SAE S2.02 -- Rapport pour la ressource Graphes

Groupe E3 : FOURMAINTRAUX Camille, DESRUMAUX julien,

Version 2

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

Étude d'un premier exemple

Dans le premier cas, nous avons que six étudiants échantillon : Tableau markdom

Noms	Prénoms	Pays	Naissance	Genre	hobbies	conditions
A	Adonia	FRANCE	2004-07-15	female	sports,technology	
B	Beelatrix	FRANCE	2004-07-15	female	culture,science	allergique
C	Adonia	FRANCE	2004-07-15	female	science,reading	
X	Xolag	ITALY	2004-07-15	male	culture,technology	
Y	Yak	ITALY	2004-07-15	male	science,reading	aAnimaux
Z	Zander	ITALY	2004-07-15	male	technology	

On peut déterminer facilement les meilleures combinaisons et les pires.

Déjà, B-Y est la pire, car ils ne sont même pas compatibles. Belleatrix est allergique aux animaux et Yak en possède. C-Y sont les deux seuls à aimer la lecture, B-X sont les deux seuls à aimer la culture, il reste A-Z et A-X, dont les trois aiment la technologie. D'autres combinaisons sont peut-être possibles. On obtient le tableau suivant :

Légende : OO = Très bien(Au moins 1 point commun), O = Bien(1 point commun), N = Mauvais(Pas de poins commun), NN = Très mauvais(Imcompatible).

sommet	A	B	C
X	O	OO	N
Y	N	NN	OO
Z	O	N	N

La combinaison optimale est donc A-Z,B-X,C-Y, un appariement AX est possible, mais au détriment d'un meilleur appariments B-X qui devendrait forcément B-Z.

Modélisation de l'exemple

GRAPHE-V1: [Arête(A, X, 60.0), Arête(A, Y, 70.0), Arête(A, Z, 60.0), Arête(B, X, 60.0), Arête(B, Y, 60.0), Arête(B, Z, 70.0), Arête(C, X, 70.0), Arête(C, Y, 50.0), Arête(C, Z, 70.0)]

sommet	A	B	C
X	20	20	30
Y	30	100	10
Z	20	30	30

sommet	A	B	C
X	O	O	N
Y	N	NN	OO
Z	O	N	N

Attention, plus le score de compatibilité est grand, plus le poids est petit. Nous avons mis à jour le tableau en prenant maintenant en compte les résultats du calcul. Nous avons obtenu les mêmes résultats à une différence près. Le poids est calculé en fonction du nombre de contraintes respectées et du nombre de points

communs partagés. Un point commun partagé vaut 10 points. C et Y partagent 2 points communs, mais B et Z n'en partagent aucun. Il y a donc une différence de 20 points (2 * 10) entre eux.

Modélisation pour la Version 1

Cette version de la méthode "weight" ne se base que sur la compatibilité et le nombre de hobbies en commun. Par défaut, le poids est fixé au plus grand : 96.0. Si les deux adolescents passés en paramètres ne sont pas compatibles, alors la fonction le renvoie directement. Sinon, elle soustrait 66 à ce poids et poursuit avec le nombre d'hobbies en commun. Cela permet de différencier les étudiants incompatibles de ceux ayant un mauvais score simplement parce qu'ils n'ont rien en commun.

Ensuite, la fonction fait appel à une autre fonction qui calcule le nombre d'hobbies en commun et diminue le poids de 10 fois le nombre d'hobbies en commun. La fonction se termine en renvoyant le poids.

Cette méthode détermine si deux adolescents sont compatibles en faisant appel à une méthode de Teenager : "compatibleWithGuest", qui renvoie un booléen en fonction de si deux teenagers n'ont pas de critères les rendant incompatibles, comme l'hôte ayant des animaux et l'invité ayant une allergie aux animaux.

Cette fonction est appelée à la création de chaque arête du graphe pour déterminer le poids de celle-ci et permettre aux calculs de l'utiliser.

Implémentation de la Version 1

La classe TestAffectationUtil et la fonction weight(renommée weight V1 depuis l'apparition de la version 2) ont étés implémantées. La classe de test "TestAffectationUtil" teste toute les fonctions d'AffectationUtil utilisé dans la version 1 comme weightV1, initGraphV1 ou creerEchantillonAdoslescents.

Exemple de vérification de l'incompatibilité

Cet exemple est particulier car les données ne sont pas valides et peuvent conduire à l'association d'étudiants qui ne sont pas valides. J'ai ajouté le test "testIncompatibilityVsHobbies" qui charge le fichier CSV, teste les données en créant un graphe et un calcul d'affectation, puis vérifie que toutes les affectations ont été faites avec des étudiants compatibles les uns avec les autres. Le test indique un résultat positif.

Version 2

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

Exemple minimal pour la gestion de l'historique

J'ai créé un échantillon de 8 étudiants, dont les hobbies et les conditions sont vides pour ne pas interférer avec les tests sur l'historique. Je les ai d'abord créés et ai effectué le calcul de l'affectation sans leur attribuer d'historique, afin d'avoir une entrée dans l'historique sans causer de bug. Ensuite, je leur ai ajouté leur condition d'historique et j'ai refait une affectation pour voir si l'historique allait être pris en compte.

La première affectation a donné : AW, BX, CY, DZ. Remarque : Les binômes sont présentés dans l'ordre alphabétique, car il n'y avait aucun critère permettant de différencier des binômes particuliers.

La première affectation à donnée : AW,BX,CY,DZ. Remarque : C'est dans l'ordre alphabétique, car il n'y avait aucun critère pour différencier des binômes particulier.

Noms	Prénoms	Pays	Naissance	Genre	hobbies	conditions	historique
A	Adonia	FRANCE	2004-07-15	female			same
B	Beelatrix	FRANCE	2004-07-15	female			other
C	Adonia	FRANCE	2004-07-15	female			other
D	Denise	FRANCE	2004-07-15	female			
W	Waluigi	ITALY	2004-07-15	male			same
X	Xolag	ITALY	2004-07-15	male			other
Y	Yak	ITALY	2004-07-15	male			same
Z	Zander	ITALY	2004-07-15	male			

On peut déjà déterminer l'une des combinaisons possibles que devrait renvoyer le nouveau calcul. En effet, A et W veulent tous les deux "same", donc ils devront reformer le même binôme. Par contre, B et X veulent tous les deux "other", donc ils ne devront absolument pas reformer le même binôme. À partir de là, cela se complique un peu car C veut "other" et Y veut "same". Dans ce cas, il y a priorité à "other" et on essaiera de ne pas les mettre ensemble. De même pour D et Z qui n'ont rien précisé. Dans ce cas, c'est neutre et cela n'ajoute ni ne diminue aucun poids. Dans le cas où il y a un "non-précisé" et un "précisé" (comme "other" ou "same"), alors le critère précisé a priorité.

Récapitulons :

sommet	A	B	C	D
W	OO	NN	NN	NN
X	NN	NN	O	O

sommet	A	B	C	D
Z	NN	O	O	O

Le manque de critères fait que plusieurs combinaisons optimales sont possibles, précisement trois, tel que : BY, CX, DZ ainsi que BY, CZ, DX et enfin BZ, CX, DY.

Deuxième exemple pour la gestion d'historique

J'ai modifié l'échantillon de 8 étudiants en y ajoutant des hobbies. De la même manière, je les ai d'abord créés et j'ai calculé l'affectation en laissant les historiques vides, afin d'avoir une entrée dans l'historique sans causer de problèmes. Ensuite, je leur ai ajouté leur condition d'historique et j'ai refait une affectation pour voir si l'historique allait être pris en compte et comment les hobbies allaient influencer le résultat.

Noms	Prénoms	Pays	Naissance	Genre	hobbies	conditions	historique
A	Adonia	FRANCE	2004-07-15	female	science,movies,games		same
B	Beelatrix	FRANCE	2004-07-15	female	technology		other
C	Adonia	FRANCE	2004-07-15	female	culture		other
D	Denise	FRANCE	2004-07-15	female	movies		
W	Waluigi	ITALY	2004-07-15	male	science,games		same
X	Xolag	ITALY	2004-07-15	male	technology,games		other
Y	Yak	ITALY	2004-07-15	male	movies		same
Z	Zander	ITALY	2004-07-15	male	culture,science		

La première affectation (qui ne se base pas sur l'historique) devrait donner : AW, BX, CZ, DY, car ils ont le plus d'hobbies en commun.

Si à partir de là on sauvegarde l'historique et on recalcule l'affectation en prenant en compte celui-ci, on peut également déterminer les combinaisons possibles.

En effet, A et W veulent tous les deux "same", donc ils devront reformer le même binôme. Par contre, B et X veulent tous les deux "other", donc ils ne devront absolument pas reformer le même binôme. C veut "other", mais Z n'a rien précisé. Dans ce cas, on appliquera "other" et on essayera de ne pas les remettre ensemble. D n'a rien précisé, mais Y veut "same". Dans ce cas, on appliquera "same" et on essaiera de les remettre ensemble.

Récapitulons :

sommet	A	B	C	D
W	OO	NN	NN	NN
X	NN	NN	O	NN
Y	NN	NN	NN	OO
Z	NN	O	O	NN

Si nous partons du principe que AW et DY doivent être ensemble, alors la seule combinaison optimale restante est AW, BZ, CX, DY.

Modélisation pour les exemples

GRAPHE-SEULEMENT_HISTORIQUE: [Sommets: {A, B, C, D, W, X, Y, Z}, Arêtes et poids: (A, W):170.0, (A, X):270.0, (A, Y):290.0, (A, Z):290.0, (B, W):270.0, (B, X):320.0, (B, Y):270.0, (B, Z):250.0, (C, W):270.0, (C, X):250.0, (C, Y):320.0, (C, Z):250.0, (D, W):290.0, (D, X):250.0, (D, Y):290.0, (D, Z):270.0]

sommet	A	B	C	D
W	170	270	270	290
X	270	320	250	250
Y	290	270	320	290
Z	290	250	250	270

sommet	A	B	C	D
W	OO	N	N	N

sommet	A	B	C	D
X	N	NN	O	O
Y	N	N	NN	N
Z	N	O	O	N

GRAPHE-HISTORIQUE&HOBBIES: [Sommets: {A, B, C, D, W, X, Y, Z}, Arêtes et poids: (A, W):150.0, (A, X):260.0, (A, Y):280.0, (A, Z):280.0, (B, W):270.0, (B, X):310.0, (B, Y):270.0, (B, Z):250.0, (C, W):270.0, (C, X):250.0, (C, Y):270.0, (C, Z):310.0, (D, W):290.0, (D, X):250.0, (D, Y):245.0, (D, Z):270.0]

sommet	A	B	C	D
W	150	270	270	290
X	260	310	250	250
Y	280	270	270	245
Z	280	250	310	270

sommet	A	B	C	D
W	OO	N	N	N
X	N	NN	O	O
Y	N	N	N	O
Z	N	O	NN	N

Nous avons mis à jour le tableau en prenant maintenant en compte les hobbies en plus de l'historique.

Le poids est défini à 300 et est diminué (ou augmenté) en fonction de l'historique et des hobbies. Plus le score de compatibilité augmente, plus le poids d'une arête diminue. Le fait que AW et DY veuillent être ensemble à nouveau a grandement influencé leur score, tandis que BX et CZ ne voulant plus être ensemble a fait augmenter leur score.

On remarque clairement que les hobbies ont influencé les résultats, car certains poids ont diminué grâce à une meilleure affinité entre les étudiants. Au début, cela a créé de nouveaux binômes plutôt que simplement des binômes par ordre alphabétique.

Implémentation de l'historique de la Version 2

Dans le but d'implémenter l'historique dans la fonction weight, j'ai ajouté deux fonctions : History.deserialization et History.rechercheSejours. D'abord, la méthode vérifie si un historique existe et si oui, elle utilise deserialization pour récupérer cet historique et créer un objet History avec, qui sera utilisé pour les prochains calculs. Un objet History contient une liste d'objets Séjours qui eux-mêmes contiennent deux étudiants. On peut retrouver un séjour en particulier dans cette liste grâce à la méthode rechercheSejour qui prend 2 étudiants et la liste de Séjours. La méthode weight calculant un poids entre deux étudiants, on vérifie si l'historique désérialisé est nul ou vide et si ce n'est pas le cas, on recherche si un séjour avec ces deux étudiants existe. Si la méthode trouve que ces deux étudiants étaient en groupe l'année dernière, alors la fonction regarde leur condition d'historique et modifie le score en fonction. S'ils veulent rester en binôme, on applique un gros bonus, et à l'inverse, un gros malus, et si c'est plus indécis, on privilégie la séparation.

Test pour l'historique de la Version 2

Pour tester cet historique, j'ai créé la classe TestV2AffectationUtil qui crée les données d'entrée (étudiants, historique), crée le graphe, calcule l'affectation, et teste que le résultat est comme attendu, et ce pour les deux tests qu'elle contient. Le premier ne teste que l'historique seul, et le deuxième teste l'historique et les hobbies en même temps.

Prendre en compte les autres préférences

Les autres préférences sont calculées de la même manière : un bonus en cas d'affinité et un malus dans le cas contraire. Un bonus est appliqué pour chaque hobby en commun avec un autre étudiant, ce qui fait que deux étudiants partageant plusieurs hobbies auront un meilleur score que deux étudiants partageant un seul ou aucun hobby. Un bonus est appliqué si les étudiants n'ont pas une grande différence d'âge (-1,5 an), et un autre si leur préférence de genre est bien respectée (une partie du bonus pour l'invité et une autre pour l'hôte).

L'incompatibilité en tant que malus

Nous n'avons pas créé de fonction weightAdvanced, j'ai plutôt modifié la fonction weight d'origine.

Les autres préférences sont calculées en tant que bonus, mais dans le cas des contraintes telles que les allergies et la nourriture, la fonction ne fonctionne plus en tant que bonus, mais en tant que malus. Un malus est appliqué si un hôte possède un animal et que le guest y est allergique. Pour la nourriture, plus l'hôte ne propose pas le type de nourriture demandé par l'invité, plus le malus sera grand. Par exemple, si un invité demande un régime végétarien sans noix ni gluten, un premier hôte ne proposant qu'un régime végétarien aura un plus grand malus qu'un second hôte proposant un régime végétarien sans noix, même si le second hôte ne propose pas tous les types de régimes demandés. De même pour l'historique, si deux étudiants anciennement en binôme demandent à être séparés pour l'année prochaine, un gros malus sera appliqué pour empêcher qu'ils soient à nouveau ensemble, bien que cela reste techniquement possible, quoique improbable.