SAE S2.02 -- Rapport pour la ressource Graphes

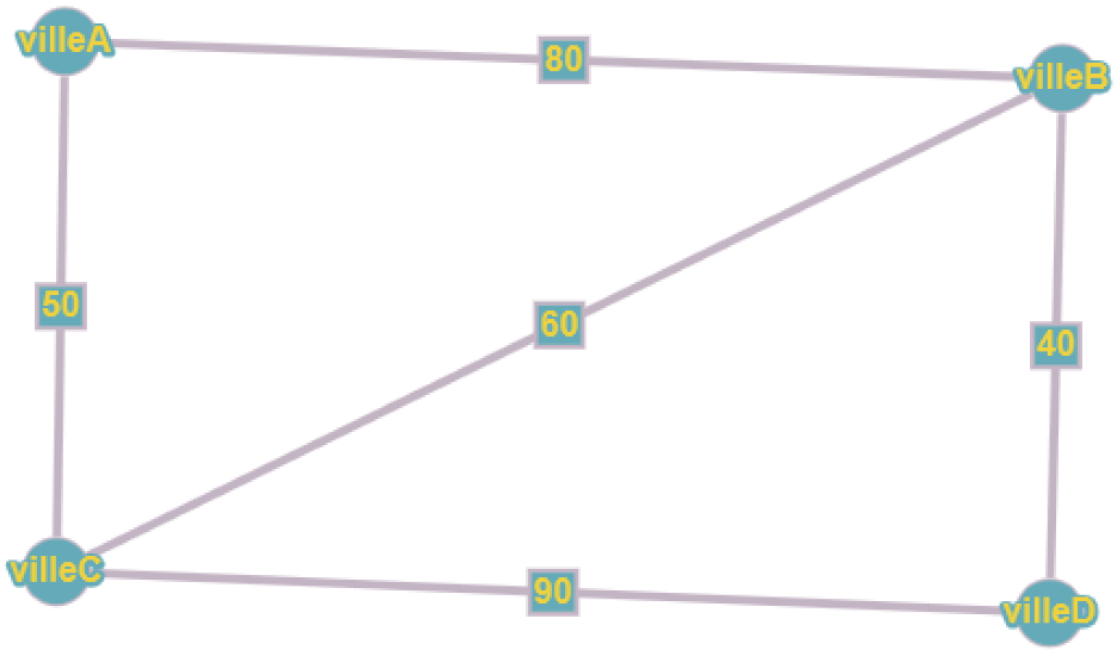
Fryson Adrien et Roget Benjamin, Groupe B6

Version 1 : un seul moyen de transport

Présentation d'un exemple

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.

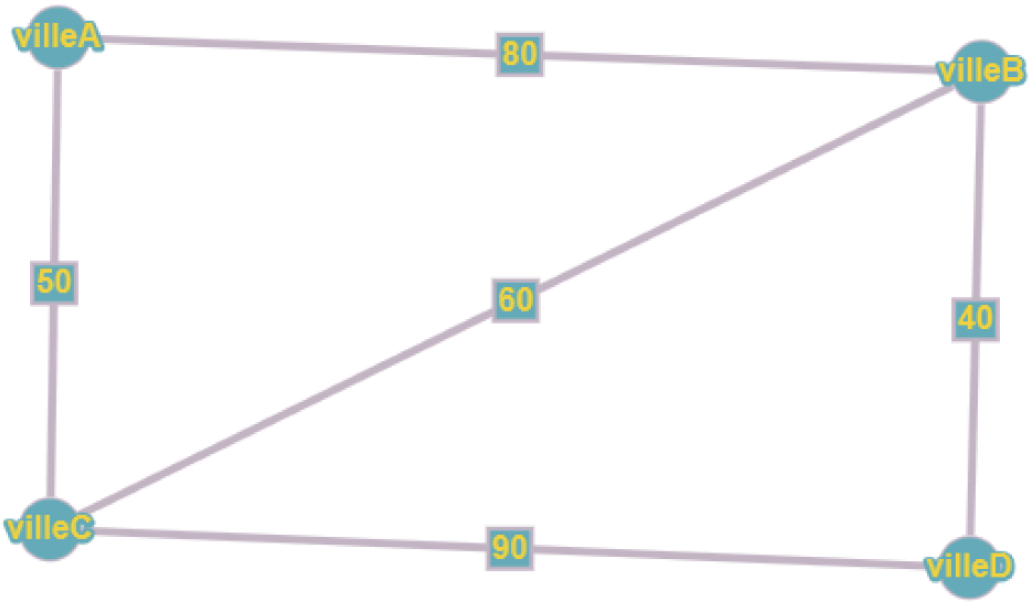
Les choix à sa disposition peuvent être illustrés comme suit :



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



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

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.

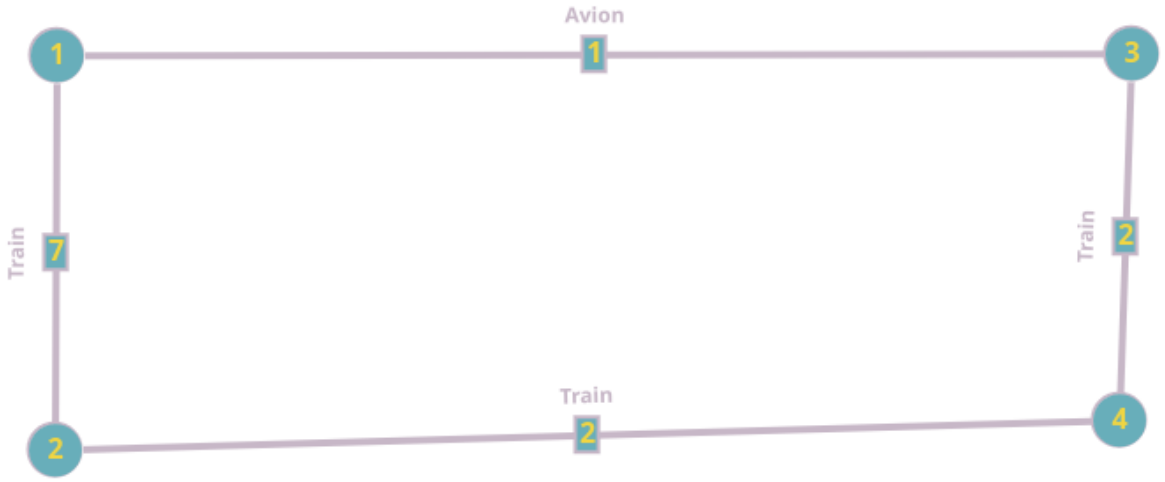
L'algorithme qui doit être utilisé pour résoudre ce problème est l'algorithme de Dijkstra. Il prend en paramètre un graphe avec uniquement des poids positifs et un sommet de départ.

Implémentation de la Version 1

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)]*

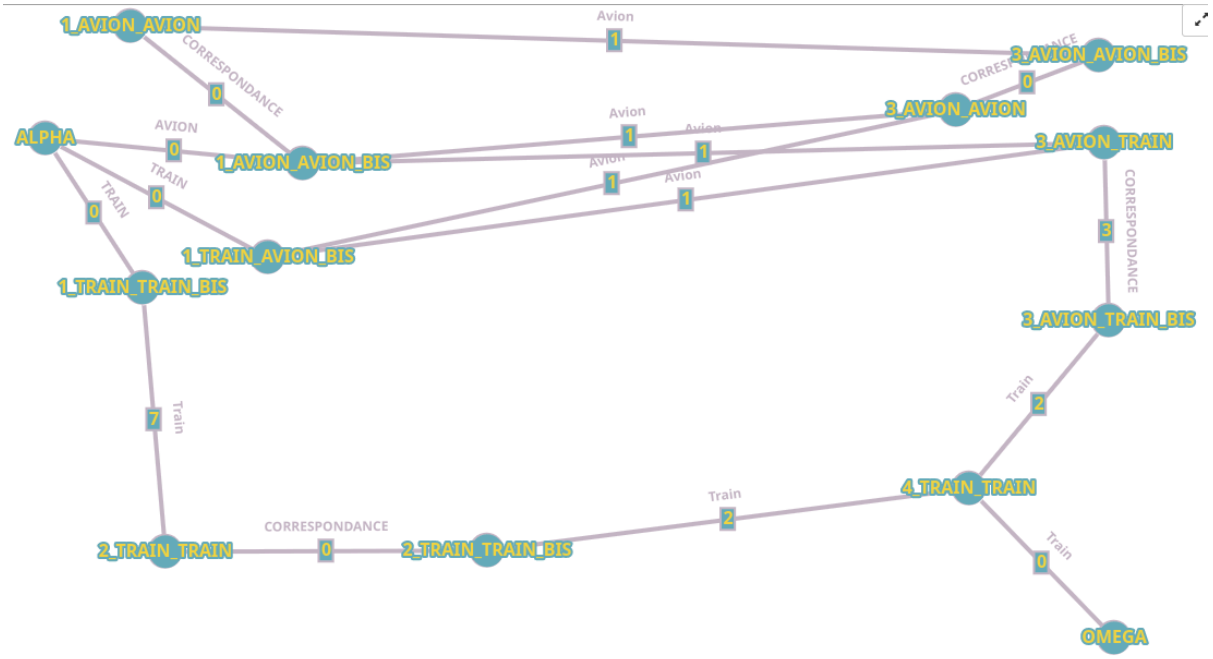
Version 2 : multimodalité et prise en compte des correspondances

Présentation d'un exemple



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 chemins qui ont une boucle).

Modèle pour l'exemple



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

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.

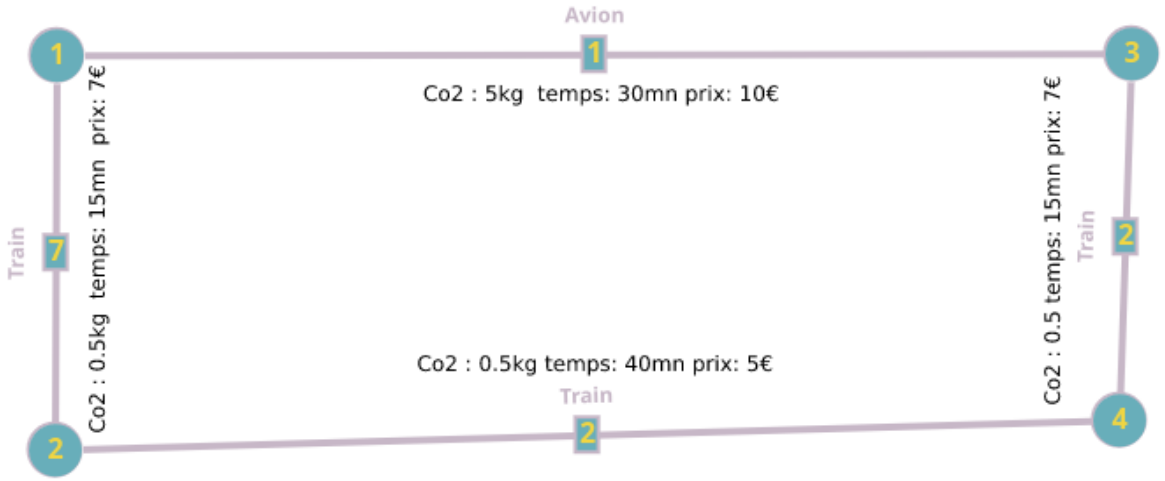
Enfin, pour résoudre le problème, il ne reste plus qu'à lier le sommet ALPHA à tous les lieux de la ville de départ possédant le suffixe '_BIS'. Puis, lier le sommet OMEGA aux sommets de la ville de destination ne possédant pas ce suffixe. Ensuite, il faut lancer l'algorithme de Dijkstra en lui fournissant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

Implémentation de la Version 2

***VoyageurTestReadMe**, **6edb7bbf481c3f304eb5a45ff5be5d48a7310751** et un [tests/graphes/VoyageurTestReadMe.java · 6edb7bbf481c3f304eb5a45ff5be5d48a7310751 · SAE2.01-2.02 / 2024 / B6 · GitLab (univ-lille.fr)]*

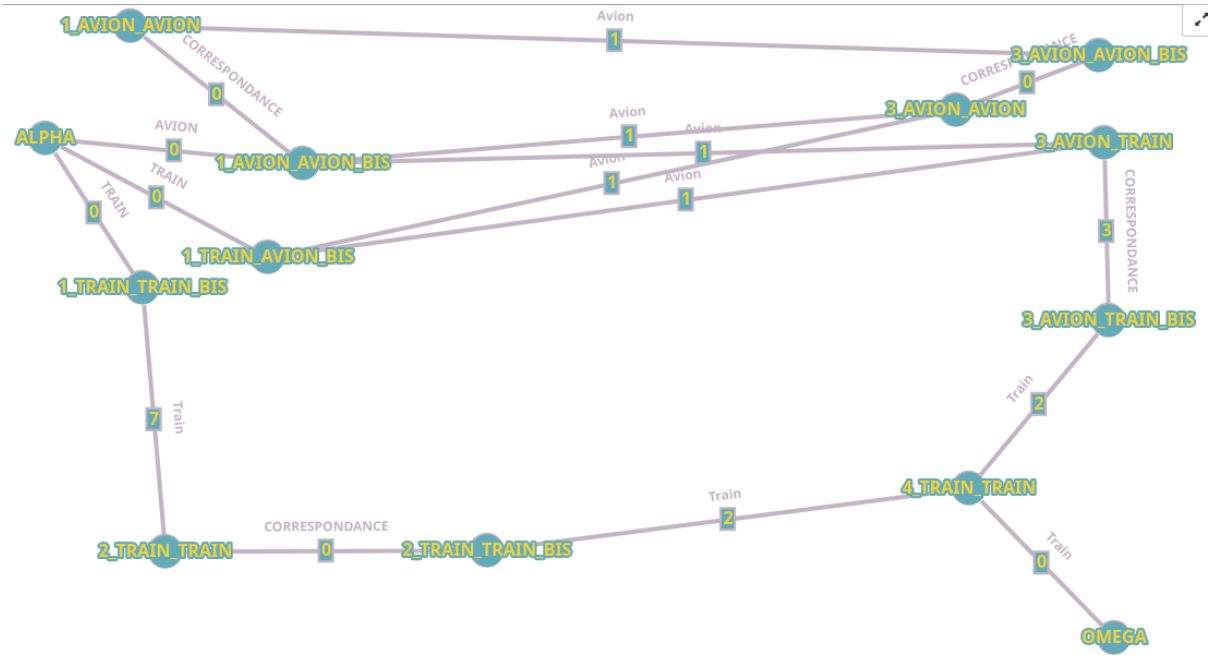
Version 3 : optimisation multi-critères

Présentation d'un exemple



Dans cet exemple, l'utilisateur est intéressé par plusieurs critères et est capable de prendre n'importe quel moyen de transport. Il cherche à se rendre de la ville 1 à la ville 4. Sachant qu'il y a une correspondance à la ville 3 donnant un malus de 3 dans tous les critères. Admettons que l'utilisateur ait un ordre de préférence : (CO2 > prix > temps).

Alors, une bonne solution serait le chemin 1 -> 2 -> 4 car il permet de minimiser à la fois le premier et le second critère (1 kg de CO2, 55 min et 12 €). L'autre chemin 1 -> 3 -> 4 correspond beaucoup moins bien aux critères de l'utilisateur (8,5 kg de CO2, 48 min et 20 €). Si l'utilisateur avait donné une plus grande importance au temps de trajet, alors sans doute que le second chemin aurait été le meilleur.



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 en optimisant les critères suivants cet ordre (Co2 > prix > temps):

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

Ce trajet émet 1 kg de Co2, prend 55 min et coûte 12€.

Le dernier meilleur chemin est :

ALPHA -> 1_TRAIN_AVION_BIS -> 3_AVION_TRAIN -> 3_AVION_TRAIN_BIS -> 4_TRAIN_TRAIN -> OMEGA
avec une émission de 8.5 kg de Co2, 48 min et coûte 17€.

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.

Pour hiérarchiser les chemins entre eux du meilleur selon les critères donnés au pire, nous avons opté pour un système de score.

Pour calculer le score, nous suivons cette formule : c = (p1c1) + (p2c2) + (p3c3) où c est le score attribué au chemin, p1, p2 et p3 sont les poids pondérés du chemin suivant les différents critères, et c1, c2 et c3 sont les coefficients que l'on donne à chaque critère avec c1 + c2 + c3 = 1.

Pour calculer les p, on suit cette formule : p = v/pmax, où p est le poids pondéré, v est la valeur du poids, et pmax est une constante représentant le poids maximum. Cette valeur est arbitraire et doit être choisie judicieusement pour pouvoir avoir un classement cohérent.

Si l'on reprend notre exemple, disons que le temps maximum est de 960 min, le prix maximum 750 € et les émissions maximales de 450 kg.

Pour les coefficients, nous décidons de leur donner ces valeurs : c1 = 0.6, c2 = 0.35 et c3 = 0.05.

Calculons le score du premier chemin de l'exemple :

Nous avons, dans l'ordre d'importance des critères pour le CO2, p1 = 1 / 450, p2 = 12 / 750 et p3 = 55 / 960.
Nous avons alors C = 0.6*(1/450) + 0.35*(12/750) + 0.05*(55/960) = 0.00979.

Le score du meilleur chemin est donc de 0.00979.
Le score du deuxième chemin est de 0,0231.
Nous avons donc bien le score du premier chemin inférieur au second, ce qui nous permet d'arriver aux mêmes conclusions que dans l'exemple.

IhmInterfaceImplTest, **17/06/2024 cb5e9e081c92e64f76ff4bbba25974003689e91c** et un **https://gitlab.univ-lille.fr/sae2.01-2.02/2024/B6/-/blob/cb5e9e081c92e64f76ff4bbba25974003689e91c/tests/graphes/IhmInterfaceImplTest.java**