Projet « Elimination au baseball »

A réaliser par binôme, en Java 1.7

Note. Ce projet est une traduction partielle et avec modifications du document https://people.orie.cornell.edu/dpw/orie633/LectureNotes/lecture2.pdf[1] dans lequel vous pourrez trouver les preuves des différents résultats annoncés ici. Les détails sur l'organisation du baseball sont pris ici : https://tobykingsman.wordpress.com/2016/04/15/baseball-elimination-problem/

Aux Etats-Unis, le jeu de baseball est organisé en deux ligues, chacune contenant trois divisions de 4-5 équipes chacune. Pendant une saison, chaque équipe joue 162 matchs, qui ont lieu contre des équipes diverses : 76 matchs contre des équipes de sa propre division, 66 matchs contre des équipes de sa propre ligue mais d'autres divisions et 20 matchs contre des équipes de l'autre ligue. Le but de chaque équipe est de se qualifier pour les barrages, et pour cela il faut gagner sa division. Le but de ce projet est d'implémenter une méthode pour vérifier, à tout moment du championnat, quelles sont les équipes déjà éliminées (qui ne peuvent donc plus être premières de leur division) et lesquelles sont encore en lice.

Soit l'exemple suivant d'une division à 4 équipes. Le tableau indique le nom de chaque équipe, son nombre de victoires, son nombre de matchs restants et le détail des matchs restants (nombre de matchs restant par adversaire, les adversaires étant listés dans le même ordre que dans la 1ère colonne).

Team	Wins	Remaining Games	Games Against			
			NY	Bos	Tor	Bal
New York Yankees	93	8	-	1	6	1
Boston Red Sox	89	4	1	-	0	3
Toronto Blue Jays	88	7	6	0	-	1
Baltimore Orioles	86	5	1	3	1	-

Définition 1 Une équipe *gagne sa division* si, à la fin de tous les matchs, aucune autre équipe de la division n'a gagné plus de matchs qu'elle. (Des ex-aequo peuvent apparaître, c'est-à-dire que plusieurs équipes peuvent gagner la même division ; par contre, tout match est soit gagné soit perdu, sans égalité possible.)

Définition 2 Une équipe *est éliminée* à un moment donné du championnat si elle ne peut plus gagner sa division, quels que soient les matchs qu'elle gagnerait ou perdrait par la suite (mais – même si elle ne peut plus gagner sa division - elle joue tous les matchs restants).

Nous voulons savoir quelles équipes ont déjà été éliminées à un moment donné du championnat.. Clairement, dans l'exemple ci-dessus Baltimore est éliminée parce qu'elle finira la saison avec au plus 91 victoires alors que les Yankees ont déjà 93 victoires. Il est moins évident que Boston est

1 Evidemment, vous pourrez trouver une solution au problème dans ce document ou un autre. Ce n'est pas le but de l'exercice. Essayez de trouver la solution vous-même et, seulement si vous n'y arrivez pas, autorisez-vous à aller chercher ailleurs.

également éliminée. C'est vrai que Boston pourrait encore arriver à 93 matchs gagnés ... il semblerait donc qu'elle pourraient gagner à condition que les Yankees perdent le reste de leurs matchs. Cependant, si les Yankees perdent tous leurs matchs, cela signifie que Toronto va gagner 6 matchs contre les Yankees, ce qui donnera à Toronto les 94 victoires dont ils auraient besoin pour battre Boston. Notre objectif est de déterminer un moyen systématique de résoudre des problèmes comme celui-ci.

Notation. Soit T l'ensemble des équipes de la division (qui peut avoir – dans notre modélisation – un nombre arbitraire d'équipes, pas seulement 4 ou 5), et soient les notations suivantes pour toute équipe i de T :

w_i = nombre de victoires actuelles pour l'équipe i
g_i = nombre de matchs restant à jouer pour l'équipe i
g_{ij} = nombre de matchs restants entre l'équipe i et l'équipe j.

Afin de déterminer si une équipe k est éliminée ou non, au vu des scores courants et des matchs restants, un problème de flot doit être formulé et résolu. Il vous appartient de définir le réseau de transport RT_k pour chaque équipe k, et le problème de flot qui – une fois résolu – vous permettra de décider sans aucune erreur si l'équipe k est déjà éliminée ou si elle a encore une chance de se maintenir. Pour la définition de ce réseau, il est conseillée d'utiliser les notations données ci-dessus, mais vous pouvez en ajouter d'autres si vous en avez besoin.

Malheureusement pour le programmeur (et heureusement pour l'utilisateur), l'algorithme le plus efficace n'a pas besoin de tester toutes les équipes de cette manière, à cause de cette remarque :

Lemme. Si l'équipe k est éliminée, alors toutes les équipes h telles que $w_k+g_k>=w_h+g_h$ sont également éliminées.

Il s'en suit que $O(log|T|)$ constructions de réseaux et calculs de flots sont suffisants pour tester, étant donné un table[2]au comme celui de l'exemple avec $|T|$ équipes au lieu de 4, quelles équipes sont déjà éliminées et lesquelles non.

Travail à réaliser

Il vous est demandé de modéliser et de programmer (en Java 1.7[2], par binôme), la méthode décrite ci-dessus pour calculer quelles équipes ne sont pas encore éliminées à un moment donné du championnat. La méthode doit être *aussi efficace que possible.*

Le programme prendra en entrée un fichier avec la structure suivante.

- Sur la première ligne : le nombre n d'équipes (qui ne se limite pas à 4 ou 5 ; il est aussi grand qu'on veut)
- Sur les n lignes suivantes : pour chaque équipe, son indice i (dans l'ordre croissant de 1 à n) , son nom (sans espaces), son nombre de victoires, son nombre de matchs à jouer, et – pour toute équipe d'indice allant de 1 à n dans cet ordre – le nombre de matchs à jouer contre cette équipe (comme dans l'exemple fourni, sauf qu'on mettra -1 au lieu de «-» sur la diagonale). Pour l'exemple fourni, le fichier serait le suivant (un seul espace sépare deux colonnes) :

2 Afin d'éviter l'utilisation de fonctionnalités plus récentes qui permettent d'écrire du code dont la complexité serait insuffisamment maîtrisée.

```
4
1 New-York-Yankees 93 8 -1 1 6 1
2 Boston-Red-Sox 89 4 1 -1 0 3
3 Toronto-Blue-Jays 88 7 6 0 -1 1
4 Baltimore-Orioles 86 5 1 3 1 -1
```

Nous supposerons que les entrées dans ce fichier sont correctes. Le fichier se termine par une ligne vide.

Le programme devra permettre à l'utilisateur de savoir – en sortie – les équipes éliminées et comment il a été décidé qu'elles sont éliminées (plus précisément, pour quelles équipes *k* a été réalisé le test impliquant le réseau de transport et, dès qu'une équipe *k* a été identifiée comme éliminée, quelles autres équipes ont été considérées comme éliminées grâce au Lemme fourni).

Le programme implémentera, parmi d'autres méthodes, les méthodes suivantes :

- Preflot ou FordFulkerson, qui implémente la méthode des préflots ou celle de Ford-Fulkerson.
- ConstructionReseau, qui construit un réseau de transport selon la description fournie dans votre rapport, pour une équipe *k*.
- TestEliminationEquipe, qui réalise le test impliquant le réseau de transport pour une équipe *k*.
- TestEliminationToutes, qui décide pour toutes les équipes si elles sont éliminées ou non.

Le programme doit être entièrement issu de votre travail[3].

Travail à rendre

Les programmes doivent doivent fonctionner *parfaitement* sur les machines du CIE, sous Linux, en ligne de commande. Tout programme qui ne compile pas, ne démarre pas, ne récupère pas le fichier sous la forme indiquée, pose des questions que l'utilisateur ne comprend pas etc. ou qui a besoin qu'on entre dans le code pour une raison ou pour une autre sera considéré comme non-évaluable, et votre travail aura été inutile. Assurez-vous que vous fournissez un programme en état de marche et que l'utilisateur a toutes les informations nécessaires pour l'exécuter dès le premier essai.

Un rapport d'au maximum 12 pages sera fourni, comprenant (entre autres) :

- les commandes de compilation et exécution en mode console
- la modélisation du problème : la définition (avec un dessin) du réseau de transport, des capacités, du flot recherché, ainsi que la justification que la modélisation correspond parfaitement au problème posé.
- une vue globale de votre approche, sous la forme d'un algorithme (c'est-à-dire en pseudo-code) dont les instructions sont des appels à des procédures/fonctions ; la spécification et les paramètres de chaque procédure/fonction sont précisément décrits

3 Toute ressemblance - même mineure - entre deux programmes, ou entre un programme et du code sur Internet, sera considérée comme non-fortuite et entraînera automatiquement la note de 0 sur la partie concernée ainsi que sur toute autre partie utilisant, même très peu, la partie concernée. Une pénalité supplémentaire de 5 points sera également appliquée. Vous devez prendre des mesures pour vous assurer que votre code ne ressemble à aucun autre et n'est pas partagé, avec ou sans votre accord.

- pour chaque méthode parmi celles demandées, sauf l'algorithme de calcul de flot :
  - le détail de la procédure/fonction, sous forme algorithmique (c'est-à-dire en pseudo-code);
  - l'explication de son fonctionnement;
  - sa complexité ;
  - les raisons pour lesquelles vous considérez cet algorithme aussi efficace que possible.
- des jeux de données commentés

Important

Les fichiers du programme, ainsi que les jeux de données, seront mis dans un répertoire Nom1Nom2 (où Nom1 et Nom2 sont les noms des deux étudiants du binôme). Ce répertoire sera ensuite archivé sous le nom Nom1Nom2.zip. L'archive sera déposée sur Madoc, au plus tard le vendredi 9/12/2022 à 23h59 (Madoc n'acceptera pas de retard).

La toute dernière séance de TP (6/6) n'est pas une séance de travail au projet, mais une séance où vous devez faire une démo de votre projet.

Tout est important pour la notation. En particulier, il sera accordé beaucoup d'attention au respect des consignes et à la recherche d'une complexité minimum - garantie d'une efficacité maximum - pour votre méthode, via la mise en place des structures de données les plus adaptées.