Théorie des Graphes
Projet noté

MADANI Abdenour
TRIOLET Hugo

Licence 3
2021 - 2022

Table des matières

1 Introduction **2**
1.1 Objectifs 2
1.2 Définitions 2
1.3 Résumé de notre approche 3

2 Exercices **3**
2.1 Exercice 1 3
2.2 Exercice 2 4
2.3 Exercice 3 6
2.4 Exercice 4 7

3 Exemples d'utilisation du code **7**
3.1 Comment utiliser le code 7

4 Références **9**

1 Introduction

1.1 Objectifs

Les objectifs de ce TPs sont :

- Orienter un graphe non-orienté en un graphe fortement connexe,
- Décomposer en graphe en chaînes (selon Schmidt[2]),
- Établir si un graphe est 2-connexe ou 2-arête,
- Calculer les composantes 2-connexes et 2-arêtes-connexes sinon.

On utilisera pour ceci **SageMath** (bibliothèque de fonctions pour Python).

1.2 Définitions

🎓 DÉFINITION

Pont

Arête dont la suppression augmente le nombre de composantes connexes du graphe restant.
(aussi appelé arête déconnectante)

🎓 DÉFINITION

Sommet d'articulation

Sommet dont la suppression augmente le nombre de composantes connexes du graphe restant.
(aussi appelé sommet déconnectant ou noeud d'articulation)

🎓 DÉFINITION

2-connexité (*2-sommet-connexité*)

Un graphe est dit 2-connexe s'il n'admet pas de sommet d'articulation.

🎓 DÉFINITION

2-arête-connexité

Un graphe est dit 2-arête-connexe s'il n'admet pas de pont.

🎓 DÉFINITION

Depth-First Search (DFS)

Parcours en profondeur d'un graphe.

🎓 DÉFINITION

Depth-First Index (DFI)[2]

Date à laquelle le DFS a **débuté** sur un noeud.

🎓 DÉFINITION

Graphe sous-jacent

Pour un graphe orienté, le graphe sous-jacent correspond au graphe avec les mêmes noeuds, mais dont les arêtes ne sont plus orientées. (le graphe sous-jacent est donc non-orienté)

1.3 Résumé de notre approche

On implémente la décomposition en chaînes[2].
Celle-ci nous permet d'obtenir tous les ponts, ainsi que tous les sommets d'articulation du graphe.
On en déduit ensuite si le graphe est 2-connexe, 2-arête-connexe, ou aucun des deux, grâce à l'Algorithme 1 de Schmidt[2].
On calcule ensuite les composantes 2-connexes et 2-arêtes-connexes.

Vis-à-vis du code, nous l'avons documenté à l'aide de la *docstring* de Python, les fonctions se comprennent donc naturellement grâce à celle-ci.

2 Exercices

2.1 Exercice 1

Dans une premier temps il est demandé d'implémenter les algorithmes de calcul de 2-connexité dû à Schmidt[2].
(1) Calculer les composantes 2-connexes d'un graphe.
(2) Calculer les composantes 2-arêtes-connexes d'un graphe.
On commence par effectuer la décomposition en chaînes.

Grâce aux Lemmes 4 et 5[2], on peut identifier les ponts et sommets d'articulation du graphe.

Pour obtenir les composantes 2-connexes :
On prend le graphe original, on le copie.
Ensuite, on fait un traitement différent sur chaque pont, puis sur chaque sommet d'articulation en début de cycle.

Algorithm 1 Composantes 2-sommets-connexes

Entrée : *ponts*, la liste des ponts, ainsi que *chaines* la liste des chaînes, en variable globale
Sortie : Les composantes 2-sommets-connexes

Prenons une copie du graphe du départ, dans laquelle nous allons effectuer nos modifications.

for each pont (u, v) **do**
On supprime l'arête (u, v) du graphe.
On rajoute un noeud u' (si déjà pris, u_2, puis u_3, etc) et de même un noeud v'.
On rajoute une arête (u', v') dans le graphe.

for each sommet u en début de cycle, à partir de C_2 **do**
Le cycle est de la forme : $u, v_1, ..., v_k, u$

On supprime les arêtes (u, v_1) et (u, v_k) du graphe.
On rajoute un noeud u' (si déjà pris, u_2, puis u_3, etc)
On rajoute deux arêts (u', v_1) et (u', v_k) dans le graphe.

SAUF :
Si le sommet est de degré 2 (c'est-à-dire que ses autres arêtes ont été supprimées entre temps).
Dans ce cas, on ne fait rien.

return le graphe des composantes 2-sommets-connexes

Voir fonction `calcule_comp_2_sommet_connexe`.

Pour obtenir les composantes 2-arêtes-connexes :
On prend le graphe original, on supprime ses ponts.
Ensuite, on enlève les noeuds restants de degré 0.
Voir fonction `calcule_comp_2_arete_connexe`.

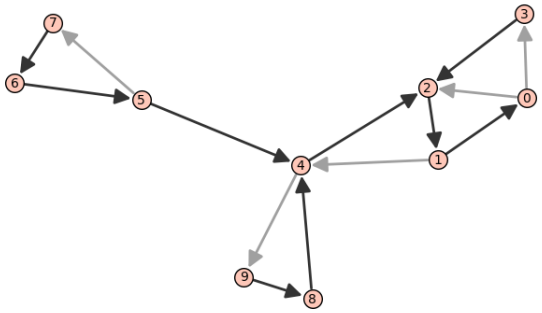
2.2 Exercice 2

Trouvez un orientation en un graphe fortement connexe[2] ou trouvez une arête déconnectante.

On effectue un DFS pour orienter toutes les arêtes, tel que décrit dans la *Figure 1b*[2].
On oriente les arcs de parenté du fils vers le père, et les arcs arrières du père vers le fils.
D'après le théorème de Robbins[1], le graphe orienté obtenu sera fortement connexe si et seulement si celui-ci était 2-arête-connexe.

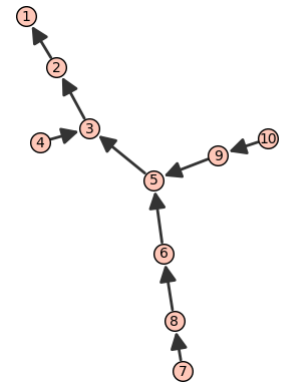
Au sein de notre fonction `parcours_graphe()`, nous avons une sous-fonction `parcours()` qui est une fonction récursive de DFS.
Si jamais le graphe n'est pas connexe, nous faisons appel à la fonction `lance_parcours()` qui lance un DFS sur chaque sommet non-visité.
Durant la fonction `parcours()`, nous allons créer un nouveau graphe qui contiendra, en gris foncé, **l'arbre de parcours en profondeur**, et en gris clair tous les arcs arrières.

> **ATTENTION**
> Ce graphe est appelé `arbre_parcours` dans le code (bien qu'il contienne les arcs arrière).

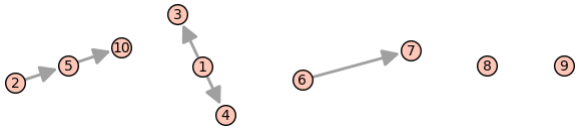


`arbre_parcours`

Pour plus de commodités, nous créons aussi deux graphes à part contenant chacun soit uniquement l'arbre de parcours, soit uniquement les arrières : appelés respectivement `arbre_parcours_uniquement` et `arriere`.



arbre_parcours_uniquement



arriere

Le graphe `arbre_parcours` est donc une orientation en graphe fortement connexe si et seulement si la variable `deux_arete_connexe` renvoyée par la fonction vaut `True`.
Si `deux_arete_connexe` vaut `False`, alors il suffit de regarder la variable `ponts` renvoyée par la fonction (ou `graphe_ponts.edges()`).

2.3 Exercice 3

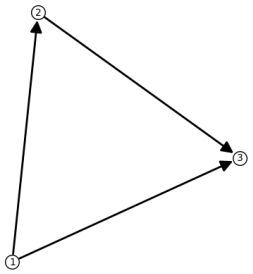
Montrez qu'un graphe est fortement connexe[2] si et seulement si chaque arc est présent dans au moins un circuit de G.
$\Longrightarrow$: On suppose que quelque soit e appartenant à E, e est présent dans au moins un circuit de G. On pose e = x,y, avec x,y appartenant à V.
Puisque e est un chemin du sommet x au sommet et que e est présent dans au moins un circuit de G, cela veut dire que le chemin de y à x existe.
Autrement dit, qq soit e un arc tel que e = (x, y) (resp. (y, x)), alors qq soit x, y appartenant à V, il existe un chemin de x à y (resp. y à x) (donc e) et il existe un chemin de y à x (resp. de x à y). Donc, par définition, cela implique que le graphe est fortement connexe.

$\Longleftarrow$: Par contraposée, on suppose qu'il existe un arc e appartenant à E, tel que e n'apparaisse dans aucun des circuits de G.
Ainsi, si pose e = (x, y) (resp. (y, x)), avec x, y appartenant à V, alors cela veut dire qu'il existe un chemin de x à y (resp. de y à x) mais il n'en existe pas

de y à x (resp. de x à y)
Donc le graphe G n'est pas fortement connexe.

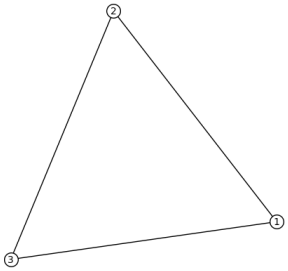
2.4 Exercice 4

Prouvez ou infirmez l'affirmation suivante :
Un graphe est fortement connexe si et seulement si son graphe sous-jacent est 2-arête connexe.
Voici un exemple qui infirme la proposition. Soit ce graphe non-orienté :



Un graphe orienté non fortement connexe

Celui-ci n'est pas fortement connexe, car on ne peut pas accéder à 1 à partir de 3 par exemple. Or voici son graphe sous-jacent :



Son graphe sous-jacent

Celui-ci est bien 2-arête-connexe, pourtant le graphe original n'était pas fortement connexe, cela conclut la preuve.

3 Exemples d'utilisation du code

3.1 Comment utiliser le code

Il faut évidemment disposer de SageMath pour exécuter le code. Par exemple, en l'installant sur le site officiel, et en l'exécutant dans un NoteBook tel que Jupyter.
On commence par exécuter le code contenant toutes les fonctions ainsi que

quelques variables globales.
On crée un graphe, puis on utilise la fonction `affiche_infos()`

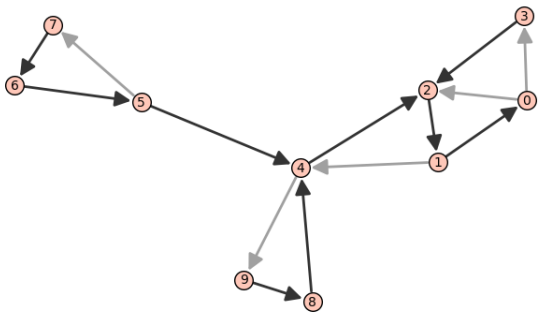
```
graphe_exemple_schmidt = Graph()
graphe_exemple_schmidt.add_edges([(1, 2), (1, 3), (1, 4), (2, 3),
    (2, 5), (3, 4), (3, 5), (5, 6), (5, 9), (5, 10), (6, 7), (6, 8)
    , (7, 8), (9, 10)])

infos = affiche_infos(graphe_exemple_schmidt)
```

```
Entrée [43]: graphe_exemple_schmidt = Graph()
             graphe_exemple_schmidt.add_edges([(1, 2), (1, 3), (1, 4), (2, 3), (2, 5), (3, 4), (3, 5),
                             (5, 6), (5, 9), (5, 10), (6, 7), (6, 8), (7, 8), (9, 10)])

             infos = affiche_infos(graphe_exemple_schmidt)

           sommets d'articulation : {5, 6}

           ponts : [(5, 6, None)]

           chaines : [[1, 3, 2, 1], [1, 4, 3], [2, 5, 3], [5, 10, 9, 5], [6, 7, 8, 6]]
           Le graphe est connexe
```

Exemple d'utilisation 1

Dans la fonction `parcours_graphe()`, à la fin, on peut voir un dictionnaire `informations`. On peut lui extraire les informations que l'on désire.
Pour afficher un arbre de parcours avec les couleurs du rapport, on utilise `plot_couleur`.

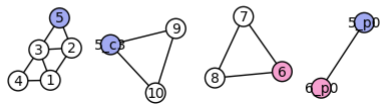
```
plot_couleur(infos['arbre_parcours'])
```



Affichage des composantes 2-connexes avec les sommets d'articulation séparés de la même couleur :

```
affiche_comp_2_sommet_connexe(graphe_exemple_schmidt)
```

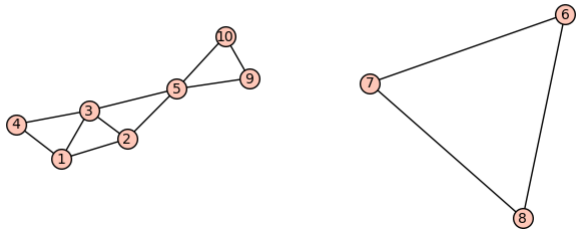
```
Entrée [78]: affiche_comp_2_sommet_connexe(graphe_exemple_schmidt)
```



Affichage des composantes 2-arêtes-connexes

```
plot_couleur(infos['composantes_2_arete_connexe'])
```

```
Entrée [79]: plot_couleur(infos['composantes_2_arete_connexe'])
Out[79]:
```



4 Références

References

[1] Herbert Robbins, *A theorem on graphs, with an application to a problem on traffic control*, American Mathematical Monthly 46, 281–283.

[2] Jens M. Schmidt, *A Simple Test on 2-Vertex- and 2-Edge-Connectivity*, Inf. Process. Lett. **113** (2013), no. 7, 241–244.