COMPTE RENDU TRAVAUX PRATIQUES

GRAPHES

SERIE N°1

Test de connexité d'un réseau critique

A l'attention de :

M.OURIACHI Khadir

Réalisé Par :

OLAZAGAZTI Lucas

SERRANO Pierre

I) Position du problème

Lors des phases de conception et surtout de test d'un réseau de communication dans les domaines critiques tels que la sécurité, la défense ou la finance, il est nécessaire de garantir certaines fonctionnalités. Notamment, il est primordial de garantir une communication depuis n'importe quel nœud du réseau vers n'importe quel autre pour certaines parties du réseau suivant une configuration utilisée. Ceci est pris en charge par l'ingénieur chargé de la conception du réseau.

Dans une partie du réseau, garantir une communication entre deux nœuds revient à étudier la connexité de ce sous-réseau et donc à l'analyse de la connexité du modèle de graphe. Ainsi, on peut ramener ce problème à un problème de recherche de composantes fortement connexes dans un graphe.

Plusieurs outils ont été utilisés pour l'étude du graphe :

- Il existe un algorithme, dit Kosaraju-Sharir, qui recherche toutes les composantes fortement connexe d'un graphe. L'analyse du graphe reposera donc sur cet algorithme.
- J'ai décide de développer les différents programmes aidant à l'analyse du graphe en Java, à l'aide de l'environnement de développement intégré Netbeans.
- J'ai utilisé la librairie GraphStream afin de visualiser des graphes.

II) Réalisation

Voici la configuration du modèle de graphe pour le cas étudié :
La décision a été prise d'utiliser un graphe orienté G pour modéliser le réseau. De plus G = (S, A) où

- S est l'ensemble des sommets $s \in S$ représentant chacun un nœud du réseau
- A est l'ensemble des arcs $(s_i, s_j) \in A$ représentant chacun une connexion entre le nœud représenté par son extrémité initiale s_i et le nœud représenté par son extrémité terminale s_j.

Étudier la connexité de certaines parties du réseau (communication entre deux nœuds) revient à chercher les différentes composantes fortement connexes du graphe représentant ce réseau.

Je me suis donc appuyé sur l'algorithme Kosaraju-Sharir, qui recherche toutes les composantes fortement connexe d'un graphe. Cet algorithme consiste à parcourir une première fois en profondeur le graphe, ce qui donne un ordre entre les sommets.Ensuite, il faut parcourir une deuxième fois en profondeur le graphe dual, en explorant les sommets dans l'ordre inverse de l'ordre donné par le premier parcours. Ce deuxième parcours produit des arbres qui sont les composantes fortement connexes du graphe. Il existe différentes solutions pour implémenter l'algorithme Kosaraju-Sharir. Lorsque le graphe est donné sous forme de liste d'adjacence, l'algorithme a une complexité linéaire qui dépend du nombre de sommets et d'arcs du graphe. Ainsi, la complexité est en O(|S|+|A|). J'ai retenu la solution suivante :

La classe Graphe comporte deux attributs :

- un entier *S* correspondant au nombre de sommets du graphe
- une liste chaînée *adj*, où chaque cellule est une liste chaînée d'entier, correspondant à la liste d'adjacence du graphe

De plus, plusieurs méthodes sont implémentées :

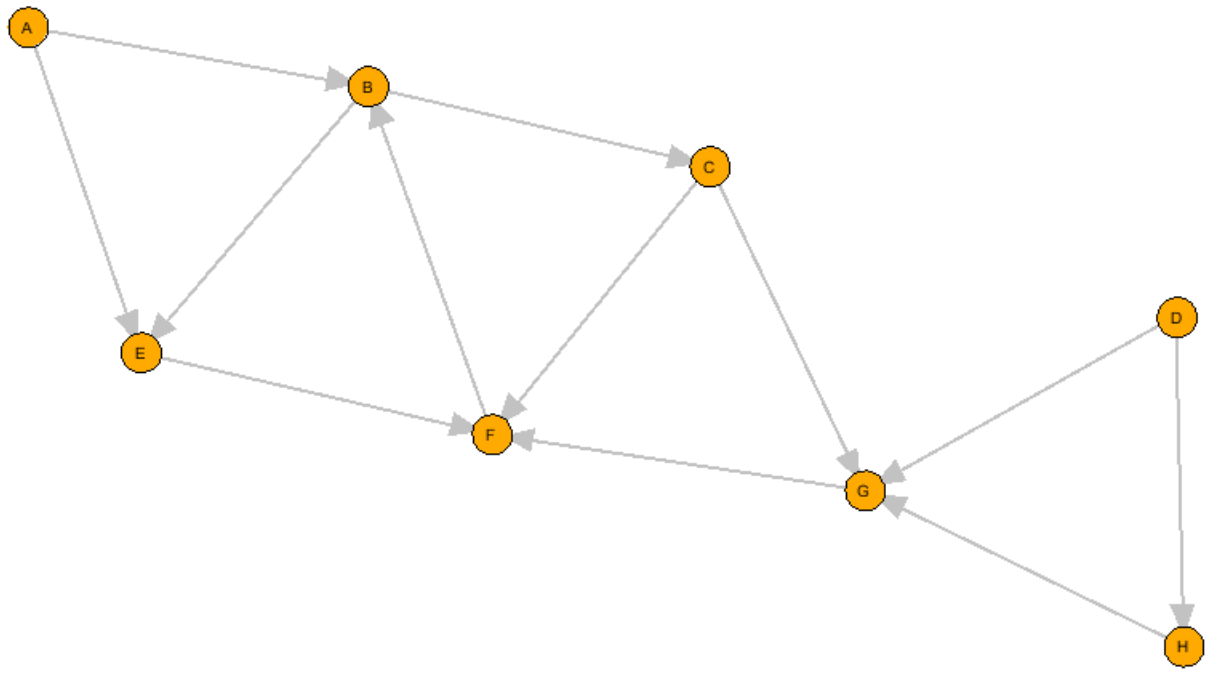
- le constructeur Graphe(entier) qui initialise le nombre de sommets et la liste d'adjacence du graphe,

- la méthode ajoutArc(entier S_i, entier S_j) qui ajoute le nouvel arc $S_i \rightarrow S_j$ au graphe,
- la méthode getDual() qui retourne le graphe dual $G^t(S, A^t)$ du graphe G,
- la méthode affichPPS(entier S, estVisite[booleen]) qui affiche le parcours en profondeur a partir d'un sommet du graphe,
- la méthode rangerSommets(entier S, estVisite[booleen], Stack) qui est une fonction récursive qui permet de ranger dans un stack les sommets du graphe s'ils ont pas déjà été rangés,
- la méthode afficherCFC() qui trouve et qui affiche les composantes fortement connexes du graphe,
- la methode toGraphStream qui me permet d'afficher dans une fenêtre le graphe. Les nœuds sont vert clair si c'est un graphe dual, orange sinon.

Ainsi, il me suffit de créer un graphe pour pouvoir le manipuler. J'ai utilisé comme exemple le graphe G tel que G=(S,A) avec

- S={A, B, C, D, E, F, G, H}
- A={(A→B), (A→E), (B→C), (B→E), (C→F), (C→G), (D→G), (D→H), (E→F), (F→B), (G→F), (H→G)}

On peut visualiser le graphe G ainsi



Ma classe Graphe implémente un graphe ayant des sommets qui ont des noms « d'entiers » partant de 0. Ainsi, pour le graphe G, on a :

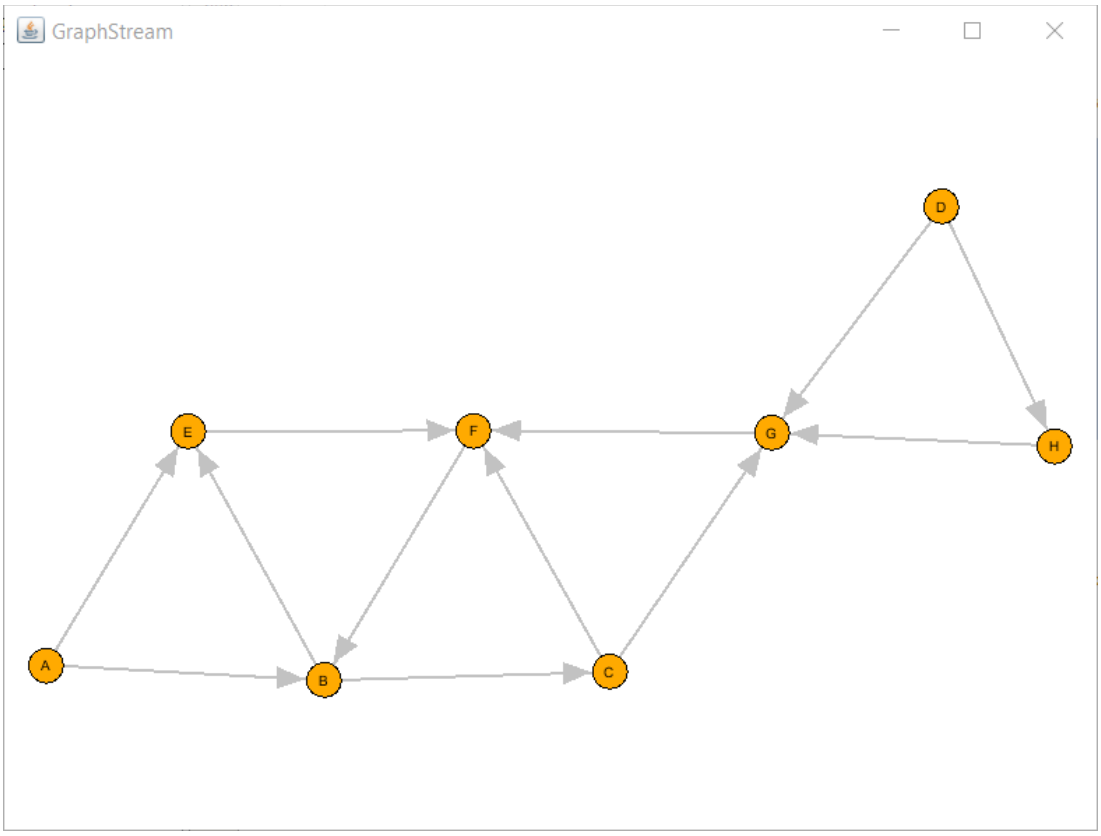
- A prend la valeur 0,
- B prend la valeur 1,
- C prend la valeur 2,
- D prend la valeur 3,
- E prend la valeur 4,
- F prend la valeur 5,
- G prend la valeur 6,
- H prend la valeur 7.

Voici le code correspondant à la création du graphe G :

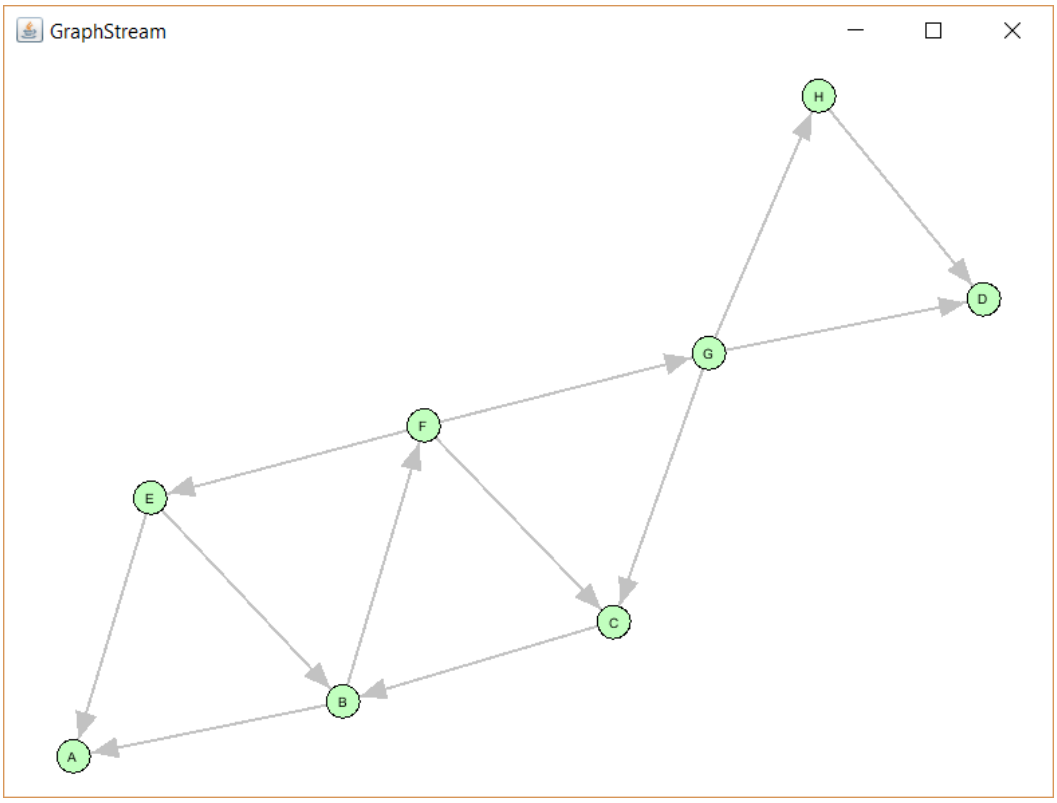
```
Graphe g = new Graphe(8);
g.ajoutArc(0, 1);
g.ajoutArc(0, 4);
g.ajoutArc(1, 2);
g.ajoutArc(1, 4);
g.ajoutArc(2, 5);
g.ajoutArc(2, 6);
g.ajoutArc(3, 6);
g.ajoutArc(3, 7);
g.ajoutArc(4, 5);
g.ajoutArc(5, 1);
g.ajoutArc(6, 5);
g.ajoutArc(7, 6);
```

Après avoir appelé la fonction afficherCFC(), j'ai obtenu les résultats suivants :

Deux nouvelles fenêtres, une visualisant le graphe G (avec les nœuds orange) et l'autre visualisant le graphe G^t, graphe dual de G (avec les nœuds verts clair).



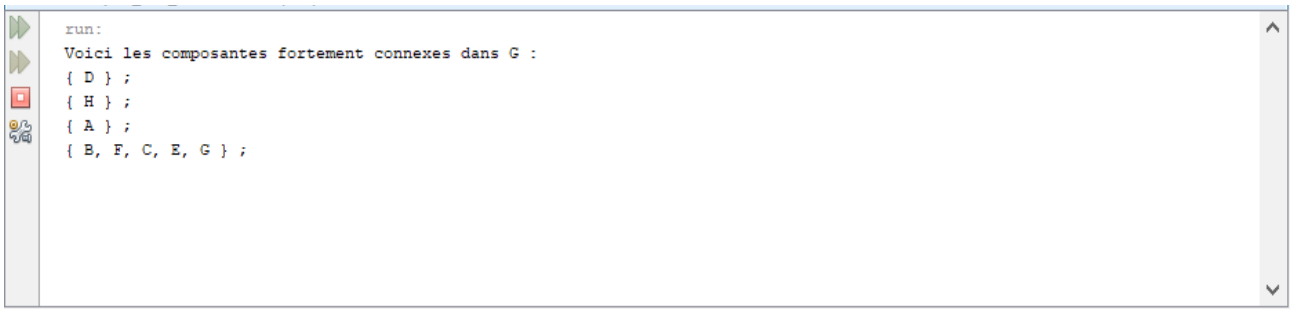
Fenêtre 1 : GraphStream permet de visualiser le graphe G



Fenêtre 2 : GraphStream permet de visualiser le graphe dual G^t

De plus, la console indique ceci :

« Voici les composantes fortement connexes dans G :
{ D } ;
{ H } ;
{ A } ;
{ B, F, C, E, G } ; »

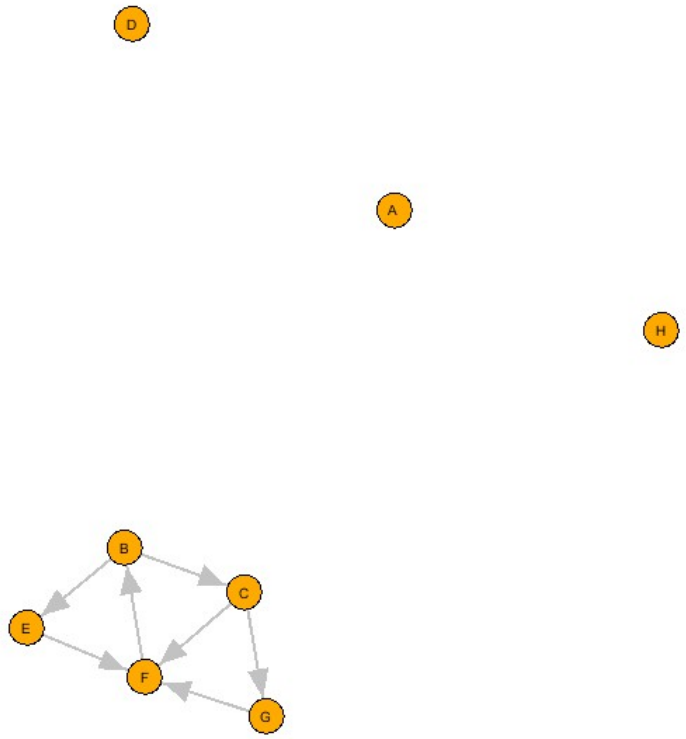


Capture d'écran de la console après avoir lancé le programme

Par conséquent, le graphe G = (S, A) a pour composantes fortement connexes les quatre ensembles de nœuds suivants :

1. { A },
2. { D },
3. { H },
4. { B, F, C, E, G }.

On peut visualiser les composantes fortement connexes ainsi :



On remarque que dans notre exemple, une partie du réseau est fortement connexe et représente environ 60 % du réseau. Il n'y a qu'un pôle où la communication est garantie de n'importe quel nœud du réseau vers n'importe quel autre nœud du réseau.

III) Conclusion

En réalisant cette série de travaux pratiques, j'ai développé mes compétences dans la programmation de modèles et d'algorithmes sur les graphes, j'ai appris à utiliser la librairie GraphStream qui permet de visualiser des graphes en Java et j'ai appris à l'incorporer dans d'autres programmes que ne reconnaît pas GraphStream sans traduction.

J'ai aussi pu voir concrètement qu'il est primordial de savoir déterminer les composantes fortement connexes d'un graphe pour étudier la connexité d'un graphe. Ainsi j'ai pris connaissance de l'algorithme de Kosaraju-Sharir qui permet de déterminer les différentes composantes fortement connexes d'un graphe et j'ai pu implémenter les parcours en profondeur de graphe notamment.

En déterminant les composantes fortement connexes, on peut en déduire la connexité d'un graphe, et lorsque le domaine d'application l'exige, la réponse à la question : « est-ce que les nœuds d'un réseau peuvent communiquer à n'importe quel nœud du même réseau, au moins pour une partie du réseau »