SAE S2.02 – Rapport pour la ressource Graphes

Zhang Dylan , Marquis Nathan, Flagothier Lorenzo, groupe B3

Version 1 : un seul moyen de transport

Cette section traite uniquement de la Version 1 du projet.

Présentation d'un exemple

Alice vit à Paris et doit se rendre à une conférence à Lyon. En chemin, elle doit s'arrêter à Dijon et à Macon pour des réunions. Elle utilise une plateforme de planification de trajets pour déterminer les meilleures options de transport. Les moyens de transport disponibles incluent le train, le bus, et l'avion. Alice peut changer de moyen de transport entre chaque étape du voyage.

Moyens de transport disponible

- **Train :**
  - Paris -> Dijon : 1h30, 30€, 10 kg/CO2e
  - Dijon -> Macon : 1h, 15€, 5 kg/CO2e
  - Macon -> Lyon : 45min, 10€, 3 kg/CO2e
- **Bus :**
  - Paris -> Dijon : 3h, 10€, 20 kg/CO2e
  - Dijon -> Macon : 2h, 5€, 10 kg/CO2e
  - Macon -> Lyon : 1h30, 5€, 8 kg/CO2e
- **Avion :** (utilisable uniquement pour Paris -> Lyon)
  - Paris -> Lyon : 1h (ajouter 2h pour embarquement et débarquement), 100€, 150 kg/CO2e

Critère de Durée Totale

- **Train :** 3h15
- **Bus :** 6h30
- **Combinaison Train + Bus :** 5h

Critère de Coût

- **Bus :** 20€
- **Combinaison Train + Bus :** 40€
- **Train :** 55€

Critère de Coût Énergétique

- **Train :** 18 kg/CO2e
- **Combinaison Train + Bus :** 28 kg/CO2e
- **Bus :** 38 kg/CO2e

Meilleurs Itinéraires et Pourquoi

1. **Train (toutes les étapes) :**
   - **Avantages :** Durée totale la plus courte (3h15), coût énergétique le plus bas (18 kg/CO2e), ponctualité, pas de changement de mode de transport.
   - **Inconvénients :** Coût plus élevé (55€).
2. **Combinaison Train + Bus :**
   - **Avantages :** Durée relativement courte (5h), coût modéré (40€), coût énergétique raisonnable (28 kg/CO2e).
   - **Inconvénients :** Nécessite un changement de mode de transport, potentiellement moins confortable.

Conclusion pour Alice Pour Alice, le **train (toutes les étapes)** est le meilleur choix en termes de durée totale, ce qui est son critère principal. Bien que plus coûteux, il offre une durée de voyage la plus courte sans besoin de changer de mode de transport et a également le coût énergétique le plus bas. La **combinaison Train + Bus**, bien que légèrement plus longue, offre une alternative raisonnable en termes de coût et de flexibilité.

Résumé des Itinéraires

- **Train :** Paris -> Dijon (1h30), Dijon -> Macon (1h), Macon -> Lyon (45min), coût 55€, coût énergétique 18 kg/CO2e, durée totale 3h15.
- **Bus :** Paris -> Dijon (3h), Dijon -> Macon (2h), Macon -> Lyon (1h30), coût 20€, coût énergétique 38 kg/CO2e, durée totale 6h30.
- **Combinaison Train + Bus :** Paris -> Dijon (Train, 1h30), Dijon -> Macon (Bus, 2h), Macon -> Lyon (Bus, 1h30), coût 40€, coût énergétique 28 kg/CO2e, durée totale 5h.

Ainsi, Alice choisira probablement de prendre le **train pour toutes les étapes** de son voyage à Lyon, malgré le coût plus élevé, afin de minimiser la durée totale du voyage tout en réduisant son empreinte carbone.

Modèle pour l'exemple

Modélisation pour la Version 1 dans le cas général

Pour résoudre le problème on doit définir un graphe non orienté avec pour chaque sommet qui représente une ville.

1. Sommets du graphe - Paris - Dijon - Macon - Lyon

Ces sommets représente le graphe donné en exemple

2. Arêtes du graphe - Parmi les arêtes, chaque arêtes représentent un segment de trajet entre deux sommets (villes), avec un moyen de transport spécifique. - Par exemple, il y a une arête entre Paris et Dijon pour le train, une autre arête entre Paris et Dijon pour le bus, etc.

3. Poids des arêtes - Les poids des arêtes peuvent représenter différents critères d'optimisation, tels que le temps de trajet, le coût monétaire, ou les émissions de CO2. - Par exemple, pour le critère de durée totale, le poids de l'arête Paris -> Dijon en train serait 1h30.

4. L'algorithme - L'algorithme de Dijkstra est bien adapté pour trouver le chemin le plus court dans un graphe avec des poids non négatifs, comme dans notre cas où les poids représentent le temps de trajet. - Créer un tableau dist pour stocker la distance minimale de la source (Paris) à chaque sommet, initialisée à l'infini sauf pour la source elle-même, qui est à 0. - Utiliser une file de priorité pour gérer les sommets à explorer, initialement contenant seulement la source. - Tant que la file de priorité n'est pas vide - Extraire le sommet u avec la plus petite distance. - Pour chaque voisin v de u, calculer la distance potentielle alt = dist[u] + poids(u, v). - Si alt est plus petit que la distance actuelle dist[v], mettre à jour dist[v] et insérer v dans la file de priorité. - Les distances minimales de Paris à tous les autres sommets sont trouvées.

```
Dijkstra(G, source):
    dist[source] = 0
    pour chaque sommet v dans G:
        si v != source:
            dist[v] = infini
        ajouter v à la file de priorité Q avec la priorité dist[v]

    tant que Q n'est pas vide:
        u = extraire sommet avec la plus petite distance de Q
        pour chaque voisin v de u:
            alt = dist[u] + poids(u, v)
            si alt < dist[v]:
                dist[v] = alt
                mettre à jour la priorité de v dans Q

    retourner dist
```

Implémentation de la Version 1

Le nom de la classe test : **UsePlateformeAlice**,

L'identité du commit : **ce13a900727281d8c90ed4e7ced45d1d67c0d69f**

La date : **21 Mai 2024 at 3:31:50 PM**

Le lien pour accéder : **src/UsePlateformeAlice.java**.

Version 2 : multimodalité et prise en compte des correspondances

---

Présentation d'un exemple

Même exemple que la v1

Alice vit à Paris et doit se rendre à une conférence à Lyon. En chemin, elle doit s'arrêter à Dijon et à Macon pour des réunions. Elle utilise une plateforme de planification de trajets pour déterminer les meilleures options de transport. Les moyens de transport disponibles incluent le train, le bus, et l'avion. Alice peut changer de moyen de transport entre chaque étape du voyage.

Modèle pour l'exemple

Modélisation pour la Version 2 dans le cas général

Pour les sommets et les arètes on garde la v1

Adaptation pour inclure les coûts de correspondance Modéliser chaque sommet comme une paire (ville, moyen de transport). Les arêtes incluront les trajets directs et les correspondances avec leurs coûts respectifs.

```
Dijkstra(G, source):
    dist[source] = 0
    pour chaque sommet (ville, transport) dans G:
        si (ville, transport) != source:
            dist[(ville, transport)] = infini
        ajouter (ville, transport) à la file de priorité Q avec la priorité dist[(ville, tra

    tant que Q n'est pas vide:
        (ville_u, transport_u) = extraire sommet avec la plus petite distance de Q
        pour chaque voisin (ville_v, transport_v) de (ville_u, transport_u):
            alt = dist[(ville_u, transport_u)] + poids((ville_u, transport_u), (ville_v, tra
            si alt < dist[(ville_v, transport_v)]:
                dist[(ville_v, transport_v)] = alt
                mettre à jour la priorité de (ville_v, transport_v) dans Q

    retourner dist

### Implémentation de la Version 2
```

Initialisation :

dist[source] est initialisée à 0, les autres distances sont initialisées à l'infini. Chaque sommet (ville, transport) est ajouté à la file de priorité avec la priorité dist[(ville, transport)].

Le nom de la classe test : **UsePlateformeAlice**

L'identité du commit : **bbcd88b4e36a03721d3fa8fb6254064972287630**

La date : **June 7, 2024**

Le lien pour accéder : **src/UsePlateformeAlice.java**.

Version 3 : optimisation multi-critères

Fin du rapport

Barème sur 30 pts

Toute question sur le barème est à adresser à iovka.boneva@univ-lille.fr

- Rapport non rendu à temps -> note 0
- **(7, décomposé comme suit)** Divers
  - **(1,5)** Respect de la structure du rapport
  - **(1,5)** Section Version 1 rendue pour le 18/05/2024. Cette version peut contenir les parties en italique.
  - **(1,5)** Section Version 2 rendue pour le 08/06/2024. Cette version peut contenir les parties en italique.
  - **(1)** Utilisation de vocabulaire précis sur les graphes (termes vu en cours, noms des algorithmes, etc.)
  - **(1,5)** Style d'écriture fluide et compréhensible
- **(8, décomposé comme suit)** Solution pour la Version 1
  - **(2)** Exemple pertinent (illustre tous les aspects du problème) et lisible (en particulier, ni trop grand ni trop petit, bien présenté)
  - **(4)** Le modèle de l'exemple permet de trouver la solution sur l'exemple. La modélisation pour le cas général permet de résoudre le problème posé
  - **(2)** L'implémentation de l'exemple est correcte et fonctionnelle
- **(6, décomposé comme suit)** Solution pour la Version 2
  - **(1)** Exemple pertinent
  - **(4)** le modèle de l'exemple permet de trouver la solution sur l'exemple. La modélisation pour le cas général permet de résoudre le problème posé
  - **(1)** L'implémentation de l'exemple est correcte et fonctionnelle
- **(3)** Qualité de la description de la solution (concerne les sections "Modèlisation dans le cas général" pour les Versions 1 et 2):
  - La modélisation pour le cas général est décrite de manière abstraite mais précise et complète. Pour vous donner une idée, un · e étudiant · e de BUT qui a validé les ressources Graphes et Dev devrait être en mesure d'implémenter votre solution d'après la description que vous en faites, sans avoir à trop réfléchir.
- **(6)** Solution pour la Version 3: mêmes critères que pour la Version 2