I Problématique

Après chaque pillage de navire, le capitaine d'un bateau pirate se sert en premier, puis répartit le reste du butin entre les membres de son équipage. Parmi toutes les richesses qu'il n'a pas gardées pour lui, il laisse un objet précieux à chaque pirate. Pour apaiser les tensions sur son navire, il demande à chaque pirate de lui donner ses préférences sur les objets à partager, et il essaye de les respecter au mieux. En plus de leurs préférences, il tient compte des relations au sein de l'équipage. Certains pirates s'apprécient peu, et s'il y a des sources de jalousies entre eux, cela pourrait mener à des disputes ou une mutinerie...
Le capitaine pirate nous demande de l'aider, en développant un logiciel qui permet :

1) de représenter les pirates et les relations entre eux ;

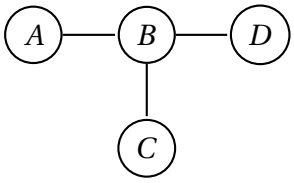
2) de simuler le partage des ressources entre eux ;

3) de calculer le coût d'une solution (le nombre de pirates jaloux), et de le minimiser.

Nous réaliserons ces tâches de façon incrémentale, afin d'avoir à la fin du semestre un logiciel fonctionnel.

II Modélisation

Une instance du problème est composée d'un équipage de n pirates, qui doivent se partager n objets. On représente les relations entre les membres de l'équipage comme un graphe non orienté, dans lequel chaque noeud représente un pirate, et chaque arête dans le graphe représente une relation « ne s'aiment pas » entre deux pirates. En plus du graphe, on connait les préférences de chaque pirate concernant les objets à partager. Pour un pirate donné p, ses préférences peuvent être représentées comme une liste l_p telle que l'objet $l_p[i]$ est strictement préféré à l'objet $l_p[j]$ si $i < j$. On note alors $l_p[i] \succ_p l_p[j]$. La Figure 1 décrit un exemple où l'équipage est composé de quatre pirates A, B, C et D, qui doivent se partager quatre objet o_1, o_2, o_3 et o_4. Les pirates s'entendent bien, sauf B qui n'aime personne (et n'est aimé de personne). La liste l_A associée au pirate A signifie que les préférences de A sont $o_1 \succ_A o_2 \succ_A o_3 \succ_A o_4$. Les autres listes définissent de manière similaire les préférences des autres pirates ($\succ_B, \succ_C, \succ_D$).



$A : o_1, o_2, o_3, o_4$
$B : o_1, o_3, o_2, o_4$
$C : o_3, o_2, o_1, o_4$
$D : o_1, o_4, o_2, o_3$

FIGURE 1 – Un exemple d'équipage avec les préférences des pirates

Une solution du problème est une affectation d'un objet à chaque pirate. Un pirate est jaloux si un de ses voisins obtient un objet qu'il aurait préféré avoir. Par exemple, si B obtient o_3, et C obtient o_2, alors C est jaloux de B car il aurait préféré avoir l'objet de son voisin. Le nombre de pirates jaloux représente le coût d'une affectation. Remarque : un pirate ne compte que pour 1 dans le coût de la solution, même s'il est jaloux de plusieurs de ses voisins.
La Figure 2 donne deux exemples de solutions. Dans la solution de gauche,

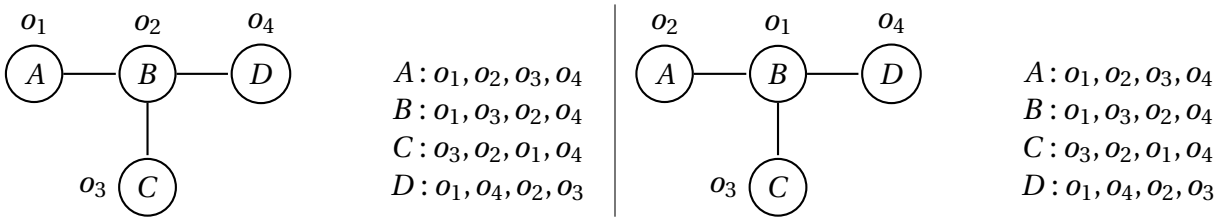


FIGURE 2 – Deux exemples de solutions

- A reçoit son objet préféré, il n'est pas jaloux. Même chose pour C.
- D reçoit son deuxième objet préféré. Il aurait préféré o_1, mais ce n'est pas grave dans ce cas, car ce n'est pas son voisin (B) qui a reçu cet objet. D n'est donc pas jaloux.
- B est jaloux de A qui a reçu o_1 (car $o_1 \succ_B o_2$) et de C qui a reçu o_3 (car $o_3 \succ_B o_2$).

Le coût de la solution est donc 1 (seul B est jaloux).
Dans la solution de droite,

- B et C ont reçu leur objet préféré, ils ne sont donc pas jaloux.
- A a reçu l'objet o_2, or il aurait préféré l'objet o_1 détenu par son voisin B. Donc A est jaloux de B. C'est la même chose pour D avec o_4.

A et D sont tous les deux jaloux de B, donc le coût de la solution est 2. Cette solution est donc moins bonne que la solution précédente.

III Instructions

1. Tâches à réaliser
Pour la première étape du projet, vous devez développer un programme qui permet à un utilisateur de configurer l'équipage d'un bateau pirate. Au démarrage, le programme doit demander à l'utilisateur le nombre de pirates n. Ce nombre est également égal au nombre d'objets que les pirates doivent se partager. On considèrera pour l'instant que les pirates sont nommés par les lettres de l'alphabet en majuscule, le premier s'appelant A, le deuxième B,… Le projet sera donc testé avec au plus 26 pirates (pour la partie 1, mais cela changera dans la partie 2). Les objets sont numérotés de 1 à n.
Une fois que le nombre de pirates n est fixé, un menu s'affiche avec deux options :

1) ajouter une relation ;

2) ajouter des préférences ;

3) fin.

Dans le cas où l'option 1 est retenue, on demande à l'utilisateur les pirates entre lesquels il faut ajouter une relation « ne s'aiment pas », puis on revient au menu précédent. Dans le cas où l'option 2 est retenue, l'utilisateur doit d'abord taper le nom d'un pirate, suivi de la liste qui correspond aux préférences du pirate. Par exemple,

```
A 1 2 3
```

signifie que le pirate préfère l'objet 1, puis l'objet 2, puis l'objet 3. Veillez à bien séparer les informations par (au moins un) espace. Dans le cas où l'option 3 est retenue, l'utilisateur a terminé de représenter les relations entre les membres de l'équipage. Dans ce cas, vous devez vérifier que chaque pirate possède bien une liste de préférences. L'utilisateur peut maintenant essayer de trouver une solution au problème. Une solution naïve est proposée automatiquement à l'utilisateur : chaque pirate (dans l'ordre de votre choix) reçoit son objet préféré s'il est disponible, ou son deuxième objet préféré s'il est disponible,...
Ensuite, l'utilisateur fait face à un menu qui propose trois options :

1) échanger objets ;

2) afficher coût ;

3) fin.

Quand l'option 1 est sélectionnée, on demande à l'utilisateur d'indiquer les noms de deux pirates, puis le programme échange les objets affectés à ces deux pirates.
Quand l'option 2 est sélectionnée, le programme affiche le coût de la solution actuelle.
Après chaque action de l'utilisateur, un message rappelle qui possède actuellement quel objet, avec le format suivant :

```
A:o2
B:o1
C:o3
D:o4
```

qui correspond à la solution décrit dans la partie droite de la Figure 2.
Enfin, si l'option 3 est sélectionnée, le programme s'arrête.
Il est nécessaire de gérer certaines erreurs. Par exemple, quand le menu propose trois options, il faut indiquer à l'utilisateur que sa réponse est incorrect s'il essaye de taper 4. Par contre, il n'est pas demandé (pour cette première partie) de gérer les erreurs dues à l'entrée d'une information de mauvais type (par exemple si l'utilisateur entre le nombre 1,5 au lieu d'un nombre entier). Ce type d'erreur sera à gérer dans la second partie.

2. Remise du projet

Ce projet est à réaliser par groupes de **deux ou trois étudiants**, issus du **même groupe de TD**. Votre code source, correctement documenté, sera à remettre sur Moodle au plus tard le 7 Novembre 2021, sous forme d'une archive jar ou zip (un seul dépôt par binôme/trinôme).
Des conseils sur l'implémentation seront fournis prochainement sur Moodle.