SAÉ 2.02

Exploration algorithmique d'un problème

KEISER Kenan S2C

EDOUARD Justin S2C

Voir les annexes dans la section annexe

2. Représentation d'un graphe

Ce qu'on a fait :

- Création de la classe Arc avec un nœud destination (String) et un coût (double).
- Création de la classe Arcs, qui contient une liste d'objets Arc et des méthodes pour ajouter et lire les arcs.
- Création de l'interface Graphe avec deux méthodes : listeNoeuds() et suivants(n).
- Implémentation de cette interface dans la classe GrapheListe, en utilisant deux listes :
  - noeuds : liste des noms de nœuds,
  - adjacence : liste d'objets Arcs.
- Méthode ajouterArc() ajoutant un arc dans le graphe et les nœuds si besoin.
- Création d'une méthode main() pour construire le graphe de l'exemple du sujet.
- Ajout d'une méthode toString() pour afficher le graphe comme dans l'exemple.
- Tests unitaires pour vérifier que le graphe est correctement construit.

3. Calcul du plus court chemin par point fixe

Ce qu'on a fait :

- Écriture de l'algorithme du point fixe (Bellman-Ford) en pseudo-code (cf voir le fichier java BellmanFord pour voir l'algo ou voir annexe 1).
- Utilisation de la classe Valeurs pour stocker les distances (L(X)) et parents.

- **Implémentation dans la classe BellmanFord de la méthode resoudre(Graphe g, String depart) qui applique l'algorithme.**
- **Ajout d'une méthode main() pour lancer l'algorithme et afficher les distances et parents.**
- **Création d'un test unitaire pour vérifier que les distances et parents sont corrects.**
- **Ajout d'une méthode calculerChemin(destination) dans la classe Valeurs pour retrouver le chemin le plus court.**

4. Calcul du plus court chemin par Dijkstra

Ce qu'on a fait :

- **Traduction de l'algorithme de Dijkstra (en pseudo-code) en Java dans la classe Dijkstra avec la méthode resoudre(Graphe g, String depart).**
- **Ajout de tests unitaires pour vérifier le bon fonctionnement de l'algorithme.**
- **Création de la classe MainDijkstra pour tester l'algorithme sur un graphe d'exemple.**

5. Validation et expérimentation

Ce qu'on a fait :

- **Ajout d'un constructeur dans GrapheListe pour charger un graphe depuis un fichier texte.**
- **Comparaison des temps d'exécution entre Dijkstra et Bellman-Ford sur plusieurs graphes.**
- **Conclusion : Dijkstra est plus rapide dans la majorité des cas, lors des exécutions, la différence n'était pas flagrante sur nos graphes testés mais bien présente.**
- **Ce résultat n'est pas étonnant puisque Dijkstra utilise une file de priorité pour n'explorer chaque sommet et arête qu'une seule fois.**

6. Application : métro parisien

Ce qu'on a fait :

- **Modification de la classe Arc pour ajouter le numéro de ligne du métro.**
- **Création de la classe LireReseau avec une méthode statique lire(String fichier) pour lire un graphe métro depuis un fichier.**
- **Implémentation de la classe MainMetro pour tester 5 trajets (Voir annexe 2) avec les deux algorithmes et afficher :**
  - **la station de départ et d'arrivée,**
  - **le chemin,**
  - **le temps de calcul de chaque algorithme.**
- **Création d'une nouvelle version resoudre2() qui ajoute une pénalité de 10 unités si on change de ligne.**
- **Ajout des trajets avec pénalité dans MainMetro (Voir annexe 3).**
- **Comparaison des trajets obtenus avec ceux proposés par le site de la RATP (Voir annexe 4).**

7. Conclusion générale

Grâce à cette SAE, nous avons appris à :

- **représenter un graphe orienté pondéré en Java ;**
- **implémenter deux algorithmes classiques de plus court chemin ;**
- **comparer leurs performances ;**
- **adapter ces algorithmes à un cas réel (réseau de métro).**

Difficultés rencontrées :

- **comprendre la structure de GrapheListe ;**
- **gérer les indices entre les listes de nœuds et d'arcs ;**
- **bien comparer les performances.**

Annexe

Annexe 1

```
/**
 * Algo Point Fixe
 *
 * fonction pointFixe(G : graphe, depart : chaîne) : rien
 *    début
 *       pour chaque X dans G faire
 *          L[X] ← +∞
 *          parent[X] ← ""
 *       fpour
 *
 *       L[depart] ← 0
 *       parent[depart] ← ""
 *
 *       modifie ← vrai
 *
 *       tant que modifie = vrai faire
 *          modifie ← faux
 *
 *          pour chaque X dans G faire
 *             voisins ← voisins de X dans G
 *
 *             pour chaque N dans voisins faire
 *                coutArc ← coût de l'arc de X à N dans G
 *                nouvelleValeur ← L[X] + coutArc
```

```
*
*                 si nouvelleValeur < L[N] alors
*                   L[N] ← nouvelleValeur
*                   parent[N] ← X
*                   modifie ← vrai
*                 fsi
*               fpour
*           fpour
*       ftant
*   fin
*
* Lexique:
* G: graphe orienté
* X: chaîne
* N: chaîne
* depart: chaîne
* L: table de réels (associée à chaque nœud)
* parent: table de chaînes (associée à chaque nœud)
* voisins: tableau de chaînes
* nouvelleValeur: réel
* coutArc: réel
* modifie: booléen
*
*/
```

Annexe 2

Départ	Arrivé	Chemin	Temps calcul Bellman-Ford en millisecondes	Temps calcul Dijkstra en millisecondes
La Défense	Châtelet	La Défense - Châtelet	14	4
Gare du Nord	Gare de Lyon	Gare du Nord - Gare de l'Est - Jacques Bonsergent - République - Filles du Calvaire - (Saint-Sébastien-Froissart) - Chemin Vert - Bastille - Gare de Lyon	27	25
Montparnasse Bienvenue	Saint-Lazare	Montparnasse Bienvenue-(Notre-Dame-des-Champs)-Rennes- Sèvres Babylone- Rue du Bac-Solférino-Assemblée Nationale-Concorde-Madeleine-(Saint-Lazare)	2	2
Nation	La Défense	Nation - La Défense	28	23
Châtelet	Gare du Nord	Châtelet - Les Halles - Étienne Marcel - Réaumur Sébastopol - (Strasbourg Saint-Denis) -	21	22

		Château d'Eau - Gare de l'Est - Gare du Nord		

Annexe 3

//Penalite

Départ	Arrivé	Chemin	Temps calcul Bellman-Ford en millisecondes	Temps calcul Dijkstra en millisecondes
La Défense	Châtelet	La Défense - Châtelet	2	1
Gare du Nord	Gare de Lyon	Gare du Nord - Gare de l'Est - Jacques Bonsergent - République - Filles du Calvaire - (Saint-Sébastien-Froissart) - Chemin Vert - Bastille - Gare de Lyon	221	16
Montparnasse Bienvenue	Saint-Lazare	Montparnasse Bienvenue-(Notre-Dame-des-Champs)-Rennes- Sèvres Babylone- Rue du Bac-Solférino-Assemblée Nationale-Concorde-Madeleine-(Saint-Lazare)	1	1
Nation	La Défense	Nation - La Défense	89	13
Châtelet	Gare du Nord	Châtelet - Les Halles - Étienne Marcel - Réaumur Sébastopol - (Strasbourg Saint-Denis) -	68	9

		Château d'Eau - Gare de l'Est - Gare du Nord		

Annexe 4

Départ	Arrivé	RATP
La Défense	Châtelet	Trajet direct, similaire aux résultats précédent
Gare du Nord	Gare de Lyon	Gare du Nord - Châtelet - Gare de Lyon, est le trajet le plus court proposé par la RATP, on peut voir une énorme différence, notre trajet calculé précédemment nous faisant passer par bien plus de station
Montparnasse Bienvenue	Saint-Lazare	Un trajet direct, est le trajet le plus court proposé par la RATP, on peut voir une énorme différence, notre trajet calculé précédemment nous faisant passer par bien plus de station
Nation	La Défense	Trajet direct, similaire aux résultats précédent
Châtelet	Gare du Nord	Châtelet - Gare du Nord, est le trajet le plus court proposé par la RATP, on peut voir une énorme différence, notre trajet calculé précédemment nous faisant passer par bien plus de station

Nous pouvons donc constater une nette différence, ainsi, nos algorithmes nous donne des trajet semblant bien plus long que ceux donner par la RATP, cela est sans doute dû à un problème de notre part, lors du recalcul de la pénalité, nos chemins doivent être sans doute faussées, puisque comme on peut le constater, nos chemins ne changent pas.

Nous pouvons tout de même constater que la RATP donne un chemin relativement efficace prennant en compte les départ et arrêt de chaque ligne, y compris lors des changements de ligne pour donner le chemin le plus efficace possible.

Annexe 5

Diagramm de classe représentant l'ensemble des liaisons entre nos classes