Vous pouvez télécharger le présent fichier au format .md depuis la page Moodle de la SAÉ à l'aide d'un clic droit sur le lien, "Enregistrer la cible du lien" ou équivalent.

Le rapport à rendre doit être au format pdf ou html (éventuellement format md lisible sur gitlab) et doit suivre le plan donné ici. Il doit se trouver dans un répertoire `graphes` à la racine de votre dépôt git. Le premier rendu concerne uniquement la Version 1, le second rendu concerne la Version 2.

Pour chaque partie du plan nous indiquons *en italique* ce que cette partie doit contenir au minimum.

Le rapport n'est pas un DS où on se contente de répondre aux questions. Dans le rapport **vous devez écrire des phrases et ajouter toutes les explications** qui vous semblent nécessaires pour le rendre compréhensible et agréable à lire, comme tout rapport se doit d'être.

SAE S2.02 – Rapport pour la ressource Graphes

Noms des auteurs, groupe

Version 1

Sera évaluée à partir du tag git `Graphes-v1`

Étude d'un premier exemple

Énumérer tous les appariements acceptables (c'est à dire qui asocient des adolescents compatibles) pour les données de l'Exemple 1, en supposant que les français rendent visite aux italiens.

Justifier pourquoi l'appariement optimal est Bellatrix–Xolag, Adonia–Zander, et Callista–Yak; cette explication ne doit pas parler de graphes, mais se baser uniquement sur les données du problème.

Modélisation de l'exemple

Donner un graphe qui modélise l'Exemple 1, plus précisément, la matrice d'adjacence de ce graphe. Expliquez comment vous avez choisi le poids pour chacune des arêtes.

Modélisation pour la Version 1

Décrire une modélisation générale pour la Version 1. C'est à dire, donner une formule ou une description précise qui décrit comment, étant donné un adolescent hôte et un adolescent visiteur, on détermine le poids de l'arête entre ces deux adolescents en fonction des critères considérés dans la Version 1.

Implémentation de la Version 1

Cette partie du travail sera évaluée à partir du code. Implémenter la fonction weight de la classe AffectationUtil en suivant la modélisation proposée. Puis, implémenter une classe TestAffectationVersion1 qui montre que votre implémentation est correcte. La classe de test doit contenir au moins une méthode de test comme ceci: - créer les adolescents de l'Exemple 1 - construire le graphe modèle pour ces adolescents; le graphe sera de type fr.ulille.but.GrahpeNonOrienteValue - calculer l'affectation optimale en utilisant la classe fr.ulille.but.CalculAffectation - écrire des assertions (assertEquals . . .) qui vérifient que le résultat de l'affectation calculé à l'étape précédente est bien celui attendu

Si vous n'êtes pas à l'aise avec les tests unitaires, votre classe TestAffectationVersion1 peut contenir une méthode main à la palce de la méthode de test, dans ce cas vous afficherez dans le terminal l'appariement résultat.

Exemple de vérification de l'incompatibilité

*Cet exemple va mettre au défi votre modèle vis à vis de la prise en compte de l'incompatibilité entre adolescents

Récupérez sur Moodle le fichier de données `compatibilityVsHobbies.csv`. Expliquez quelle est sa particularité de cet exemple. Écrire la méthode de test qui test qui change cet exemple, construit le graphe modèle, calcule l'affectation, et finalement vérifie qu'aucune paire d'adolescents non compatibles n'a été construite par l'algorithme.*

Version 2

Sera évaluée à partir du tag git `Graphes-v2`

Exemple minimal pour la gestion de l'historique

Présenter un exemple minimal qui est pertinent pour tester l'historique. L'exemple contiendra: - huit adolescents de deux pays différents tels que - certains des adolescents expriment des préférences d'historique (critère HISTORY). Toutes les valeurs possibles pour ce critère doivent être présentes - aucun des adolescents n'est allergique aux animaux en aucun n'a exprimé de passe-temps, ainsi pour l'instant on peut se concentrer uniquement sur la gestion de l'historique - un historique, c'est à dire une collection de paires d'adolescents qui étaient correspondants l'année passée. Ces paires doivent permettre d'illustrer les différents cas de figure qui peuvent se présenter par rapport aux contraintes d'historique et les huit adolescents

Puis, donner un appariement optimal qui tient compte des données d'historique, et expliquer pourquoi il est optimal. L'explication ne doit pas parler des graphes, mais uniquement des adolescents et les critères exprimés.

Deuxième exemple pour la gestion d'historique

Modifiez l'exemple précédent en ajoutant des préférences liées aux passe-temps. Donnez l'appariement que vous considérez optimal dans ce cas. En particulier, expliquez comment vous comptez combiner une éventuelle affinité liée à l'historique avec l'affinité liée aux passe-temps. Rappelons que l'historique peut compter comme une contrainte rédhibitoire ou comme une préférence, voir le sujet pour plus de précisions.

Donner l'appariement que vous considérez optimal dans ce deuxième exemple, toujours sans parler de graphes.

Modélisation pour les exemples

Pour chacun des deux exemples précédents, donnez un graphe (donné par sa matrice d'adjacence) tel que l'affectation minimale dans ce graphe correspond à l'appariement optimal identifié plus haut. Expliquez comment vous avez choisi le poids pour chacune des arêtes.

Modélisation pour l'historique de la Version 2

Décrire une modélisation générale pour la Version 1. C'est à dire, donner une formule ou une description précise qui décrit comment, étant donné un adolescent hôte et un adolescent visiteur, on détermine le poids de l'arête entre ces deux adolescents en fonction des critères considérés dans la Version 1. Décrire également comment vous construisez le graphe modèle à partir des données en entrée.

Implémentation de l'historique de la Version 2

Quelles fonctions de votre code avez-vous modifié pour prendre en compte le critère historique ? Donnez ici les noms des méthodes (et leur classe), à quoi elles servent, et quelles modifications vous avez apportées. Essayez d'être synthétique.

Test pour l'historique de la Version 2

Créer la classe de TestAffectationVersion2 qui contiendra deux méthodes de test, une pour chacun des exemples. Chacune de ces méthodes doit avoir la même structure que pour TestAffectationVersion1, c'est à dire créer les données d'entrée (adolescents, historique), créer le graphe, calculer l'affectation, et tester que le résultat est comme attendu.

Prendre en compte les autres préférences

Pour chacun des autres critères d'affinité que vous décidez de prendre en compte, décrire comment vous changez la fonction weight de la classe AffectationUtil.

L'incompatibilité en tant que malus

Proposer une formule ou une description précise qui explique comment calculer le poids d'une arête en considérant les incompatibilités comme des malus et les critères satisfaits comme des bonus. Implémenter cette formule dans une seconde méthode appelée `weightAdvanced`*, ceci pour éviter de casser votre code. Puis, écrire une méthode de test qui permet d'illustrer le calcul d'affectation basé sur* `weightAdvanced`*. Vous pouvez égalmente tester l'affectation en utilisant le fichier de données* `incompatibilityVsBonus.csv`*.*