Université de Nantes

Département d'informatique

Rapport projet Graphes 2

Présenté par : YADEL GAYA.
BEN HAMOUDI MELYSSA.

Enseignante : Irena Rusu

Année universitaire : 2024/2025

Présentation du projet et objectifs

TOURNOI DE BASEBALL EN GRAPHE

Présentation du problème :

Le but de ce projet est de modéliser et d'implémenter une méthode pour vérifier si une équipe dans un championnat de baseball (aux USA) peut se qualifier pour les barrages. Pour cela, chaque équipe doit gagner sa division.

Chaque équipe joue un ensemble de matchs contre d'autres équipes, et un match gagné rapporte 1 point supplémentaire. Les équipes jouent le même nombre de matchs durant la saison.

Pour la modélisation, des instances (jeux de données) sont utilisées, présentées de la manière suivante :

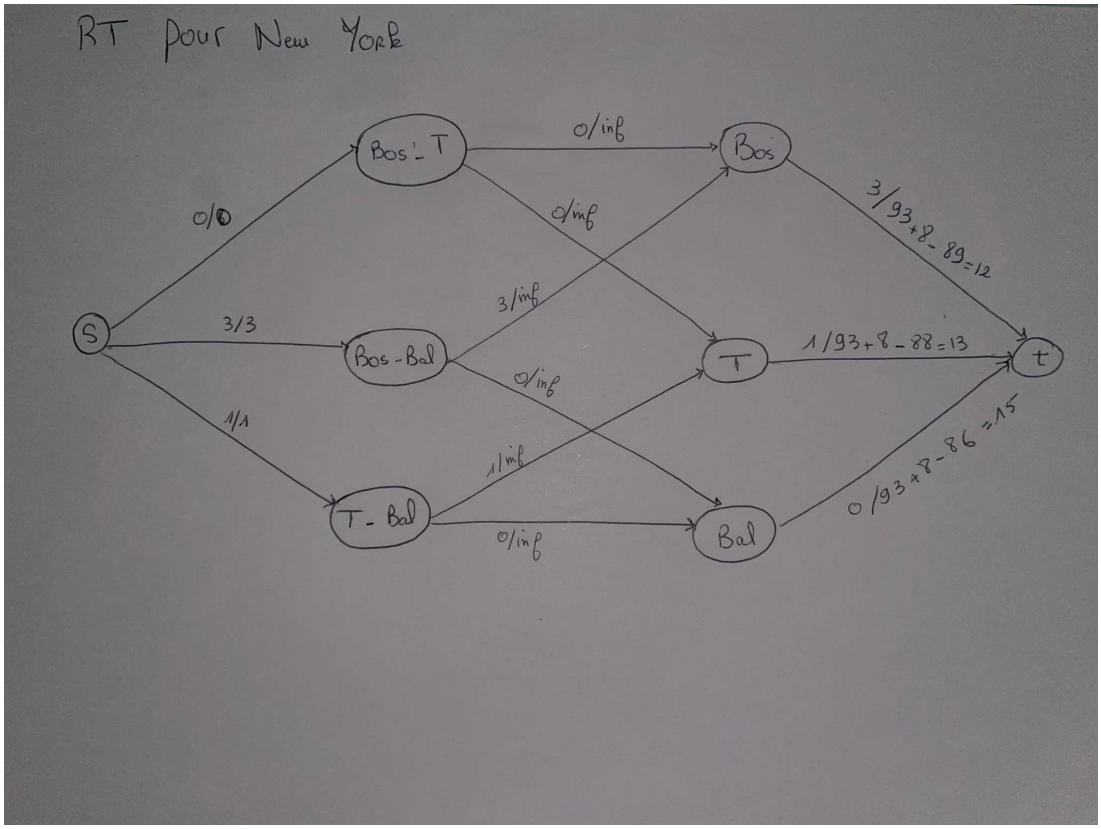
4

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

La première colonne représente le numéro de l'équipe, la deuxième son nom, suivie des points, du nombre de matchs restants, et enfin des matchs restants contre chaque équipe (le -1 indique qu'une équipe ne peut pas jouer contre elle-même).

Modélisation du problème :

Pour modéliser ce problème, nous avons besoin de tracer un réseau de transport pour chaque équipe en fonction de l'instance donnée. Prenons l'exemple de l'équipe New-York-Yankees, dont le réseau de transport est représenté ci-dessous :



Explication du réseau :

1) Un sommet s représente la source et un sommet t représente le puits, comme pour tous les réseaux de transport.

2) La source est reliée à 3 sommets représentant les matchs entre les autres équipes (sans compter les matchs impliquant New-York-Yankees). Ainsi, on a 3 matchs : BOSTON-TORONTO, BOSTON-BALTIMORE, et TORONTO-BALTIMORE.

3) Les sommets des matchs sont reliés à 3 autres sommets représentant les équipes qui jouent ces matchs (Boston, Toronto, Baltimore).

4) Ces sommets sont ensuite reliés au sommet t, représentant le puits.

5) Les arêtes entre la source et les matchs représentent les matchs devant être joués (par exemple : BOSTON-BALTIMORE doivent jouer 3 matchs).

6) Les arêtes entre les matchs et les équipes représentent le nombre de points qu'une équipe peut gagner si elle remporte ses matchs. Par exemple, pour le match BOSTON-BALTIMORE, si BALTIMORE gagne ses 3 matchs, son flux devient 3 sur l'infini, car elle gagne 3 points. Si au contraire, BALTIMORE ne remporte aucun match, son flux reste à 0.

7) Les arêtes entre les équipes et le puits représentent le nombre de points qu'elles peuvent gagner en fonction de leur nombre total de matchs remportés.

Applications :

Ce travail est programmé sous Java 1.7, en utilisant l'algorithme de Ford-Fulkerson pour calculer un flot maximal. Pour chaque équipe, nous testerons si elle est éliminée en fonction de son flux. Le projet nécessite les classes suivantes :

- **Championnat** : qui construit un championnat par division.
- **Equipe** : qui donne les caractéristiques de chaque équipe.
- **Ford-Fulkerson** : qui applique l'algorithme de Ford-Fulkerson.
- **RéseauTransport** : qui construit le réseau de transport pour chaque équipe.
- **Main** : pour exécuter et appeler nos classes et méthodes.
- **Edge et FlowNetwork** : classes additionnelles pour gérer les arêtes et les réseaux de flots.

Class et méthodes utilisées

Présentation des classes et méthodes utilisées

Introduction

Le programme en Java a pour but de simuler un tournoi entre plusieurs équipes et de déterminer si une équipe peut être éliminée en fonction de ses matchs restants et de ses victoires possibles. Pour ce faire, le programme utilise des réseaux de transport et un algorithme de flot maximal (Ford-Fulkerson) pour simuler les matchs restants et vérifier si l'équipe peut être éliminée. Le code est composé de plusieurs classes, chacune ayant un rôle spécifique dans le processus de calcul.

1 Classes et Fonctionnalités

1.1 Classe `Equipe`

La classe `Equipe` représente une équipe participante au tournoi. Elle contient les informations suivantes :

- **nom** : Le nom de l'équipe.
- **victoires** : Le nombre de victoires de l'équipe dans le tournoi.
- **matchsRestants** : Le nombre de matchs restants pour l'équipe.
- **matchsContre** : Un tableau représentant le nombre de matchs restants contre chaque autre équipe.

Cette classe permet de stocker et gérer les informations nécessaires pour simuler les matchs et vérifier si l'équipe peut être éliminée.

1.2 Classe `FlowNetwork`

La classe `FlowNetwork` est utilisée pour créer un réseau de transport qui représente les matchs restants entre les équipes. Elle construit un graphe dans lequel les sommets représentent les équipes et les arcs représentent les matchs restants. Cette classe permet de modéliser le problème sous forme de réseau pour calculer les flots maximaux et analyser les éliminations possibles des équipes.

1.3 Classe `FordFulkerson`

La classe `FordFulkerson` implémente l'algorithme de Ford-Fulkerson pour calculer le flot maximal dans un réseau. Elle est utilisée pour déterminer le maximum de matchs que chaque équipe peut encore gagner, en fonction des matchs restants contre les autres équipes. L'algorithme est essentiel pour analyser si une équipe peut être éliminée ou non.

1.4 Classe `Main`

La classe `Main` est le point d'entrée du programme. Elle gère la lecture du fichier d'entrée contenant les informations sur les équipes et leur nombre de victoires, ainsi que les matchs restants. Elle crée ensuite les objets `Equipe`, les ajoute à un `Championnat`, et pour chaque

équipe, elle vérifie si elle est éliminée à l'aide des réseaux de transport et de l'algorithme de Ford-Fulkerson.

1.5 Classe `ReseauTransport`

La classe `ReseauTransport` représente un réseau de transport spécifique pour chaque équipe. Elle est utilisée pour construire un réseau de flot et vérifier si l'équipe est éliminée en fonction des matchs restants. Elle contient les méthodes suivantes :

- `buildFlowNetwork()` : Construit un réseau de flot à partir des informations des matchs restants.
- `estEliminee()` : Vérifie si l'équipe est éliminée en utilisant l'algorithme de flot maximal.

1.6 Classe `Championnat`

La classe `Championnat` gère l'ensemble des équipes dans le tournoi. Elle crée des objets `ReseauTransport` pour chaque équipe et coordonne le calcul des flots maximaux pour vérifier si une équipe peut être éliminée. Elle contient des méthodes pour créer le réseau de transport pour chaque équipe.

1.7 Classe `Edge`

La classe `Edge` représente une arête dans le réseau de transport, reliant deux sommets (équipes). Elle contient des informations sur le poids de l'arête, qui correspond au nombre de matchs restants entre les deux équipes. Cette classe est utilisée par `FlowNetwork` pour construire le graphe de matchs restants.

2 Fonctionnement du Programme

Le programme commence par lire un fichier d'entrée `JDD3.txt`, qui contient les informations sur les équipes et leurs matchs restants. Chaque équipe est représentée par un objet de la classe `Equipe`, qui est ajouté à une liste. Ensuite, un objet `Championnat` est créé pour gérer l'ensemble des équipes.

Pour chaque équipe, un réseau de transport est créé à l'aide de la classe `ReseauTransport`. Le réseau est ensuite utilisé pour calculer le flot maximal avec l'algorithme de Ford-Fulkerson. Si l'équipe peut être éliminée, un message indiquant son élimination est affiché, sinon, un message indiquant qu'elle n'est pas éliminée est affiché.

3 Gestion des Erreurs

Le programme inclut des mécanismes de gestion des erreurs pour assurer la validité des données :

- Vérification si le fichier d'entrée existe et est accessible.
- Validation des données d'entrée pour s'assurer qu'elles sont complètes et bien formatées.
- Gestion des erreurs liées à la construction des réseaux de transport et au calcul du flot maximal.

Compilation et exécution

Compilation et exécution de l'ensemble des codes sources

Pour compiler et exécuter le programme Java, voici les étapes à suivre :

1) Compilation du code :

La commande suivante compile le fichier 'Main.java' qui contient la méthode 'Main' de notre programme. On s'assure que tous les fichiers Java nécessaires (tels que 'Championnat.java', 'Equipe.java', 'FordFulkerson.java', etc.) sont dans le même répertoire ou correctement référencés dans notre projet.

```
javac Main.java
```

Cette commande génère des fichiers '.class' correspondant à chaque fichier '.java' source. Par exemple, après avoir exécuté la commande 'javac Main.java', un fichier 'Main.class' sera créé, qui peut ensuite être exécuté.

2) Exécution du programme :

Une fois le code compilé, on pourra exécuter notre programme avec la commande suivante. Cette commande lance la méthode 'main' de notre programme Java :

```
java Main
```

Le programme va s'exécuter et nous permettera de tester différentes instances. Il suffit de fournir les fichiers d'instance appropriés en entrée pour que le programme fonctionne sur eux.

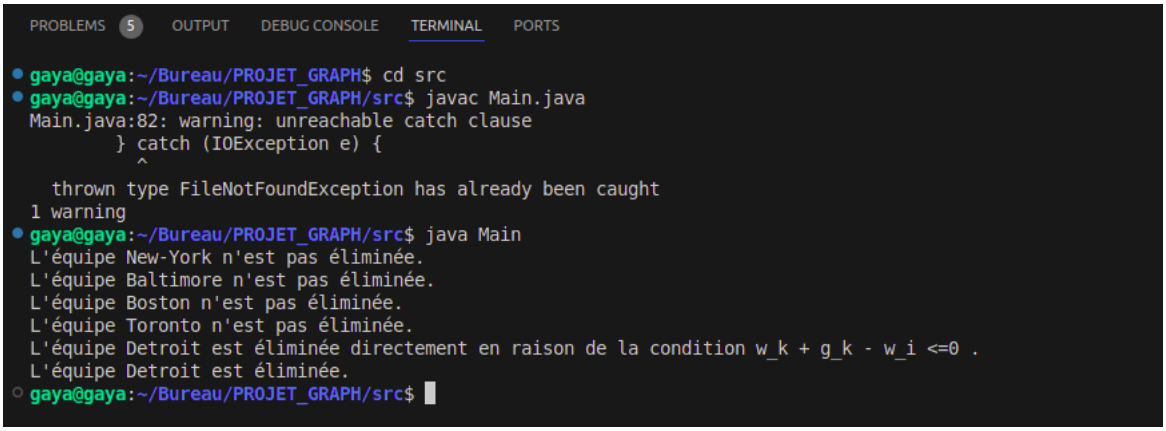
3) Tests sur des jeux de données :

On va tester le programme sur les deux instances suivantes : 'JDD1.txt' et 'JDD3.txt'. Ces fichiers contiennent les données des équipes et des matchs pour les différents cas de test.

Pour tester ces instances, on s'assure que les fichiers sont correctement placés dans le même répertoire que notre fichier 'Main.java'.

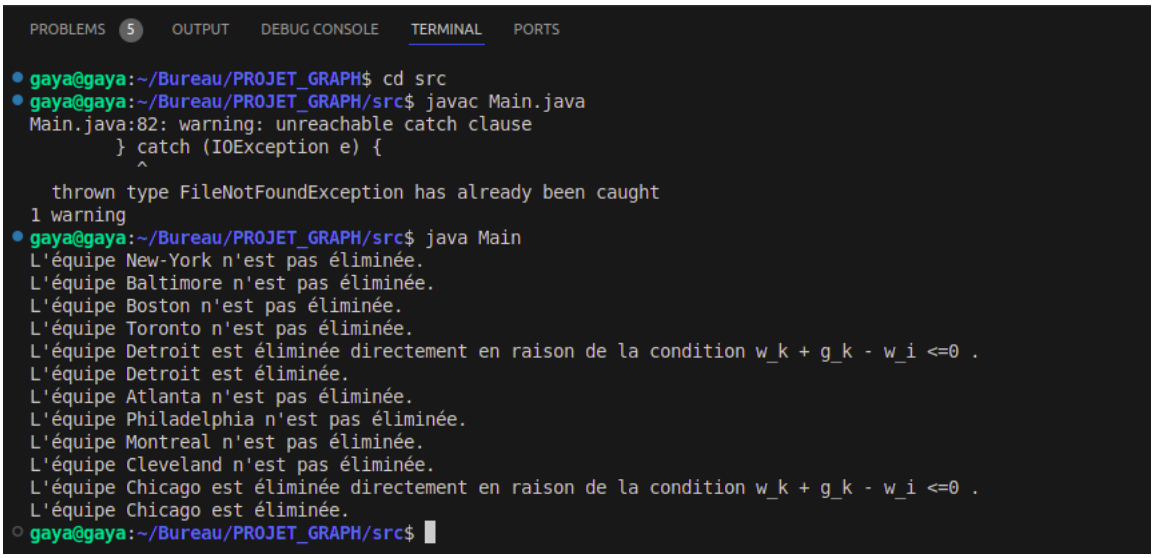
Test sur l'instance 'JDD1.txt' :

Voici un exemple de test avec l'instance 'JDD1.txt' :



Test sur l'instance 'JDD3.txt' :

Après avoir testé l'instance 'JDD1.txt', on va effectuer un autre test avec l'instance 'JDD3.txt' :



Remarques :

1) Avant l'exécution faut se rendre sur le sous dossier src avec la commande "cd src"

2) Lors de l'affichage des résultats, on affiche aussi la raison de l'élimination de l'équipe

3) Le fichier de jeux de donnée, on le lit par les commandes suivantes qui sont dans le script Main :

File file = new File("JDD3.txt") ; // Utilisation directe du fichier dans le répertoire src

Scanner sc = new Scanner(file) ; // Lecture du fichier d'entré avec scanner