Partie Algorithmique

Application de création de groupes de TD et de TP

La partie algorithmique du projet est faite en Java et interagit avec l'application que vous programmez dans la partie « Qualité de développement ». Les algorithmes que vous écrivez doivent pouvoir être appelés depuis l'application, éventuellement avec des choix de paramètres, et leurs résultats (les groupes constitués) affichés dans l'application.

Votre objectif principal est de générer des groupes qui respectent certaines contraintes (nombre d'étudiants par groupe, respect des covoiturages, etc.) et en tentant d'optimiser certains critères (moyennes équilibrées entre les groupes, répartition des différents bacs, etc.). PVous utiliserez les méthodes algorithmiques vues en cours (algorithmes gloutons, algorithmes de force brute) avec éventuellement des améliorations ou de approches créatives.

Chaque membre du groupe choisit un **mode de création de groupe** (correspondant à un scenario réaliste) : un ensemble de contraintes à faire respecter obligatoirement et des critères à optimiser au mieux. Comme il y aura plusieurs critères, on déterminera un score à minimiser / maximiser qui prend en compte les différents critères.

- Exemple : on cherche à faire des groupes d'apprentis.
  Contraintes : entre 16 et 18 étudiants, pas moins de 4 filles par groupe, covoiturages respectés.
  Critère d'optimisation : pour équilibrer au mieux les moyennes et le nombre d'étudiants sans entreprise entre les groupes, on va chercher à minimiser le score M+N, où M = plus grand écart de moyenne entre deux groupes, et N = nombre max d'étudiants sans entreprise dans un groupe.

Chaque étudiant propose, pour son mode de création, au moins un algorithme de force brute et deux algorithmes gloutons. Même si certaines fonctions sont partagées entre plusieurs modes de création (ce qui est bien !), chaque membre du groupe doit personnellement écrire au moins deux algorithmes gloutons.

Le code produit doit fonctionner ensemble et avec l'application ; en particulier, on utiliser les mêmes objets / classes et on doit pouvoir les appeler via l'application sur les données contenues dans la base.

De plus, vous devez écrire un programme qui teste les différents algorithmes gloutons sur des jeux de tests générés au hasard et indique, pour chaque critère, lequel est le plus efficace en moyenne.

Vous utiliserez la classe List ou Arraylist au moins à un endroit.

Votre code doit respecter les bonnes pratiques que vous avez apprises. Il renvoie un message d'erreur clair si l'entrée ne correspond pas à vos attentes (aucun étudiant, pas de solution valable, etc.). Chaque fonction a un commentaire court qui indique ses arguments, la sortie

fournie, ce qu'elle fait et l'auteur de la fonction. Les noms de variables et fichiers sont clairs et cohérents, pas de code dupliqué inutilement ou de code inutilisé, les niveaux de visibilité des fonctions (public ou private) sont bien choisis, etc.

Rendu : Collectivement, tout le code java que vous avez fait dans un dossier de groupe ainsi qu'un document readme qui explique clairement chaque mode de création de groupe et son responsable, donne les résultats des tests aléatoires, et explique les algorithmes qui vous semblent compliqués / les améliorations que vous avez trouvées, etc.

Notation : Vous êtes notés sur vos réalisations, avec une partie de la note qui porte sur : l'intégration (fonctionne avec le reste du projet) ; la robustesse (gère les erreurs) ; la clarté et l'organisation du code ; la documentation.
Les 2/3 de la note est collective, avec potentiellement une différence en fonction de l'implication. Vous êtes individuellement responsable, et noté individuellement, sur le bon fonctionnement de votre mode de création (même s'il y a du code partagé entre plusieurs modes écrit par d'autres).
Les absences non justifiées et les séances passées sur d'autres tâches que la partie algo du projet sont prises en compte quand nous jugeons de votre implication.

Le travail supplémentaire pertinent (algorithmique) est récompensé. Faire exactement le contenu demandé amène à une note entre 13 et 16 (suivant la difficulté des algos implémentés / du mode de décision) et les points supplémentaires viennent de : optimisations ou améliorations de vos méthodes, autres algorithmes ou approches, deuxième mode de création, etc. N'hésitez pas à nous parler de vos idées et mettez-les en valeur dans la documentation et/ou à la soutenance.

Soutenance : Vous devez pouvoir faire rapidement une démonstration de votre code utilisé depuis l'application Java et / ou depuis la console, en nous expliquant tout ce que vous avez implémenté, le tout en 10 minutes maximum. La soutenance nous sert à récolter des informations ; ce n'est pas un test d'éloquence. Vous devez être capables d'expliquer votre code et de répondre à des questions, chaque mode de décision étant expliqué par son responsable.