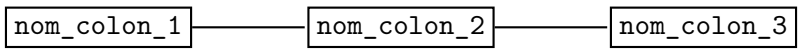
Projet : Partage de biens d'une colonie spatiale (partie 2)

I Entrées-Sorties

Lors de la première phase du projet, vous avez dû construire à la main votre colonie (nombres de colons, préférences des colons et liens entre les colons). Cela peut s'avérer fastidieux surtout avec un grand nombre de colons. La première modification majeure va consister à simplifier cette phase là. Nous souhaitons maintenant que le programme puisse lire dans un fichier texte la description de la colonie (c'est-à-dire le graphe des relations, et les préférences de chaque colons). Un tel fichier devra respecter le format suivant :

```
colon(nom_colon_1).
colon(nom_colon_2).
colon(nom_colon_3).
ressource(nom_ressource_1).
ressource(nom_ressource_2).
ressource(nom_ressource_3).
deteste(nom_colon_1,nom_colon_2).
deteste(nom_colon_2,nom_colon_3).
preferences(nom_colon_1,nom_ressource_1,nom_ressource_2,nom_ressource_3).
preferences(nom_colon_2,nom_ressource_2,nom_ressource_1,nom_ressource_3).
preferences(nom_colon_3,nom_ressource_3,nom_ressource_1,nom_ressource_2).
```

Pour être valide, le fichier doit impérativement respecter l'ordre de définitions des éléments : d'abord définir les colons, puis les ressources, puis le liens entre colons et enfin les préférences des colons. Les colons et les ressources peuvent avoir un nom quelconque mais, pour simplifier les choses, nous supposerons qu'ils seront uniquement constitués de caractères alpha-numériques (des lettres minuscules ou majuscules et des chiffres). Par exemple, le fichier précédemment décrit correspond à la colonie et aux préférences illustrés en Figure 1.




```
nom_colon_1: nom_ressource_1 > nom_ressource_2 > nom_ressource_3
nom_colon_2: nom_ressource_2 > nom_ressource_1 > nom_ressource_3
nom_colon_3: nom_ressource_3 > nom_ressource_1 > nom_ressource_2
```

FIGURE 1 – La colonie et les préférences correspondant au fichier texte

II Automatisation de la recherche de solution

Rechercher une solution manuellement s'avère également être un exercice fastidieux. L'objectif est donc d'utiliser un algorithme permettant d'automatiser la recherche d'une meilleure solution (i.e., trouver une affection ayant un nombre de colons jaloux plus petit). Nous présentons ici un algorithme naïf pour une résolution approximative du problème. On suppose ici que la colonie est constitué du graphe et des préférences (cela ne veut pas dire que c'est la meilleure modélisation possible, ni qu'il faut obligatoirement que votre modélisation soit similaire). On suppose aussi que la solution est une liste telle que le premier élément est la ressource allouée au premier colon, le deuxième élément la ressource allouée au deuxième colon, … (même remarque sur la modélisation).

Algorithm 1 Un algorithme approximatif (naïf)

Input: Une colonie E, un entier positif k
Ensure: Une affectation colon/ressource
  $i \leftarrow 0$
  $S \leftarrow$ solutionNaive(E)
  while $i < k$ **do**
    Choisir au hasard un pirate $p \in E$
    Choisir au hasard un voisin $q \in E$ de p
    $S' \leftarrow$ echange(p,q)
    if cout(S) > cout(S') **then**
      $S \leftarrow S'$
    end if
    $i \leftarrow i+1$
  end while
  return S

La fonction `solutionNaive(E)` retourne l'affectation naïve utilisée pour l'initialisation de l'allocation dans la partie 1 (le premier colon reçoit sa ressource préférée, le deuxième colon reçoit sa ressource préférée si elle est encore disponible, sinon sa deuxième ressource préférée,...).
La fonction `echange(p,q)` échange tout simplement l'objet du colon `p` avec l'objet du colon `q`.
La fonction `cout(S)` retourne le nombre de colons jaloux dans l'affectation S passée en paramètre.
Intuitivement, à partir d'une affectation potentielle, on essaye d'améliorer le résultat localement en échangeant les ressources de deux colons. Au bout de k (tentatives d')échanges, on s'arrête (autrement, l'algorithme bouclerait indéfiniment).
Bien entendu, cet algorithme ne garantit pas de trouver une solution optimale, il permet seulement de s'en approcher (plus ou moins bien). C'est pour cela que l'on parle d'algorithme approximatif. Cet algorithme peut vous servir de base de travail pour proposer un algorithme plus efficace.

III Tâches à réaliser

Pour la seconde étape du projet, vous devez développer un programme qui permet à un utilisateur de configurer la colonie à partir d'un fichier texte (voir Section I). Le programme prend en entrée **sur la ligne de commande** le nom d'un fichier (ou plutôt le chemin vers le fichier texte) dans lequel la colonie et les préférences sont définies. Rappelons que les paramètres de la ligne de commande sont utilisables via le paramètre `String[] args` de la méthode `main`. Si vous ne passez pas d'arguments en ligne de commande, le programme effectuera le travail que vous avez fait lors de la partie 1 du projet (i.e., construction manuelle de la colonie et recherche manuelle d'une meilleure solution).
Avant de passer à la suite, il vous est demandé de vérifier que les informations stockées dans le fichier texte respecte bien le format et ne comporte pas d'erreurs. Voici une liste non exhaustive des erreurs à vérifier :

- le fichier respecte bien la syntaxe définie dans la section I (e.g., l'ordre est bien respecté, pas d'oubli de point à la fin de chaque ligne, ne pas d'éléments autres que colon, ressource, deteste, preferences) ;
- avoir le même nombre de colon et de ressources ;
- avoir le bon nombre de paramètres pour chaque élément (1 pour colon et ressource, 2 pour deteste et $n+1$ pour préférences s'il y a n colons) ;
- les arguments passés en paramètre de deteste et preferences doivent avoir été définis ;
- ...

Si vous constatez une erreur, avant d'arrêter le programme, il faudra alors la signaler à l'utilisateur et expliquer l'origine du problème accompagné de la ligne du fichier à laquelle l'erreur est survenue (attention

l'utilisateur n'a pas forcément de notions d'informatique).

Une fois que le fichier est considéré comme correct, celui-ci est lu et un menu à trois options est proposé à l'utilisateur :

1) résolution automatique ;

2) sauvegarder la solution actuelle ;

3) fin.

Dans le cas où l'option 1 est retenue, un algorithme de votre choix (possiblement inspiré de l'algorithme décrit précédemment, mais ce n'est pas obligatoire) propose une solution, et affiche le coût de la solution. Plus votre algorithme sera efficace, plus vous aurez de points concernant cette partie. Après cela, on revient au menu.
Quand l'option 2 est sélectionnée, le programme demande à l'utilisateur le nom d'un fichier (différent de celui décrivant la colonie), et y enregistre la solution actuelle. Le contenu du fichier devra respecter le format suivant :

```
nom_colon_1:nom_ressource_1
nom_colon_2:nom_ressource_2
nom_colon_3:nom_ressource_3
```

Enfin, si l'option 3 est sélectionnée, le programme s'arrête. Il est nécessaire de gérer toutes les erreurs. Par exemple, quand le menu propose trois options, il faut indiquer à l'utilisateur que sa réponse est incorrecte s'il essaye de taper 3. Il est également demandé de gérer les exceptions, donc (par exemple) si l'utilisateur tape 1.5 ou « abc » au lieu d'un nombre entier, il faut gérer l'exception qui est levée.

Bonus (optionnel) : Un groupe a la possibilité d'obtenir des points bonus si :

- le code est convenablement couvert par des tests unitaires ;
- une interface graphique du programme est proposée ;

Veuillez noter que ces tâches sont optionnels et les points seront appliqués uniquement si l'ensemble des fonctionnalités demandées a été correctement implémenté (par exemple si la résolution automatique retourne un score qui ne correspond pas à l'affectation faite alors il n'y aura aucun point bonus même si une interface graphique et des tests unitaires ont été implémentés).

IV Remise du projet

La deuxième partie du projet doit être réalisée par les mêmes binômes/trinômes que la première partie. Votre code source, correctement documenté, sera à remettre sur Moodle au plus tard le 22 décembre 2024, sous forme d'une archive jar ou zip (un seul dépôt par binôme/trinôme). Avant d'exporter votre projet sous forme d'archive, pensez à le renommer en utilisant vos noms !
En plus du code source, vous devrez fournir un fichier README qui précisera :

- la classe doit être utilisée pour exécuter votre programme (c'est-à-dire où se trouve la méthode main),
- si vous avez implémenté un algorithme de résolution automatique plus efficace que celui proposé dans ce sujet et, si oui, expliquer celui-ci ;
- les fonctionnalités vous avez correctement implémentées, et lesquelles sont manquantes ou présentent des problèmes.

Le non-respect de certaines consignes pourra être pénalisé par un malus dans la note du projet.