

INFO-F-203

Projet: Les bons comptes font les bons amis

Grigore ANTONIUC

Pavlo POLINETSKYI

19 décembre 2016

Table des matières

	2
1 Introduction	3
1.1 Le problème	3
1.2 Objectifs	3
2 Simplification de dettes	4
2.1 Description	4
2.2 Implémentation	4
2.2.1 Explication	4
2.3 Exemple	4
2.3.1 Pseudocode	5
3 Identification des communautés	7
3.1 Description	7
3.2 Implémentation	7
3.2.1 Pseudocode	7
3.3 Exemple	8
4 Identification des hubs sociaux	9
4.1 Description	9
4.2 Implémentation	9
4.2.1 Pseudocode	10
4.3 Exemple	11
5 Travail en Equipe	12
6 Sources	12

7	Tests	13
7.1	OurTests.py	13
7.1.1	Tests de simplification de dettes	13
7.1.2	Test d'identification de communautés	15
7.1.3	Test d'identification de Hubs Sociaux	16
7.2	YourTester.py	17

1 Introduction

Ce rapport accompagne le projet d'Algorithmique de deuxième année, reposant sur la manipulation de graphes.

L'énoncé laissant le choix du langage entre *Java* ou *Python*, ce projet est codé en *Python3*.

1.1 Le problème

Le sujet du projet met l'étudiant dans la peau d'un informaticien devant créer un programme mobile censé gérer les dettes entre groupes d'amis.

Un groupe d'amis est représenté sous la forme d'un graphe dirigé lu à partir d'un fichier, dans lequel les noeuds représentent les individus et les arêtes représentent les dettes entre ceux-ci.

1.2 Objectifs

L'énoncé du projet demande d'implémenter plusieurs algorithmes dont nous avons choisi les suivants.

- La simplification de dettes
- L'identification des communautés
- L'identification des hubs sociaux

2 Simplification de dettes

2.1 Description

Cet algorithme reçoit en paramètre un graphe et renvoie ce même graphe, mais présentant une simplification des dettes.

2.2 Implémentation

2.2.1 Explication

La simplification du graphe est basée sur le principe que tout cycle de dettes peut être simplifié. Cela est fait en prenant la plus petite dette présente dans le cycle, et en la soustrayant de toutes les dettes du cycle. Ainsi, il est garanti qu'une dette soit réduite à 0.

L'algorithme fonctionne en trouvant toutes les composantes fortement connexes à l'aide de l'algorithme de Tarjan et en appliquant l'algorithme de Johnson sur celles-ci afin de trouver tous les cycles élémentaires du graphe.

La dernière étape consiste dans le fait de parcourir tous les cycles, en soustrayant la dette minimale contenue dans ceux-ci.

Complexité : $O((n + e)(c + 1))$, où n est le nombre de noeuds, e est le nombre d'arcs et c le nombre de cycles du graphe.

2.3 Exemple

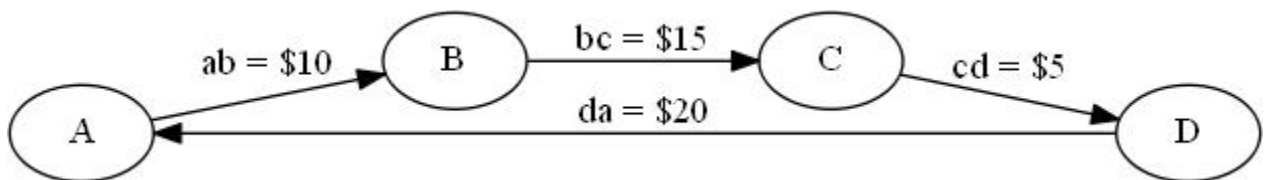


FIGURE 1 – Supposons les relations de dettes ci-dessus

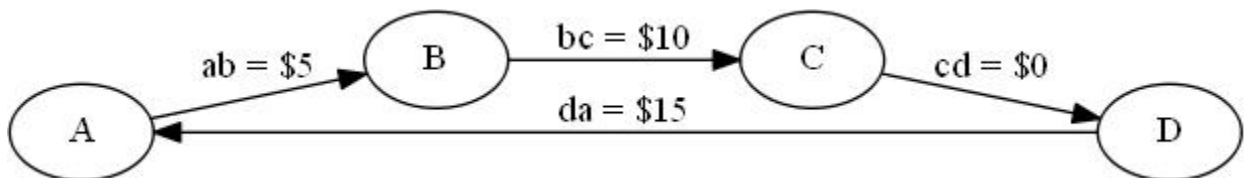


FIGURE 2 – Tel est l'état du graphe après simplification

2.3.1 Pseudocode

Algorithm *Tarjan*

Result: List of Strongly Connected Components

Function *find_sccs*

Data: allnodes, removednodes

for *each node in allnodes* **do**

if *node not in removed and node not treated* **then**

 explore(node)

end

end

Function *explore*

Data: node

 node.index = index node.lowestbackedge = index node.min = index

 stack.push(node)

for *edge in node* **do**

if *edge not in removed and edge not treated* **then**

 explore(edge) node.min = min(node.min, edge.lowestbackedge)

end

end

if *node.min < node.lowestbackedge* **then**

 node.lowestbackedge = node.min

end

else

 We have found a strongly connected component

 Pop nodes from the stack until we reach this node and add them to list of
 sccs.

end

Algorithm *Cycles_Johnson***Result:** List of Cycles**Function** *find_all_cycles***Data:** allnodes

```
list = Tarjan(allnodes) for component in list do
    while component has more than one node in it do
        findcycle(first node of component)
        delete first node of component
        recalculate sccs using Tarjan
        add any additional sccs produced due to splitting to list
    end
end
```

Function *findcycle***Data:** startingnode, currentnode, blockedset, blockedmap

```
cyclefound = False
stack.push(node)
blockedset.add(node) for edge in node do
    if currentnode is startingnode then
        foundcycle = True
        Pop nodes from stack until we reach starting node and add them to list
        of cycles
    end
    else if edge not in blockedset then
        recursionfoundcycle = findcycle(edge) cyclefound = cyclefound or
        recursionfoundcycle
    end
end
if cyclefound then
    | unblock(currentnode)
end
else
    | map this node in the blockedmap to all its edges
end
pop current node from stack
return cyclefound
```

Function *unblock***Data:** node, blockedset, blockedmap

```
blockedset.remove(node) if node in blockedmap then
    | unblock all other nodes mapped to this one
end
```

3 Identification des communautés

3.1 Description

L'algorithme d'identification des communautés doit recevoir un graphe et retourner l'ensemble des noeuds liés par des dettes.

3.2 Implémentation

L'algorithme d'identification des communautés correspond simplement à un parcours en profondeur du graphe à partir de chaque noeud.

Cet algorithme nous permet de détecter si notre graphe est divisé en plusieurs sous-graphes non-connexes et donc de trouver les différentes communautés.

Complexité : $O(n + e)$ où n est le nombre de noeuds et e est le nombre d'arcs.

3.2.1 Pseudocode

Algorithm *find_comunities*

```
Data: nodes_list  
Result: community_list  
for node in nodes_list do  
    if node is not visited then  
        community_list.append([])  
        call explore(node)  
    end  
end  
return community_list
```

Algorithm *explore*

```
Data: node  
Result: None  
visited[node] = 1 => node is visited  
community_list[-1].append(node)  
for related in adjacents of node do  
    if related is not visited then  
        call explore(related)  
    end  
end
```

3.3 Exemple

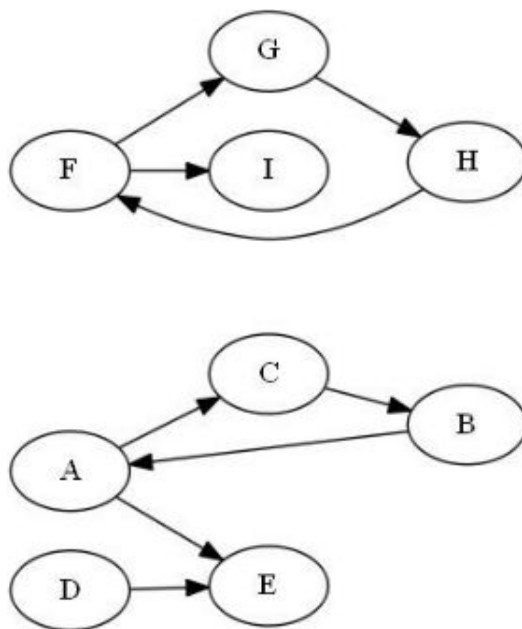


FIGURE 3 – L'image ci-dessus présente un graphe non connexe composé de deux sous graphes.

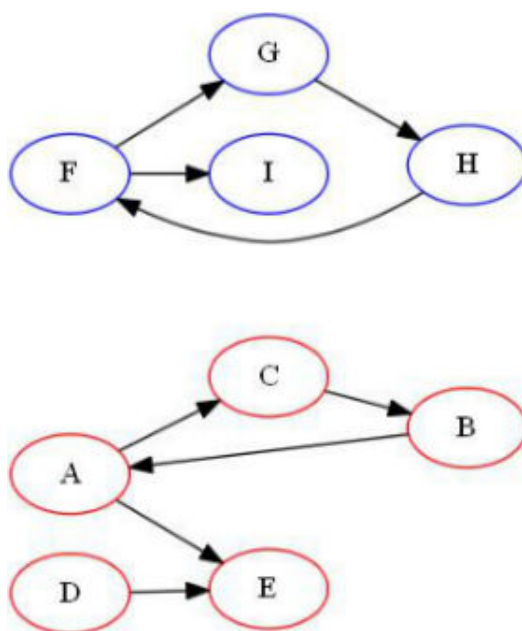


FIGURE 4 – Après exécution de l'algorithme, ces deux sous-graphes seront interprétés comme deux communautés distinctes.

4 Identification des hubs sociaux

4.1 Description

L'identification des hubs sociaux d'une communauté consiste en l'identification des noeuds dont la suppression scinderait celle-ci en deux ou plusieurs communautés.

Une condition supplémentaire est que ce noeud doit diviser une communauté en communautés d'au moins K éléments, où K est donné comme paramètre à l'algorithme.

4.2 Implémentation

Un hub social correspond un point d'articulation d'un graphe. Ainsi, une recherche des points d'articulation du graphe représentant la communauté permet d'identifier les hubs sociaux de celle-ci.

Une fois tous les points d'articulation identifiés, il faut vérifier la condition supplémentaire afin de ne renvoyer que les points divisant le graphe en sous-graphes dont le nombre de noeuds est $\geq K$.

Complexité : $O(n + e)$ où n est le nombre de noeuds et e est le nombre d'arcs.

4.2.1 Pseudocode

Algorithm *HubFinder*

Function *findallhubs*

Data: communitiesingraph

for *each community in communitiesingraph* **do**

for *each node in community* **do**

if *node not visited* **then**

 explorehub(community, node)

end

end

end

Function *explorehub*

Data: community,current

Result: List of hubs

timelist[current] = visitedtime

lowesttime[current]=visitedtime

visited[current]=True

visitedtime++

for *each adjacent of current node* **do**

if *adjacent not visited* **then**

 predecessor[adjacent] = current

 successors++

 call explorehub(adjacent)

 lowest time of current = min(lowest time of adjacent,

if *current is root and successors ≥ 1 or !root and !backedge* **then**

if *current not in List of hubs* **then**

 add current to List of hubs

end

end

end

else if *adjacent is not parent of current* **then**

 lowest child time=min(lowest child time, adjacent discovery time)

end

end

4.3 Exemple

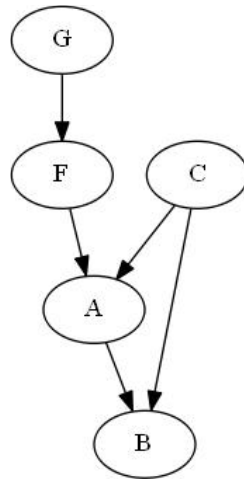


FIGURE 5 – Cette image montre un graphe connexe.

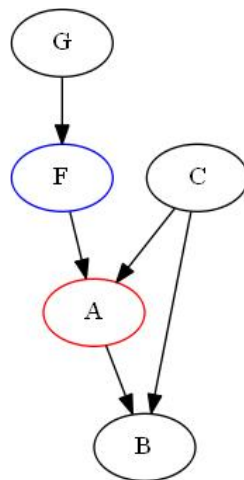


FIGURE 6 – Ce graphe présente les points d'articulation A et F .

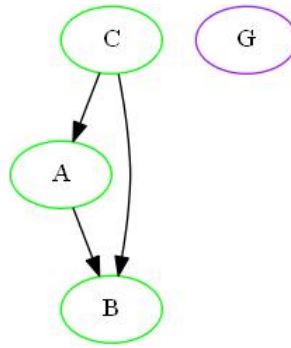


FIGURE 7 – Le point F divise le graphe en 2 communautés : CAB et G

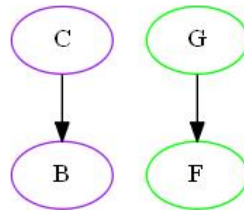


FIGURE 8 – Le point A divise le graphe en 2 communautés : CB et GF

5 Travail en Equipe

Repartition du travail :

- Simplification de dettes : Pavlo Polinetskyi
- Identification des communautés : Grigore Antoniu
- Identification de Hubs Sociaux : Grigore Antoniu
- Refactorisation du code finale : Pavlo Polinetskyi
- Ecriture des tests : Pavlo Polinetskyi
- Rapport : Ensemble

Difficultés rencontrées :

- Synchronisation du code (Appréhension de Github)
- Adaptation des algorithmes existants pour la résolution du problème posé
- Ecriture d'un rapport complet en équipe en LaTeX

6 Sources

- Algorithme de Tarjan : Syllabus
- Algorithme de Johnson : http://people.cs.vt.edu/~gback/ICPCHandbook/book/copiesfromweb/circuits_johnson.pdf
- Algorithme de Communauté (Parcours en profondeur) : Syllabus
- Algorithme de Hubs Sociaux (Point d'articulation) : Syllabus

7 Tests

7.1 OurTests.py

Fichier à exécuter pour lancer les tests prédéfinis par nous pour chaque algorithme. Voici les représentations graphiques des graphes utilisés :

7.1.1 Tests de simplification de dettes

Test 1 :

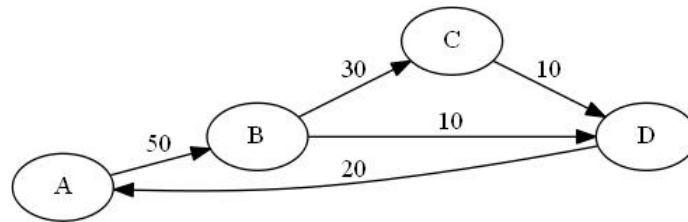


FIGURE 9 – Avant Simplification

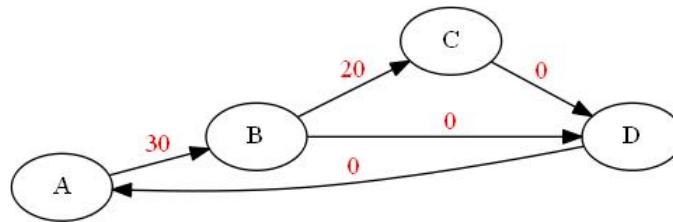


FIGURE 10 – Après Simplification

Test 2 :

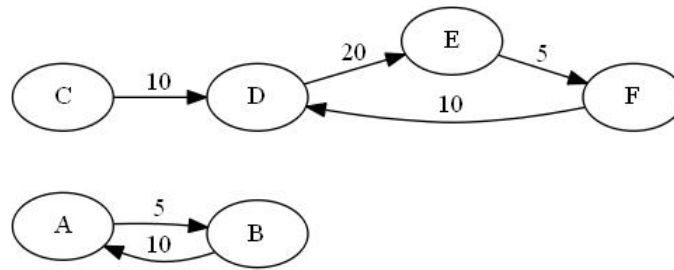


FIGURE 11 – Avant Simplification

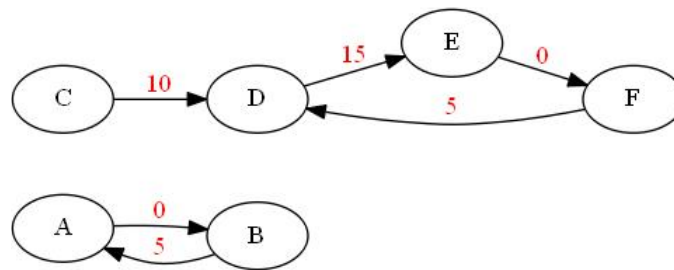


FIGURE 12 – Après Simplification

7.1.2 Test d'identification de communautés

Test 1 :

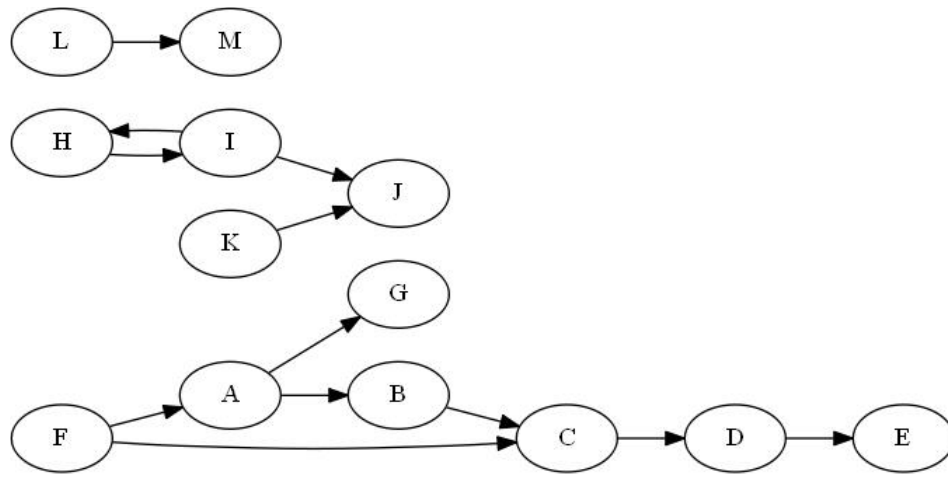


FIGURE 13 – Résultat attendu : $[[G, A, B, C, D, E, F], [I, H, J, K], [M, L]]$

7.1.3 Test d'identification de Hubs Sociaux

Test 1 :

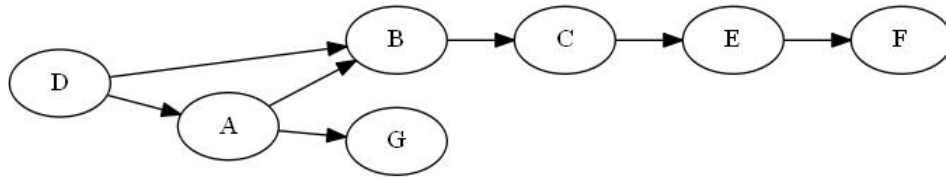


FIGURE 14 – Le graphe

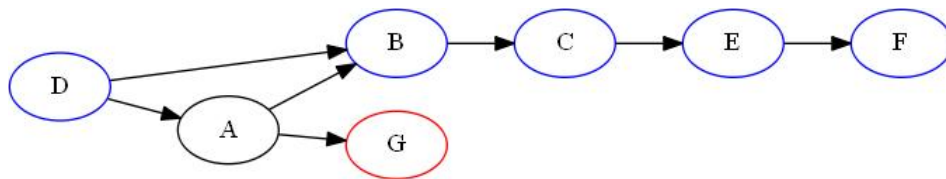


FIGURE 15 – Division en le point A

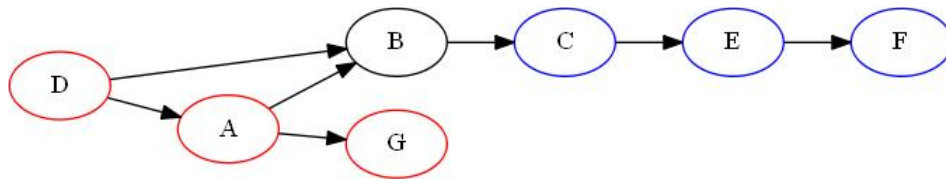


FIGURE 16 – Division en le point B

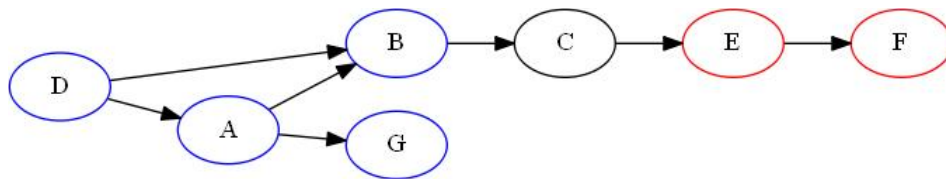


FIGURE 17 – Division en le point C

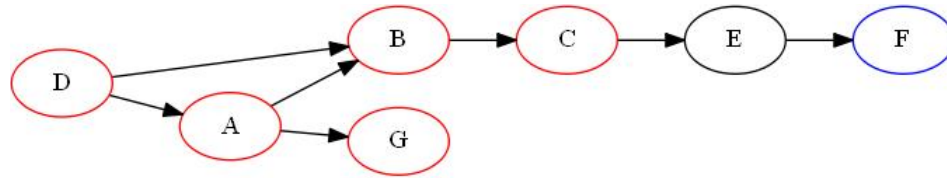


FIGURE 18 – Division en le point E

Alors les résultats attendus sont :

- Pour K=1 : [E, C, A, B]
- Pour K=2 : [C, B]
- Pour K=3 : [B]
- Pour K=4 : []

7.2 YourTester.py

Afin de tester les algorithmes avec vos propres tests, lancez le fichier YourTester.py. Voici un exemple d'utilisation :

python3 YourTester.py inputfile.txt -a -k 9

YourTester.py utilise la librairie argparse, alors pour afficher plus d'informations il suffit juste de taper :

python3 YourTester.py -h

Voici la liste des arguments possibles :

- -k chiffre
- -simplify
- -communities
- -hubs
- -a -all