Bellman - MPM S2.02 Graphe

Bellman – MPM : Méthode des Potentiels Métra

1. Graphes Orientés

Vous devez tout d'abord définir des classes qui permettront de représenter un **réseau de livraison par drones** sous forme d'un **graphe orienté pondéré**. Puis vous devez définir des classes qui permettront de représenter un **graphe orienté pondéré et sans circuit (DAG)** intégrant le tri topologique, pour gérer l'ordre des livraisons et les dépendances entre les missions.

On veut pouvoir disposer de **deux constructeurs** :

- Un **constructeur sans paramètre** qui crée un graphe vide.
- Un **constructeur avec un paramètre** de type chaîne de caractères qui est le chemin vers un fichier texte contenant une description du graphe.

On veut pouvoir effectuer a minima les **opérations suivantes** sur le graphe :

• redimensionner le nombre de sommets (ajout / retrait, dans le cas du retrait, on retire les sommets de plus grand indice);
• récupérer le nombre de sommets ;
• récupérer le nombre d'arcs ;
• ajouter un arc d'un sommet i vers un sommet j avec un poids p ;
• retirer un arc d'un sommet i vers un sommet j ;
• tester si un arc (i, j) existe dans le graphe ;
• récupérer le poids d'un arc (i, j) ;
• récupérer les successeurs d'un sommet ;
• récupérer les prédécesseurs d'un sommet ;
• récupérer les voisins d'un sommet ;
• sauvegardez le graphe dans un fichier texte dont on fournit le chemin.

Exemples de fichiers (suffixé par `.gr`) :

```
dag1.gr
5 7
1 2 7
2 0 5
...
```

La première ligne contient **le nombre de points de livraison (sommets) et le nombre de trajets disponibles (arcs)**. Les points de livraisons sont numérotés de `0` à `n - 1` (n étant le nombre de sommets). Chaque ligne à partir de la deuxième contient la description d'un trajet : **point de départ, point d'arrivée, durée estimée du trajet**.

Indication : pour tester vos algorithmes, vous êtes encouragés à créer vos propres fichiers contenant des graphes de petites tailles. Vous pouvez par exemple utiliser les graphes vus en cours de Graphes.

Production : un diagramme de classe UML.

Lecture / écriture d'un fichier texte

Module fs de Node
Pour lire un fichier, le plus simple est d'utiliser le module `fs` de Node. Ce module est installé par défaut. Les fonctions dont vous aurez besoin sont importées par l'instruction :

```
import { readFileSync, createWriteStream } from "fs";
```

Lecture d'un fichier
L'instruction suivante lit le contenu d'un fichier et stocke chaque ligne comme une chaîne de caractères dans un tableau de chaînes de caractères :

```
let lignes = readFileSync("~/Instances/Random4-n/Random4-n.21.0.gr",
"utf8").split("\n");
```

Le premier argument de la fonction `readFileSync` est une chaîne de caractères contenant le chemin jusqu'au fichier que l'on souhaite lire. Le second argument est l'encodage du fichier.
La fonction `readFileSync` retourne le contenu du fichier sous la forme d'une chaîne de caractères.
La méthode `split` des chaînes de caractères permet de découper une chaîne de caractères en plusieurs chaînes en se basant sur un caractère comme délimiteur, ici le retour à la ligne.

Indication : si vous voulez découper une chaîne par rapport aux espaces par exemple, vous pouvez le faire avec la méthode `split`. Après les instructions :

```
let chaîne = "a 1 3";
let info = chaîne.split(" ");
```

la variable info sera un tableau de chaînes de caracteres égal à `['a' , '1' , '3' ]`.

Ecriture dans un fichier Pour ecrire dans un fichier, on commence par ouvrir un flux vers un fichier avec la fonction `createWriteStream` dont l'argument est une chaîne de caractères contenant le chemin vers le fichier :

```
let output = createWriteStream("IOFiles/output.txt") ;
```

Attention : cela efface le contenu du fichier s'il existe deja. Ensuite, la méthode `write` des flux d'ecriture permet d'écrire dedans :

```
output.write("5 7\n") ;
output.write("1 2 7\n") ;
output.write("2 0 5\n") ;
```

Lorsque l'écriture dans le fichier est terminée, il faut fermer le flux : `output.end() ;`

Implémentation en TypeScript

Dans cette partie, il faut implémenter les classes que vous avez définies ci-dessus. Il faut qu'il y ait une concordance entre votre diagramme de classes et votre code. Certaines actions ne sont pas possibles. Il vous faudra decider par exemple de ce que vous voulez faire si l'utilisateur ajoute un arc entre deux sommets et que l'un des sommets n'existe pas.

Production : le code des classes + un court rapport expliquant comment vous avez définies les actions comme l'accès a des sommets qui n'existent pas.

---

2. Algorithme de Bellman

Il s'agit d'implémenter **les 4 variantes suivantes de l'algorithme de Bellman** appliqué à la logistique de drones :

- **Calcul du plus court chemin** pour un drone devant livrer un colis depuis un point de départ donné.
- **Calcul du plus long chemin** pour déterminer **le trajet le plus coûteux en temps**, ce qui est utile pour détecter les livraisons critiques.
- **Recherche de l'arborescence des trajets optimaux** pour un drone partant d'un hub central vers plusieurs destinations.
- **Recherche de l'anti-arborescence** pour identifier **le chemin optimal aboutissant à un hub spécifique**.

---

3. Méthode des Potentiels Métra (MPM)

Il s'agit d'implémenter la **méthode MPM** en utilisant **2 des variantes de l'algorithme de Bellman**.

Les résultats fournis seront :

- **Les horaires de départ au plus tôt** pour chaque drone.
- **La durée minimale** pour réaliser toutes les livraisons.
- **Les horaires de début au plus tard**, pour garantir les délais de livraison.
- **Les marges libres et totales** qui détermine les livraisons qui peuvent être ajustées sans impacter la mission globale
- **Les trajets critiques** et **un chemin critique** pour les livraisons prioritaires, qui ne peuvent être en retard.

Vous devez définir des classes qui permettront de **représenter un graphe de missions de drones**.

Format de fichier contenant un jeu de données (suffixé par `.mpm`) :

```
missions_drones.mpm
5
1 2
2 5 1 3
...
```

- La première ligne contient **le nombre total de livraisons**.
- Les livraisons sont numérotées de `1` à `n` (n étant le nombre de tâches).
- Chaque ligne à partir de la deuxième décrit une livraison : **numéro de livraison, durée, numéros des tâches qui la précèdent** (s'il y en a).

Production : les codes 4 variantes suivantes de l'algorithme de Bellman + la méthode MPM et ses résultats + fichier de tests unitaires + les fichiers / fonctions necessaires au test.

Consignes

• Le travail se fait en binome.
• Le travail doit etre rendu via un depot git privé (vous n'y mettrez pas les fichiers de graphes)
• Votre depot ne doit pas etre cree a la fin avec un unique upload des fichiers.
• Vous veillerez a mettre des textes lies aux commit pertinents.
• Le depot git devra faire apparaître différentes étapes du travail (utilisation de tag) et une vraie collaboration.
• Vous ajouterez à votre dépot vos enseignants de developpement objet.
• La lisibilité du code sera prise en compte.
• Votre code doit lever que des exceptions que vous avez prévu avec un message d'erreur informatif pour l'utilisateur.