s201-02

Votre rapport doit suivre le plan donné dans ce document.

Contraintes à respecter pour le rapport

- format: Markdown qu'on peut lire sur gitlab, ou pdf, ou html
- rapport dans un répertoire `graphes` à la racine du dépôt git
- rapport prêt le 21/06/2024; aucun délai supplémentaire ne sera accordé quelle que soit la raison donnée. Concrètement, on va récupérer la dernier daté au plus tard le 21/06/2024 et on ne verra même pas de version ultérieure du rapport, si elles existent. Minuit et une minute du 22/06/2024 sera trop tard
- respecte le plan donné ci-dessous
- garder les explications *en italique* jusque la fin pour s'y référer en écrivant le rapport
- supprimer les explications *en italique* juste avant de rendre la version finale du rapport
- le rapport est un texte **rédigé** avec des phrases intelligibles (on ne se contente pas de répondre laconiquement aux questions posées)

Idéalement, le rapport est rédigé au fur et à mesure avec le calendrier donné dans le sujet:

- section Version 1 faite avant le 18/05/2024 (1pt/20 si c'est le cas)
- section Version 2 faite avant le 08/06/2024 (1pt/20 si c'est le cas)

Finalement, l'utilisation d'un outils de génération de langage est autorisées, à condition de le faire intelligemment. En particulier, veillez à:

- avoir un rapport cohérent avec un style cohérent sur la totalité du document (niveau de langage, richesse du vocabulaire, termes utilisés, verbosité, ...)
- un rapport trop verbeux est fastidieux à lire. Si vous utilisez un outil pour faire du texte verbeux inutile, on utilisera un outil pour en faire un résumé et on corrigera uniquement le résumé
- les outils de génération insèrent parfois des phrases ne faisant pas partie du texte, mais qui s'adressent à l'interlocuteur (par exemple, pour vous informer que la limite de 2000 tokens est atteinte). La présence de telles phrases dans le rapport indique que vous n'avez pas relu et sera lourdement pénalisée.

Début du rapport Tout ce qui précède sera enlevé pour la version finale

---

SAE S2.02 -- Rapport pour la ressource Graphes

Noms des auteurs, groupe Fryson Adrien et Roget Benjamin, Groupe B6

Version 1 : un seul moyen de transport

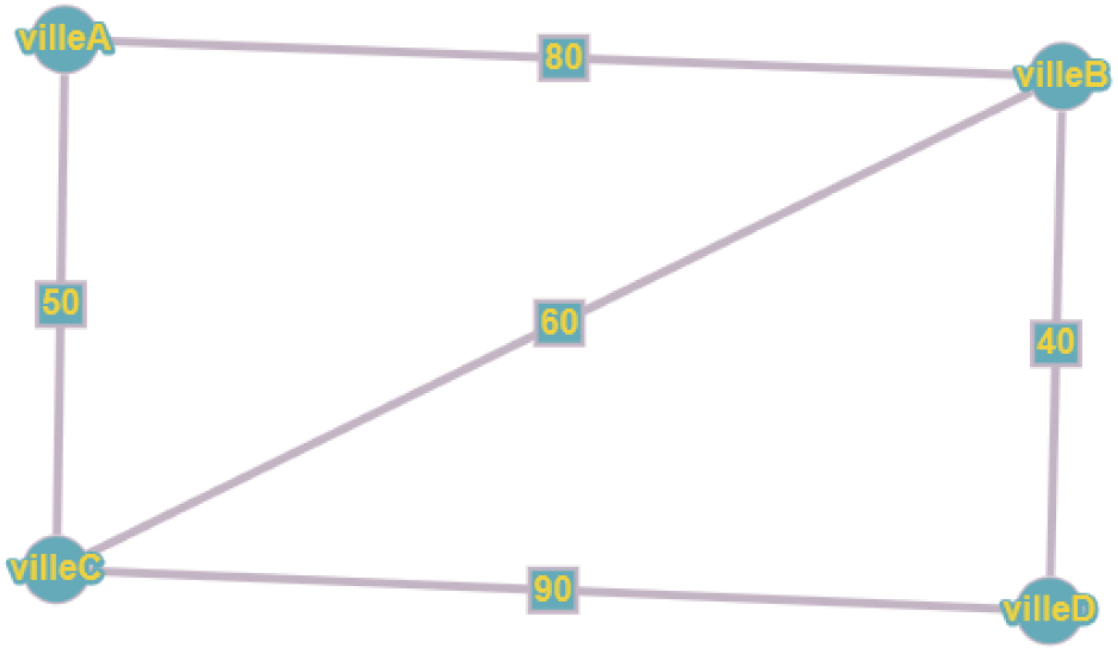
Cette section traite uniquement de la Version 1 du projet.

Présentation d'un exemple

Présenter un exemple concret de problème (données complètes pour la plateforme avec tous les moyens de transport, préférences de l'utilisatrices qui comprennent le moyen de transport choisi, le critère d'optimisation, et nombre d'itinéraires demandés). Donner la solution du problème du point de vue de l'utilisatrice, càd quels sont les itinéraires possibles, quels sont les meilleurs itinéraires et pourquoi. Pour l'instant on ne parle pas de graphes; on peut éventuellement faire des schémas.

Prenons comme exemple un utilisateur pressé, car sa voiture vient de tomber en panne. Il souhaite atteindre le plus rapidement possible la ville D où se trouve son lieu de travail. Il se trouve actuellement dans la ville A. De plus, l'utilisateur a peur de l'avion, il ne peut donc emprunter que le train.

Les choix à sa disposition peuvent être illustrés comme suit :

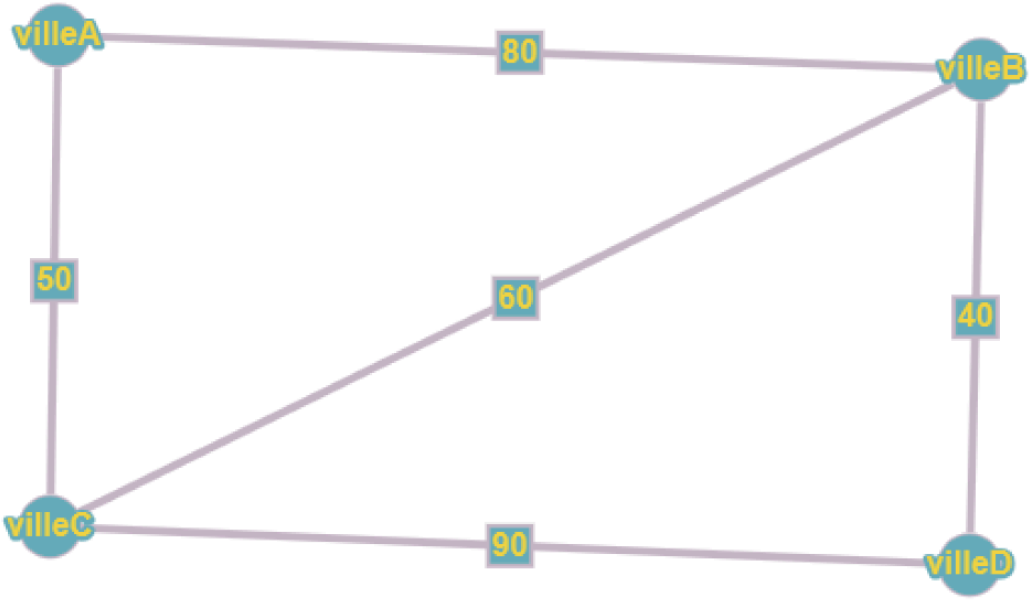


Son meilleur choix ici serait de passer par la ville B pour arriver ensuite à la ville D. Il aurait alors un trajet de 120 minutes en évitant l'avion. Les trajets les plus courts en termes de nombre d'étapes sont au minimum de 2 étapes, et le temps minimum pour un trajet de 2 étapes est de 120 minutes. L'autre chemin possible de 2 étapes prend 140 minutes.

On pourrait imaginer passer par un chemin avec 3 étapes, mais les deux choix disponibles ont un temps minimum de 150 minutes. Le seul choix logique est donc le chemin de 120 minutes.

Modèle pour l'exemple

Donner le graphe modélisant l'exemple ci-dessus.



Donner la solution du problème (càd les meilleurs itinéraires) en tant que chemins dans le graphe. Les quatres itinéraire possible sont :

1. TRAIN de villeA à villeD en passant par villeB totale: 120.0 min [A,B,D]
2. TRAIN de villeA à villeD en passant par villeC totale: 140.0 min [A,C,D]
3. TRAIN de villeA à villeD en passant par villeC, villeB totale: 150.0 min [A,C,B,D]
4. TRAIN de villeA à villeD en passant par villeB, villeC totale: 230.0 min [A,B,C,D]

Modélisation pour la Version 1 dans le cas général

Expliquer de manière abstraite comment, étant donné un problème de recherche d'itinéraire (plateforme avec tous types de lignes, moyen de transport choisi, critère d'optimisation, nombre d'itinéraires demandés) on peut construire un graphe permettant de résoudre le problème de recherche d'itinéraire. C'est à dire:

- *quels sont les sommets du graphe par rapport aux données du problème*,
- *quelles sont ses arêtes, par rapport aux données du problème*,
- *comment sont définis les poids des arêtes*,
- *quel algorithme sur les graphes permet de résoudre le problème d'itinéraire (nom de l'algorithme, arguments).*

Utiliser un vocabulaire précis sur les graphes.

Les sommets inclus dans le graphe sont les mêmes que dans les données, et les arêtes sont initialement les mêmes que dans les données. Lorsque l'utilisateur indique qu'il ne souhaite utiliser que le train, le graphe est recréé pour n'inclure que les arêtes utilisant ce moyen de transport. Pour modéliser ce problème, nous simulons un unique graphe qui utilise trois autres graphes pour résoudre les problèmes. Chaque sous-graphe représente un critère, ce qui permet que le poids des arêtes reste fidèle aux données initiales.

L'algorithme qui doit être utilisé pour résoudre ce problème est l'algorithme de Dijkstra. Il prend en paramètre un graphe avec uniquement des poids positifs et un sommet de départ.

Implémentation de la Version 1

Écrire une classe de test qui reprend l'exemple, définit toutes les données de la plateforme, construit le graphe et calcule la solution. Votre classe peut utiliser des assertions (test unitaire) ou bien afficher la solution. Donner ici le VoyageurTest.java, ***18/05/2024 b2d69ddab6a21c695e3e3c46b06dad4157fdb7d2*** *et un lien [tests/graphes/v1/VoyageurTest.java · b2d69ddab6a21c695e3e3c46b06dad4157fdb7d2 · SAE2.01-2.02 / 2024 / B6 · GitLab (univ-lille.fr)]*

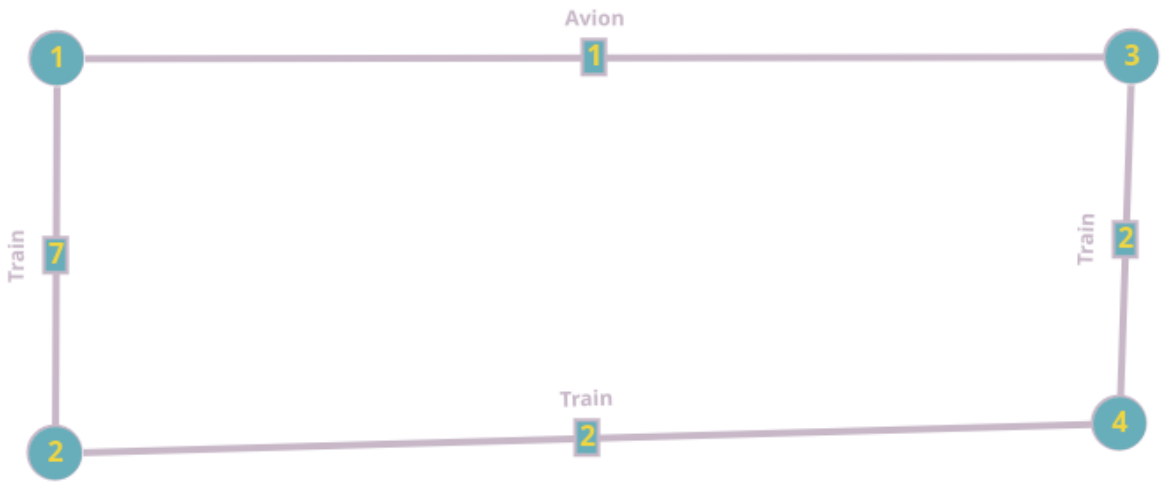
On insiste sur l'importance de spécifier le commit. En effet, quand vous commencerez la Version 2, le code utilisé pour le test de la Version 1 sera modifié. Il se peut que vous n'ayez pas le temps de finaliser la Version 2 et vous retrouver avec un code qui ne marche pas même pour la Version 1. C'est pourquoi il est important de rédiger le rapport au fur et à mesure et de donner ici un lien vers la version de votre code qui marche pour la Version 1 du projet.

Version 2 : multimodalité et prise en compte des correspondances

Cette section explique la solution pour la Version 2 du projet.

Présentation d'un exemple

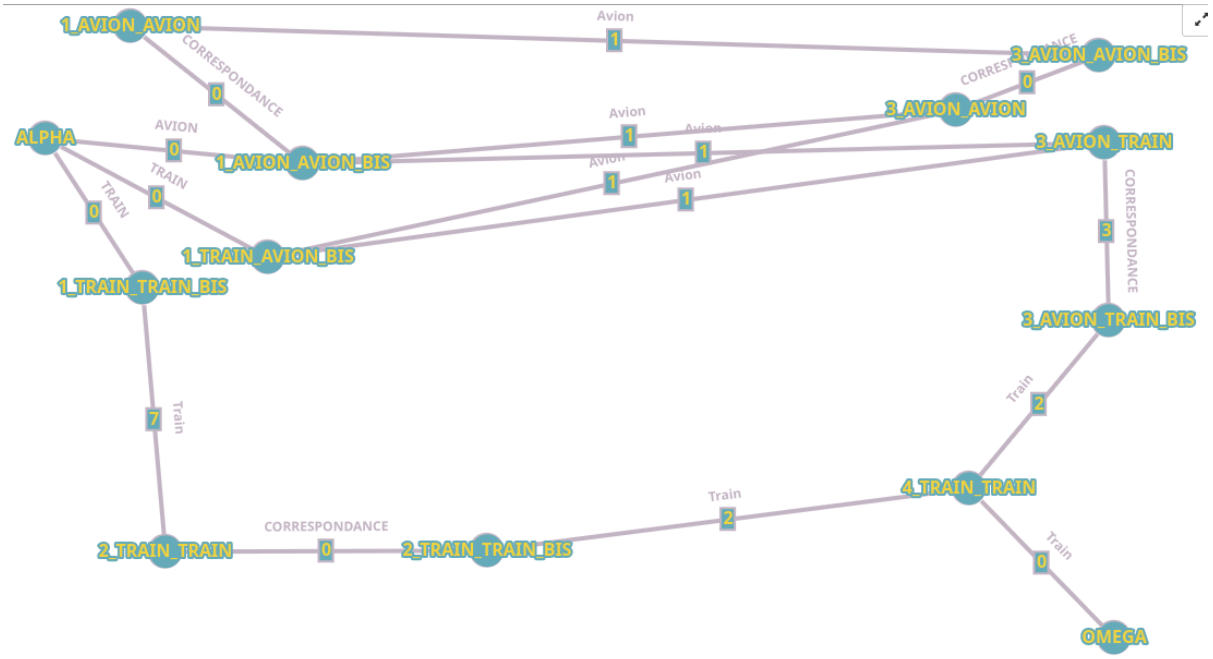
Présenter un exemple concret (plateforme, coûts de correspondance, critère d'optimalité). Donner la solution du problème du point de vue de l'utilisatrice (quels sont les itinéraires possibles, lesquels sont optimaux et pourquoi). Il est possible d'utiliser le même exemple que pour la Version 1 ou le modifier si pertinent.



Dans cet exemple, l'utilisateur n'est intéressé que par un seul critère et est capable de prendre n'importe quel moyen de transport. Il cherche à se rendre de la ville 1 à la ville 2. Sachant qu'il y a une correspondance à la ville 3 donnant un malus de 3, le meilleur chemin est 1 -> 3 -> 4 (6). Ce chemin permet d'optimiser le trajet, peu importe le moyen de transport.Le deuxième meilleur chemin implique de faire des allez retour entre 1 et 3, ce qui donnerait : 1 -> 3 -> 1 -> 3 -> 4 (8). L'autre chemin passant par la ville 2 n'a pas de malus de correspondance, mais il est le plus long (poids de 9) parmi les chemins possibles (sans compter les chemin qui ont une boucle).

Modèle pour l'exemple

Donner le graphe modélisant l'exemple ci-dessus. Donner la solution du problème (càd les meilleurs itinéraires) en tant que chemins dans le graphe.



Pour modéliser ce problème, nous avons décidé de représenter les correspondances par des arêtes entre deux sommets précis. Ainsi, chaque correspondance se trouve entre un sommet et un autre ayant le suffixe '_BIS'. Ici, l'utilisateur partant de ALPHA devra suivre le chemin suivant pour obtenir le plus court trajet :

```
ALPHA -> 1_TRAIN_AVION_BIS -> 3_AVION_TRAIN -> 3_AVION_TRAIN_BIS -> 4_TRAIN_TRAIN -> OMEGA
```

Ce chemin a un poids de 6 et correspond à l'exemple ci-dessus. Le second est : `ALPHA -> 1_TRAIN_AVION_BIS -> 3_AVION_AVION -> 3_AVION_AVION_BIS -> 1_AVION_AVION -> 1_AVION_AVION_BIS -> 3_AVION_TRAIN_BIS -> 4_TRAIN_TRAIN -> OMEGA`

Le dernier meilleur chemin est :

```
ALPHA -> 1_TRAIN_TRAIN_BIS -> 2_TRAIN_TRAIN -> 2_TRAIN_TRAIN_BIS -> 4_TRAIN_TRAIN -> OMEGA
```

Ce modèle permet de bien représenter les correspondances et d'optimiser le trajet en fonction des critères donnés.

À noter qu'ALPHA est directement lié aux '_BIS' pour éviter les coûts de correspondance de la ville de départ. OMEGA esquive également les '_BIS' pour les mêmes raisons. Donc, dans les chemins ci-dessus, la modalité d'arrivée du premier sommet que l'on atteint en partant d'ALPHA n'a aucune importance et pourrait donc être BUS, TRAIN, ou encore AVION. Le même principe s'applique pour OMEGA, mais concernant les modalités de départ des sommets le précédant.

Chaque nom de lieu suit ce format : NOM_ModArr_ModDep(_BIS), où ModArr est le moyen de transport permettant d'atteindre ce lieu et ModDep est le moyen de transport quittant ce lieu.

Modélisation pour la Version 2 dans le cas général

*Mêmes questions que pour la section correspondante de la Version 1, mais cette fois-ci les données d'entrée contiennent aussi des coûts de correspondance. Vous pouvez expliquer l'entièreté de la solution pour la Version 2, ou bien indiquer **clairement** les différences par rapport à la solution proposée pour la Version 1.*

D'abord, pour construire notre graphe, nous commençons par construire les correspondances. Pour cela, nous créons d'abord, pour chaque correspondance renseignée, un premier lieu nommé suivant le format expliqué ci-dessus, que nous lions à un autre sommet portant le même nom suivi du suffixe _BIS. Nous donnons à cette arête le poids correspondant au critère demandé. Le moyen de transport de cette arête n'a pas d'importance.

Après avoir créé toutes les correspondances renseignées, nous créons toutes les correspondances non renseignées avec tous les moyens de transport possibles en attribuant un poids nul.

Ensuite, nous créons les différentes routes possibles entre les villes. Il faut tout particulièrement faire attention à ce que les modalités de départ des villes de départ soient égales aux modalités d'arrivée des villes d'arrivée, sinon les coûts de correspondance seront incohérents.

Enfin, pour résoudre le problème, il ne reste plus qu'à lier le sommet ALPHA à tous les lieux de la ville de départ possédant le suffixe '_BIS'. Puis, lier le sommet OMEGA aux sommets de la ville de destination ne possédant pas ce suffixe. Ensuite, il faut lancer l'algorithme de Dijkstra en lui fournissant le graphe ainsi que la ville de départ.

Implémentation de la Version 2

Écrire une classe de test qui reprend l'exemple, définit toutes les données de la plateforme, construit le graphe et calcule la solution. Votre classe peut utiliser des assertions (test unitaire) ou bien afficher la solution. Donner ici le ***VoyageurTestReadMe****,* ***6edb7bbf481c3f304eb5a45ff5be5d48a7310751*** *et un [tests/graphes/VoyageurTestReadMe.java · 6edb7bbf481c3f304eb5a45ff5be5d48a7310751 · SAE2.01-2.02 / 2024 / B6 · GitLab (univ-lille.fr)]*