Data Structures Assignment: Emergency Supply Network Design

IFT2015

December 23, 2024

Introduction

In this assignment, you will design and implement a data structure-based solution to manage an emergency supply network. The goal is to efficiently track and allocate resources to cities from a network of warehouses while managing dynamic updates to supply and demand levels. This exercise focuses on implementing advanced data structures like graphs, heaps, and disjoint sets in a practical problem-solving context.

Problem Description

An international disaster relief organization is establishing an emergency supply network consisting of:

- **Cities:** Each city has a name, unique ID, coordinates as a 2D vector (x,y), demand levels, and priority.
- **Warehouses:** Each warehouse has a unique ID, coordinates as a 2D vector (x,y), and total capacity.

Resources must be allocated dynamically based on city demands and warehouse supplies, while minimizing transportation costs.

Example Scenario for Input

The following example demonstrates the network setup and illustrates the tasks in the assignment.

Network Setup

- **Cities:**
  - City A: ID = 1, Coordinates = (2, 3), Demand = 50 units, Priority = High
  - City B: ID = 2, Coordinates = (5, 7), Demand = 30 units, Priority = Medium
  - City C: ID = 3, Coordinates = (8, 2), Demand = 50 units, Priority = Low

- **Warehouses:**
  - Warehouse X: ID = 101, Coordinates = (10, 20), Capacity = 100 units
  - Warehouse Y: ID = 102, Coordinates = (15, 25), Capacity = 50 units
  - Warehouse Z: ID = 103, Coordinates = (20, 35), Capacity = 110 units

Transport Mode Selection and Cost Calculation

- **Transport Mode Selection:**
  - If the distance $d \leq 10$: Use Drone (Coefficient = 1).
  - If $10 < d \leq 20$: Use Truck (Coefficient = 2).
  - If $d > 20$: Use Rail (Coefficient = 3).
- **Cost Calculation:** The cost of transportation is calculated as:

$$\text{Cost} = \text{Distance}(x_1, y_1, x_2, y_2) \times \text{Coefficient of Transport Mode}$$

  where the distance is the Euclidean distance between two nodes:

$$\text{Distance}(x_1, y_1, x_2, y_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

  The Euclidean distance measures the straight-line distance between two points in a Cartesian plane.

Assignment Tasks and Expected Outputs

Task 1: Construction of an undirected graph

- Represent the network as a graph, with edges representing the transportation cost between cities and warehouses.
- Output the graph as a cost matrix.

Task 2: Priority-Based Resource Allocation

- Use a priority queue to allocate resources: First, select cities based on their priority (in descending order: High > Medium > Low). Then, for each city, assign resources from the warehouse with the lowest transportation cost.
- Track the remaining resources in warehouses after each allocation to reflect updated quantities.

Task 3: Resource Redistribution Using Binary Heap

- Consider these definitions:
  - Surplus: A warehouse with more than 50 units.
  - Need: A warehouse with fewer than 50 units.

- The strategy is to use surplus warehouses to supply those in need, as long as such warehouses exist.
- Use a max-heap to track warehouses with surplus resources.
- Use a min-heap to track warehouses needing resources.
- Redistribute resources from surplus warehouses to those in need.

Task 4: Dynamic Resource Sharing Among Cities

- Use a union-find (disjoint-set) structure to dynamically manage clusters of cities that share resources.
- **Initialization:** Each city starts in its own cluster. For example, considering the mentioned input scenario we have:

  Clusters: {City A}, {City B}, {City C}

- **Cluster Merging (Union Operation):**
  If two cities have exactly the same resources (e.g., both are supplied by Warehouse X), merge them into the same cluster. For example:

  Clusters: {City A, City B} (both supplied by Warehouse X), {City C}

- **Cluster Query (Find Operation):**
  Check whether two cities belong to the same cluster or not. For example:

  Query: Are City A and City C in the same cluster? – *Result* : `No`.

Expected Scenario for Output

For the mentioned input scenario, We expect the output for each task as follows:

Task 1 and 2: Graph Construction and Priority-Based Resource Allocation

Output:

```
Graph Representation (Cost Matrix):
----------------------------------------------------------------
   cities   | Warehouse 101  | Warehouse 102  | Warehouse 103  |
----------------------------------------------------------------
City A      |     37.58      |     76.66      |     110.15     |
City B      |     27.86      |     61.77      |     95.29      |
City C      |     36.22      |     72.12      |     105.34     |
----------------------------------------------------------------

Allocating resources for City A (Priority: High)
  Allocated 50 units from Warehouse 101
Allocating resources for City B (Priority: Medium)
```

```
  Allocated 30 units from Warehouse 101
Allocating resources for City C (Priority: Low)
  Allocated 20 units from Warehouse 101
  Allocated 30 units from Warehouse 102

Remaining Warehouse Capacities:
  Warehouse 101: 0 units
  Warehouse 102: 20 units
  Warehouse 103: 110 units
```

Task 3: Resource Redistribution Using Heap Structure

Output:

```
Transferred 50 units from Warehouse 103 to Warehouse 101.
Transferred 10 units from Warehouse 103 to Warehouse 102.
Final Resource Levels:
Warehouse 101: 50 units
Warehouse 102: 30 units
Warehouse 103: 50 units
```

Task 4: Dynamic Resource Sharing Among Cities

Output:

```
Initial Clusters:
City A belongs to cluster: 1
City B belongs to cluster: 2
City C belongs to cluster: 3

Merging clusters of City A and City B...
City A belongs to cluster: 1
City B belongs to cluster: 1
City C belongs to cluster: 3

Query: Are City A and City C in the same cluster?
No

Query: Are City A and City B in the same cluster?
Yes

Query: Are City B and City C in the same cluster?
No
```

Test Cases

In addition to the mentioned test case, two test cases named TestCase1.txt and TestCase2.txt are provided to test your code.

Implementation Details

- Implement the solution using three classes:
  - `EmergencySupplyNetwork`: Handles graph representation and resource allocation.
  - `ResourceRedistribution`: Manages resource redistribution using heaps.
  - `DynamicResourceSharing`: Manages clusters of cities using union-find.
- Create a separate `NetworkApp.java` file to:
  - Import and use the above classes.
  - Contain the main method that runs all tasks sequentially.
  - Save all outputs for each testcase in a structured format to a file named `Output_testCase1.json` and `Output_testCase2.json` . An example of the expected output format is provided in file named `example_output.json` to ensure consistency; make sure your output matches the structure and style of the example.
- Use the provided Makefile to compile and run your code.
- Document all code with comments explaining your logic.
- You are permitted to use pre-existing data structures in Java.

Grading System

Your work will be graded based on the following criteria:

- Correct code 10%
- Object-oriented design 20%
- Passes seen test cases 10%
- Passes all unseen test cases 50%
- Cleanliness and readability 10%
- The effectiveness of your codes will not be evaluated.

Detailed Grading Criteria

Correct code: The program solves all formats but may not necessarily find the correct values.

Object-oriented design: The program should follow object-oriented programming principles. For example, minimal separation of information between classes, adherence to interfaces, encapsulation of classes, etc.

Passes all tests: It should pass the tests.

Cleanliness and readability: The code should be clean and properly commented.

Questions

For questions, please post on the TP2 forum on StudiUM or contact the teaching assistants or professor directly:

- Francois Major: `francois.major@umontreal.ca`
- Simon Guy: `simon.guy@umontreal.ca`
- Mohamed Elyes Kanoun: `mohamed.elyes.kanoun@umontreal.ca`
- Morteza Mahdiani: `Morteza.mahdiani@umontreal.com`

Have Fun!