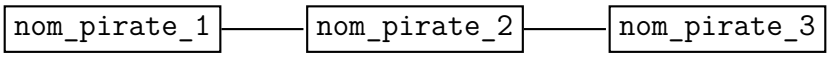
I Entrées-Sorties

Nous souhaitons maintenant que le programme puisse lire dans un fichier texte la description de l'équipage (c'est-à-dire le graphe des relations, et les préférences de chaque pirate). Un tel fichier respecte le format suivant :

```
pirate(nom_pirate_1).
pirate(nom_pirate_2).
pirate(nom_pirate_3).
objet(nom_objet_1).
objet(nom_objet_2).
objet(nom_objet_3).
deteste(nom_pirate_1,nom_pirate_2).
deteste(nom_pirate_2,nom_pirate_3).
preferences(nom_pirate_1,nom_objet_1,nom_objet_2,nom_objet_3).
preferences(nom_pirate_2,nom_objet_2,nom_objet_1,nom_objet_3).
preferences(nom_pirate_3,nom_objet_3,nom_objet_1,nom_objet_2).
```

Les pirates et les objets peuvent avoir un nom quelconque. Ce fichier correspond à l'équipage et aux préférences illustrés en Figure 1.




```
nom_pirate_1:nom_objet_1,nom_objet_2,nom_objet_3
nom_pirate_2:nom_objet_2,nom_objet_1,nom_objet_3
nom_pirate_3:nom_objet_3,nom_objet_1,nom_objet_2
```

FIGURE 1 – L'équipage et les préférences correspondant au fichier texte

II Automatisation

Nous présentons ici une algorithme naïf pour une résolution approximative du problème. On suppose ici que l'équipage 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 l'objet alloué au premier pirate, le deuxième élément l'objet alloué au deuxième pirate,… (même remarque sur la modélisation).

Algorithme 1 : Un algorithme d'approximation (naïf)

Entrées : Un équipage E, un entier k
Output : Une solution approximative du problème
int $i = 0$;
S = solutionNaive();
tant que $i < k$ **faire**
    choisir au hasard un pirate $p \in E$;
    choisir au hasard un voisin p' de p;
    S' = echange(p, p');
    si *cout(S)* > *cout(S')* **alors**
        $S = S'$;
    fin
    $i++$;
fin
retourner E

La fonction solutionNaive() retourne la solution naïve utilisée pour l'initialisation de l'allocation dans la partie 1 (le premier pirate reçoit son objet préféré, le deuxième pirate reçoit son objet préféré s'il est encore disponible, sinon son deuxième objet préféré,...). La fonction echange(p, p') échange tout simplement l'objet du pirate p avec l'objet du pirate p'. Intuitivement, à partir d'une solution potentielle, on essaye d'améliorer la solution localement en échangeant les objets de deux pirates. 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). Cet algorithme peut vous servir de base de travail pour proposer un algorithme plus efficace.

III Instructions

1. Tâches à réaliser
Pour la seconde étape du projet, vous devez développer un programme qui permet à un utilisateur de configurer l'équipage d'un bateau pirate à partir d'un fichier texte. Le programme prend en entrée sur la ligne de commande le nom d'un fichier dans lequel l'équipage et les préférences sont définis. Le nombre de pirates et leurs noms peuvent être quelconques, ainsi que les noms des objets.
Une fois que le fichier est lu, un menu à quatre options est proposé à l'utilisateur :

1) résolution automatique ;

2) résolution manuelle ;

3) sauvegarde ;

4) 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 son coût. Après cela, on revient au menu. Dans le cas où l'option 2 est retenue, l'utilisateur a la possibilité d'utiliser la résolution manuelle de la partie 1 (avec l'échange d'objet, l'affichage du coût, et la fin). Cette résolution manuelle doit être identique à ce qui était demandé dans la première partie du projet. Si cette option est choisie après qu'une résolution a déjà été faite (manuelle ou automatique), alors elle commence à partir de la dernière solution obtenue. Une fois que l'utilisateur souhaite la fin de la résolution manuelle, on revient au menu précédent.

Quand l'option 3 est sélectionnée, le programme demande à l'utilisateur le nom d'un fichier, et y enregistre la solution actuelle (qu'elle ait été obtenue par résolution manuelle ou automatique). Le contenu du fichier respecte le format suivant :

```
nom_pirate_1:objet_pirate_1
nom_pirate_2:objet_pirate_2
nom_pirate_3:objet_pirate_3
nom_pirate_4:objet_pirate_4
```

Enfin, si l'option 4 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 4. 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.

2. Remise du projet
Ce projet est à réaliser par groupes de **deux ou trois étudiants**, issus du **même groupe de TD** (les mêmes équipes que pour la partie 1). Votre code source, correctement documenté, sera à remettre sur Moodle au plus tard le 10 Décembre 2021, sous forme d'une archive jar ou zip (un seul dépôt par binôme/trinôme).