Conception d'un réseau d'approvisionnement d'urgence

IFT2015

27 novembre 2024

Introduction

Dans ce devoir, vous concevrez et implémenterez les structures de données pour gérer un réseau d'approvisionnement d'urgence. L'objectif est de suivre et d'allouer efficacement des ressources aux villes à partir d'un réseau d'entrepôts tout en gérant des mises à jour dynamiques des niveaux d'approvisionnement et des demandes. Cet exercice se concentre sur l'implémentation de structures de données avancées telles que les graphes, les tas et les ensembles disjoints dans un contexte de résolution de problèmes pratiques.

Description du problème

Une organisation internationale de secours en cas de catastrophe met en place un réseau d'approvisionnement d'urgence comprenant :

- **Villes** : chaque ville a un nom, un identifiant unique, Les coordonnées sont représentées comme un vecteur à deux dimensions (x,y), des niveaux de demande et une priorité.
- **Entrepôts** : chaque entrepôt a un identifiant unique, Les coordonnées sont représentées comme un vecteur à deux dimensions (x,y), et une capacité totale.

Les ressources doivent être allouées dynamiquement en fonction des demandes des villes et des approvisionnements des entrepôts, tout en minimisant les coûts de transport.

Exemple de scénario

Configuration du réseau

Villes :

- Ville A : ID = 1, Coordonnées = (2, 3), Demande = 50 unités, Priorité = Haute
- Ville B : ID = 2, Coordonnées = (5, 7), Demande = 30 unités, Priorité = Moyenne
- Ville C : ID = 3, Coordonnées = (8, 2), Demande = 50 unités, Priorité = Faible

Entrepôts :

- Entrepôt X : ID = 101, Coordonnées = (10, 20), Capacité = 100 unités
- Entrepôt Y : ID = 102, Coordonnées = (15, 25), Capacité = 50 unités
- Entrepôt Z : ID = 103, Coordonnées = (20, 35), Capacité = 110 unités

Sélection du mode de transport et calcul des coûts

Sélection du mode de transport :

- Si la distance $d \leq 10$: utiliser un drone (coefficient du mode de transport= 1).
- Si $10 < d \leq 20$: utiliser un camion (coefficient du mode de transport= 2).
- Si $d > 20$: utiliser un train (coefficient du mode de transport= 3).

Calcul des coûts : Le coût est calculé comme suit :

$$\text{Coût} = \text{Distance}(x_1, y_1, x_2, y_2) \times \text{Coefficient du mode de transport}$$

où la distance est la distance euclidienne entre deux nœuds :

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

Tâches du devoir et résultats attendus

1. Construction d'un graphe non-orienté

- Représenter le réseau comme un graphe, avec des arêtes représentant le coût de transport entre les villes et les entrepôts.
- Sortir le graphe sous forme de matrice des coûts.

2. Allocation de ressources basée sur la priorité

- Utiliser une file de priorité pour allouer les ressources : D'abord, sélectionner les villes en fonction de leur priorité (par ordre décroissant : Haute > Moyenne > Faible). Ensuite, pour chaque ville, allouer les ressources depuis l'entrepôt ayant le coût de transport le plus faible.
- Mettre à jour les ressources des entrepôts après chaque allocation pour refléter les quantités actualisées.

3. Redistribution des ressources avec un tas binaire

- Définir les entrepôts en **excédent** (plus de 50 unités) ou en **besoin** (moins de 50 unités).
- Utiliser un tas max pour suivre les entrepôts excédentaires et un tas min pour ceux en besoin.
- Redistribuer les ressources des entrepôts excédentaires à ceux en besoin.

4. Partage dynamique des ressources entre les villes

- Utiliser une structure d'ensemble disjoint (union-find) pour gérer dynamiquement les clusters de villes partageant des ressources.
- Implémenter les opérations d'union (fusion de clusters) et de recherche (vérification d'appartenance à un cluster).

Scénario attendu pour la sortie

Tâches 1 et 2 : Construction du graphe et allocation des ressources

Sortie :

```
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
```

Tâche 3 : Redistribution des ressources

Sortie :

```
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
```

Tâche 4 : Partage dynamique des ressources entre les villes

Sortie :

```
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
```

Cas de test

En plus du cas de test mentionné, deux cas de test nommés TestCase1.txt et TestCase2.txt sont fournis pour tester votre code.

Implémentation

- Implémentez trois classes :
  - **EmergencySupplyNetwork** : gère la représentation en graphe et l'allocation des ressources.
  - **ResourceRedistribution** : gère la redistribution avec des tas.

  - **DynamicResourceSharing** : gère les clusters de villes avec union-find.

- Créez un fichier `NetworkApp.java` pour :
  - Importer et utiliser les classes.
  - Exécuter toutes les tâches.
  - Enregistrez tous les résultats pour chaque cas de test dans un format structuré dans des fichiers nommés `Output_testCase1.json` et `Output_testCase2.json`. Un exemple du format de sortie attendu est fourni dans le fichier nommé `example_output.json` pour garantir la cohérence ; assurez-vous que votre sortie respecte la structure et le style de l'exemple.
- Utilisez le `Makefile` fourni pour compiler et exécuter le code.
- Documentez tout le code avec des commentaires expliquant votre logique.

Barème de notation

- Code correct : 10 %
- Conception orientée objet : 20 %
- Cas de test visibles passés : 10 %
- Cas de test non visibles passés : 50 %
- Propreté et lisibilité du code : 10 %
- L'efficacité de vos codes ne sera pas évaluée.

Questions ?

Pour des questions, veuillez poster sur le forum TP2 sur StudiUM ou contacter directement les assistants d'enseignement ou le professeur :

- François Major : `francois.major@umontreal.ca`
- Simon Guy : `simon.guy@umontreal.ca`

- Mohamed Elyes Kanoun : mohamed.elyes.kanoun@umontreal.ca
- Morteza Mahdiani : morteza.mahdiani@umontreal.ca

Bonne chance !