

UNIVERSITÉ NATIONALE DU VIETNAM À HANOÏ

INSTITUT FRANCOPHONE INTERNATIONAL



RECHERCHE OPÉRATIONNELLE

RAPPORT DE MINI-PROJET

Affectation de tâches

Août 2019

Étudiants :

Cleg Peter OVIL

Mike Arley MORIN

Afi Elolo Gisèle DEKPE

Enseignant : **M. NGUYEN Hoang Thach**

Promotion 23, RSC || Année Académique : 2019 - 2020

Table des matières

Introduction	2
1 Étude contextuelle	2
1.1 Contexte	2
1.2 Problématique	2
1.3 Résultats attendus	2
2 Analyse	3
2.1 Données d'entrée	3
2.2 Données en sortie	3
3 Implémentation	3
3.1 Le BFS	4
3.2 Modification du réseau	4
3.3 Construction du réseau à flot	5
3.4 Ford – Fulkerson	5
3.5 Finalisation	6
Conclusion	10

Introduction

Ce mini projet d'implémentation d'une solution pour pallier au problème d'affectation de tâches s'inscrit dans le cadre du cours de Recherche Opérationnelle. Le rapport présente nos réalisations concernant ce mini projet soumis à notre recherche ; ceci afin d'acquérir des connaissances non seulement sur les algorithmes de recherche plus précisément celui de **Ford-Fulkerson** mais aussi. Ce travail comporte une étude contextuelle, puis l'analyse des données dont nous disposons, la phase d'implémentation puis les résultats de nos tests.

1 Étude contextuelle

1.1 Contexte

Affecter des tâches signifie attribuer les tâches aux personnes de la manière la plus optimale possible. Ceci devra se faire de sorte qu'une personne ne puisse exécuter qu'une seule tâche au plus ; et ce vice-versa.

1.2 Problématique

La problématique qui s'en dégage pour notre projet, est qu'il faut déterminer le couplage maximal tout en respectant les spécifications du sujet. Il s'agit donc de d'affecter un nombre " N " de tâches et de personnes. Le problème est de maximiser le couplage avec les conditions citées plus haut dans le contexte. Ce problème peut donc se modéliser au travers d'un graphe biparti dans lequel u est l'ensemble des tâches et v l'ensemble des personnes. Les arêtes entre les sommets représenteront la possibilité qu'une tâche puisse être effectuée par une personne. Il existe plusieurs algorithmes pour résoudre le problème ainsi posé mais le sujet demande qu'on utilise celui de **Ford-Fulkerson**.

1.3 Résultats attendus

De façon claire, on attend à la fin d'avoir une affectation de tâches telles que :

- une personne fasse une tâche ;
- le nombre de personnes satisfaites soit maximale ;


Le tout matérialisé dans un programme java qui prend des données en entrée et produit des données en sortie dont nous parlerons dans les prochaines lignes.

2 Analyse

2.1 Données d'entrée

En entrée du programme, nous avons un fichier texte contenant :

- la première ligne contient N , le nombre de personnes et de tâches (il y a autant de tâches que de personnes).
- chacune des lignes du reste est la liste des tâches de préférence d'une personne.
- il s'agit des numéros de tâche séparés par des espace. Pour simplifier, les personnes et les tâches sont numérotées de 1 à N



```
5
2 3
1 4 5
3
1 2
3 4
```

2.2 Données en sortie

Le programme devra produire une liste, ou un fichier texte qui contient cette liste, des numéros de tâches affectées à des personnes de 1 à N , séparés par des espaces. Par exemple, avec $N = 3$, une liste “2 3 1” veut dire que la personne 1 fait la tâche 2, la personne 2 fait la tâche 3, la personne 3 fait la tâche 1.

3 Implémentation

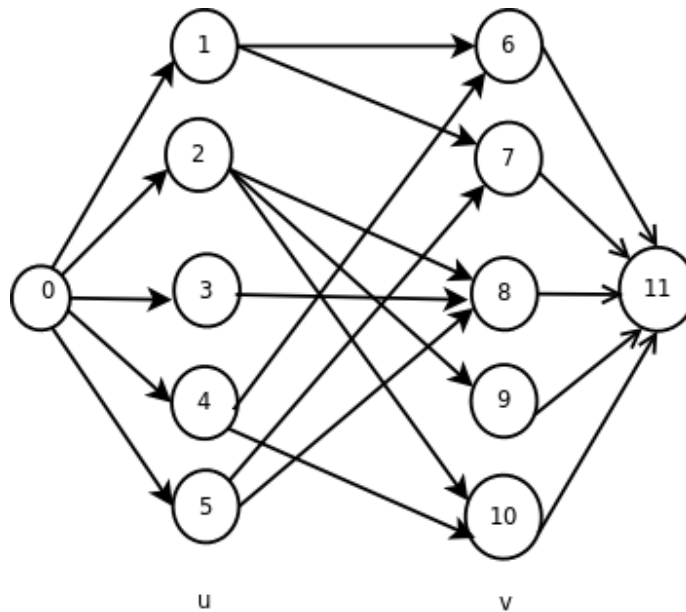
Des fichiers **Noeud.java** et **Graphe.java** ont été mis à notre disposition comme éléments de base pour créer et travailler avec des graphes orientés. Le travail du mini-projet consiste à gérer la lecture et l'écriture des fichiers, créer le réseau à flot associé au problème (qui est en fait un graphe orienté non-pondéré) et

implémenter une version simple de l'algorithme de Ford-Fulkerson qui n'utilise que le **BFS**. Les étapes suivies pour l'implémentation sont celles décrites par l'énoncé de ce mini-projet. En résumé, elles sont les suivantes :

3.1 Le BFS

En anglais **Breadth First Search**, il consiste à faire un parcours du graphe en largeur. L'algorithme qu'il utilise met le nœud source dans une file pour le traiter. Ensuite, il met tous les voisins non explorés dans la file et si la file n'est pas vide reprendre à l'étape où l'un des nœuds marqués possèdent des successeurs non traités.

Un exemple :



Dans notre cas, il nous est demandé d'écrire une méthode `public LinkedList<Noeud> cheminBFS(Noeud s, Noeud t)` dans la classe **Graphe**, qui utilise cet algorithme et qui renvoie *null* si un tel chemin n'existe pas. Les nœuds 0 et 11 correspondent respectivement à la source *s* et au puits *t*.

Avec un tel graphe, le résultat de l'algorithme serait :

Notre Chemin BFS :
0->6->7->8->9->10->2->3->1->4->5->11

3.2 Modification du réseau

Il nous est ensuite demandé d'écrire une méthode `public boolean inverseChemin(LinkedList<Noeud> chemin)` dans la classe **Graphe**, qui renvoie *true* si l'opération est faite avec succès, *false* s'il y a un arc du chemin qui n'appartient pas au graphe.

Si nous considérons le graphe précédent, le chemin inverse nous donne :

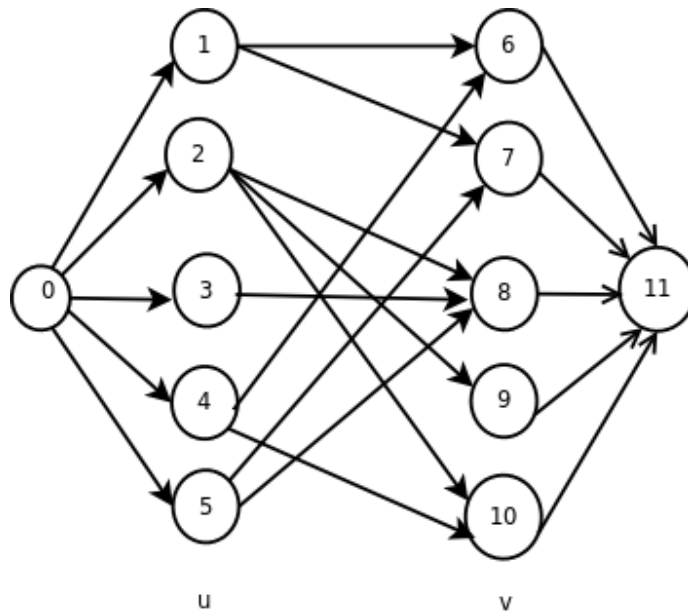
Le chemin BFS inverse :
11->1->2->3->4->5->7->9->6->8->10->0

Nous rappelons que cette méthode ne fonctionne pas bien pour un graphe général, mais fonctionne correctement sur les réseaux à flot de notre problème qui sont simples et acycliques.

3.3 Construction du réseau à flot

Pour la construction du réseau à flot, il nous faudra créer une nouvelle classe **Couplage** dans laquelle on écrira une méthode **public static Graphe creeReseau(String fichier)**. Cette méthode prendra le fichier d'entrée en paramètre.

En recherche opérationnelle, un réseau à flot c'est un graphe orienté. Un graphe orienté est un graphe dans lequel chaque arête possède une capacité et peut recevoir un flux. Le cumul des flux sur une arête ne peut pas et ne doit pas excéder sa capacité. Pour notre projet, nous nous proposons le réseau de flot suivant :

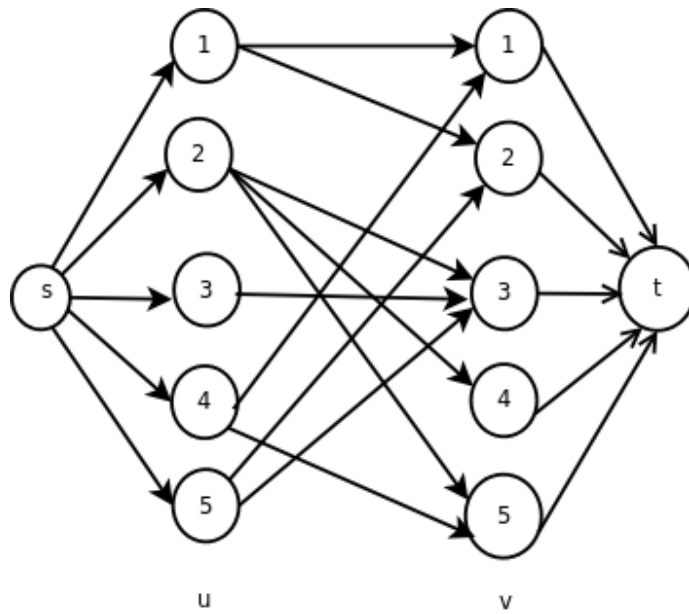


3.4 Ford – Fulkerson

Il nous faut écrire une méthode *public int ff(Noeud s, Noeud t)* qui exécute une version de l'algorithme Ford – Fulkerson utilisant le BFS.

Nous ramenons notre problème en un problème de graphe biparti.

Soit un graphe biparti G dont on note u et v, les deux sous-ensembles d'arêtes,



La valeur renvoyée est le nombre final de couples, qui est le nombre de successeurs du puits t dans le réseau final, parce que le nombre de tâches est égale au nombre de personnes ; si nous considérons l'ensemble des successeurs du puits t comme l'ensemble des personnes et celui des prédécesseurs de source s comme l'ensemble des tâches. On est dans le cas d'un dans le cas couplage parfait.

3.5 Finalisation

Il nous est demandé d'écrire dans la classe *Couplage* une méthode *main* qui lit un fichier d'entrée dont le chemin est passé en argument, puis calcule et affiche un couplage maximal.

Dès l'exécution du programme :

- il est demandé d'entrer le chemin permettant d'accéder au fichier à traiter
- lors du chargement du fichier, nous vérifions d'abord l'existence et l'exploitabilité du fichier chargé

Le traitement nous donne :

```
Console  Couplage.java
<terminated> Couplage (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 30, 201
Donner le chemin du fichier
file.txt
2
3
-1
-1
-1
1
2
4
5
-1
3
5
-1
-1
-1
1
2
4
-1
-1
3
4
-1
-1
-1
```

Notre Chemin BFS :

0->6->7->8->9->10->2->3->1->4->5->11

Chemin BFS inverse :

->11->1->2->3->4->5->7->9->6->8->10->0

Ford Fulkerson :

Le flot maximal est : 5

2 1 5 4 3

Interprétation : Ceci implique que la personne 1 fait la tâche 2, la personne 2 fait la tâche 1, la personne 3 fait la tâche 5, la personne 4 fait la tâche 4 et la personne 5 fait la tâche 3.

En essayant de changeant de changer les données du fichier d'entrée, nous obtenons des résultats tout aussi satisfaisants. Nous avons testé avec le jeu de données proposé par le professeur en classe, mis dans le fichier **test.txt** dont le contenu est le suivant :

5		
2	3	4
1	2	
5	3	
4		
1	2	

On obtient les résultats suivants :

```

Console  Couplage.java  Graphe.java  Noeud.java  file.txt  test.txt
<terminated> Couplage (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (Aug 30, 2019, 8:30:43 PM)
Entrer le nomchemin du fichier
test.txt
2
3
4
-1
-1
1
2
-1
-1
-1
5
3
-1
-1
-1
-1
4
-1
-1
-1
-1
1
2
-1
-1
-1

```

Notre Chemin BFS :

0->6->7->8->9->10->2->3->4->1->5->11

Chemin BFS inverse :

->11->1->2->3->4->5->7->10->6->8->9->0

Ford Fulkerson :

Le flot maximal est : 5

2 5 1 4 3

Interprétation : Ceci implique que la personne 1 fait la tâche 2, la personne 2 fait la tâche 5, la personne 3 fait la tâche 1, la personne 4 fait la tâche 4 et la personne 5 fait la tâche 3.

Conclusion

Ce mini-projet nous a permis comprendre les principes de l'affectation de tâches en utilisant l'algorithme de Ford-Fulkerson et le BFS ; tout en appliquant ce qui nous a été enseigné au cours de Recherche Opérationnelle. Ceci nous a également permis de confronter nos idées et d'apprendre à travailler en équipe. Nous y sommes arrivés malgré les difficultés techniques et celles liées au travail en équipe. Nous rattachons à ce document, le code source de notre projet avec les fichiers nécessaires pour la compilation et l'exécution.

Références

- Cours de Recherche Opérationnelle, M. NGUYEN Hoang Thach
- Chapitre 06 - Recherche d'un flot maximum dans un réseau de transport | SUPINFO, École Supérieure d'Informatique.html