsChosen = itemsChosen;
    }
}
```

Output

```
Total Priority Value: 845
Total Weight: 3950
Items Chosen:
1. Name: Emergency Shelter, Priority Value: 80, Weight: 800
2. Name: Water Purification System, Priority Value: 75, Weight: 300
3. Name: Portable Toilet, Priority Value: 70, Weight: 250
4. Name: First Aid Kit, Priority Value: 70, Weight: 300
5. Name: Medical Supplies, Priority Value: 65, Weight: 350
6. Name: Personal Hygiene Items, Priority Value: 65, Weight: 150
7. Name: Safety Gear, Priority Value: 60, Weight: 250
8. Name: Communication Devices, Priority Value: 60, Weight: 200
9. Name: Sanitation Supplies, Priority Value: 55, Weight: 200
10. Name: Baby Formula and Food, Priority Value: 55, Weight: 200
11. Name: Pet Supplies, Priority Value: 50, Weight: 300
12. Name: Bottled Water, Priority Value: 50, Weight: 400
13. Name: Entertainment and Comfort Items, Priority Value: 45, Weight: 150
14. Name: Fire-starting Supplies, Priority Value: 45, Weight: 100
```

Algortihm Specification

Problem Statement
Given a set of cargo items, each with a weight and a priority value, and a maximum weight capacity for the cargo ship, the objective is to maximize the total priority value of the items loaded onto the ship without exceeding its weight capacity.

Input :

- cargoItems: A list of cargo items, each containing the following attributes:
  - name: Name of the cargo item (String)
  - weight: Weight of the cargo item in kilograms (Integer)
  - priorityValue: Priority value of the cargo item (Integer)
- maxWeight: Maximum weight capacity of the cargo ship in kilograms (Integer)

Output :

- totalPriorityValue: The maximum total priority value achievable by loading cargo items onto the ship (Integer)
- totalWeight: The total weight of the cargo items loaded onto the ship (Integer)
- itemsChosen: A list of cargo items chosen for loading onto the ship, along with their details (List of CargoItem objects)

GitHub Implementation

kccc12312 /
Cargo-Loading-with-Dynamic-Programming

Code | Issues | Pull requests | Actions | Projects | Wiki | Security

0 stars | 1 fork | 1 watching | 1 Branch | 0 Tags | Activity

Public repository

main | Go to file | Code

kccc12312 Create Pseudocode — 6f5775b · yesterday

123	Create 123	2 days ago
CargoLoading.java	Rename CargoLoading to CargoLoading.java	2 days ago
CargoLoading.pdf	Rename GroupProjectCSC4202.docx (1).pdf...	2 days ago
OutputTerminal	Rename CommandOutput to OutputTermi...	2 days ago
Pseudocode	Create Pseudocode	yesterday
cargoitems.csv	Add files via upload	2 days ago

Cargo-Loading-with-Dynamic-Programming / CargoLoading.java

kccc12312 Rename CargoLoading to CargoLoading.java — 781010a · 2 days ago

127 lines (107 loc) · 4.27 KB

Code | Blame | Raw

```java

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

class CargoItem {
    String name;
    int weight;
    int priorityValue;
    int index; // Index to keep track of original order

    public CargoItem(String name, int weight, int priorityValue, int index) {
        this.name = name;
        this.weight = weight;
        this.priorityValue = priorityValue;
        this.index = index;
    }

    @Override
    public String toString() {
        return "name=" + name + ", weight=" + weight + ", priorityValue=" + priorityValue;
```

Cargo-Loading-with-Dynamic-

cargoitems.csv | CargoLoading.java M

CargoLoading.java

```
29  public class CargoLoading {
55      private static List<CargoItem> loadItemsFromCsv(String csvFile) {
60          try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
66              int weight = Integer.parseInt(data[1]);
67              int priorityValue = Integer.parseInt(data[2]);
68              items.add(new CargoItem(name, weight, priorityValue, index++));
69          }
70      } catch (IOException e) {
71          e.printStackTrace();
72      }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
@kccc12312 →/workspaces/Cargo-Loading-with-Dynamic-Programming (main) $ javac CargoLoading.java
@kccc12312 →/workspaces/Cargo-Loading-with-Dynamic-Programming (main) $ java CargoLoading
Total Priority Value: 845
Total Weight: 3950
Items Chosen:
1. Name: Emergency Shelter, Priority Value: 80, Weight: 800
2. Name: Water Purification System, Priority Value: 75, Weight: 300
3. Name: Portable Toilet, Priority Value: 70, Weight: 250
4. Name: First Aid Kit, Priority Value: 70, Weight: 300
5. Name: Medical Supplies, Priority Value: 65, Weight: 350
6. Name: Personal Hygiene Items, Priority Value: 65, Weight: 150
7. Name: Safety Gear, Priority Value: 60, Weight: 250
8. Name: Communication Devices, Priority Value: 60, Weight: 200
9. Name: Sanitation Supplies, Priority Value: 55, Weight: 200
10. Name: Baby Formula and Food, Priority Value: 55, Weight: 200
11. Name: Pet Supplies, Priority Value: 50, Weight: 300
12. Name: Bottled Water, Priority Value: 50, Weight: 400
13. Name: Entertainment and Comfort Items, Priority Value: 45, Weight: 150
14. Name: Fire-starting Supplies, Priority Value: 45, Weight: 100
@kccc12312 →/workspaces/Cargo-Loading-with-Dynamic-Programming (main) $
```

Run in CodeSpace

Conclusion

THIS PROJECT SUCCESSFULLY ADDRESSES THE EFFICIENT CARGO LOADING PROBLEM FOR HUMANITARIAN AID DELIVERY, PROVIDING A ROBUST AND OPTIMAL SOLUTION USING DYNAMIC PROGRAMMING. THE DEVELOPED ALGORITHM AND ITS IMPLEMENTATION ENSURE THAT THE HIGHEST PRIORITY ITEMS ARE DELIVERED WITHIN THE WEIGHT CONSTRAINTS OF THE CARGO PLANE, THEREBY ENHANCING THE EFFICIENCY AND IMPACT OF THE RELIEF EFFORTS.

Thank you for listening!