LINMA1691 : Théorie des graphes
Devoir 1 : Composantes Fortement Connexes

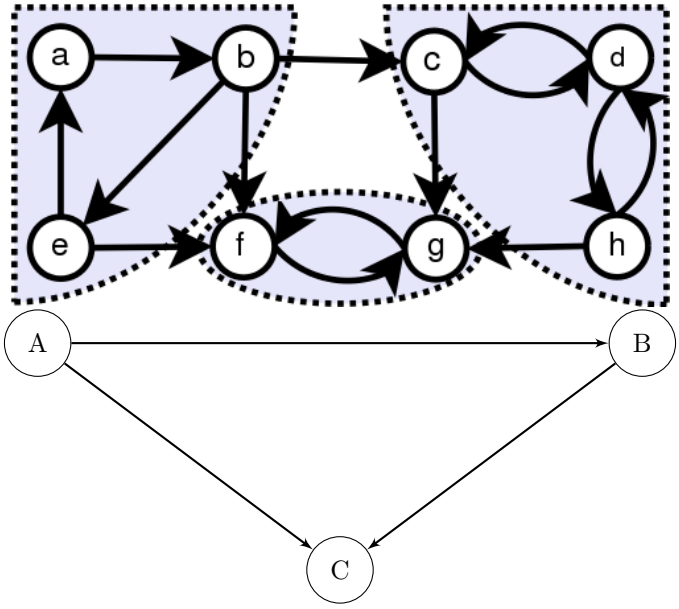
Dans ce devoir nous explorons une notion liée à la connexité pour les graphes dirigés: les composantes fortement connexes (ou strongly connected components aka SCCs). Votre objectif est de résoudre le problème qui suit. Pour vous y aider, on explique ce que sont les SCCs et référons à un algorithme efficace permettant de les calculer.

1 Composantes Fortement Connexes (SCCs)

Dans un graphe dirigé une composante fortement connexe est un ensemble de noeuds tel que pour toutes paires de noeud u et v dans l'ensemble il existe un chemin de u à v et de v à u, de plus l'ensemble de noeuds est maximal pour cette propriété.

Notez que cette relation entre noeuds u et v est réflexive, transitive et symétrique. On peut donc à partir de tout graphe déduire une partition de ses noeuds en SCCs. Et de là en déduire un nouveau graphe dirigé où chaque noeud est une composante fortement connexe dans le graphe original.

Un exemple[1] avec une graphe orienté et sa décomposition en SCCs puis le graphe des SCCs.



Un algorithme efficace parmi d'autres pour calculer ces SCCs est celui de Kosaraju. De nombreuses références librement accessibles décrivent cet algorithme. Aussi, vous êtes libre d'utiliser un autre algorithme pour calculer les SCCs, ou même de ne pas utiliser le concept de SCC du tout si vous ne le jugez pas nécessaire pour résoudre le problème de la section suivante.

[1] `https://fr.wikipedia.org/wiki/Composante_fortement_connexe`

Attention, l'algorithme de Kosaraju est souvent décrit et implémenté via des appels récursifs. Malheureusement, ceux-ci s'avèrent inefficaces[2]. Il faudra donc corriger ce défaut dans vos implémentations.

2 Rumeurs

Contexte

Vous voulez disperser dans LLN la rumeur qu'une tyrolienne va être construite entre la place des sciences et la place de l'université. De plus, vous voulez disperser cette rumeur en la donnant de manière directe à un minimum de kots.

Pour ce faire vous disposez d'informations sur quels kots partagent les rumeurs avec quels kots. Notez que cette relation n'est pas forcément symétrique.

Par exemple, il se pourrait que si le Kotangente apprend la rumeur alors le Babbelkot et l'Akapella l'apprendront aussi, mais que l'Akapella ne partage la rumeur qu'avec le Babbelkot et pas avec le Kotangente. Dans cet exemple à 3 kots, une solution optimale est de donner la rumeur au Kotangente qui la partagera avec les deux autres.

On vous demande d'écrire une fonction *solve* qui détermine le nombre minimal de kots avec qui il faut partager directement la rumeur, de façon à ce que tous les kots en aient finalement vent.

Input de la fonction *solve*

Une liste d'adjacence d'un graphe dirigé décrivant la dispersion des rumeurs. La longueur n de la liste d'adjacence est le nombre de kots. Le kot $0 \leq x < n$ partage la rumeur au kot $0 \leq y < n$ ssi y est dans la liste des voisins de x (ADJ[X]).

Output de la fonction *solve*

Un entier donnant le nombre minimal de kots à qui partager la rumeur pour que tous les kots la connaissent.

Jeux de tests

Pour vous aider à tester votre algorithme, nous fournissons un fichier *checker.py* qui appelle votre algorithme (tel qu'implémenté par *solve*) sur un jeu de tests fournis qui est *représentatif* de ceux fait dans Inginious.

Les fichiers de tests sont de la forme suivante: La première ligne contient un entier k donnant le nombre d'inputs qui suivent. Ensuite viennent k inputs, constituant chacun un test pour votre implémentation. Chaque input commence par une ligne de 2 entiers donnant le nombre de kots et le nombre de lignes restantes pour décrire ce test. Les lignes restantes ont elles-aussi 2 entiers x, y décrivant le fait que le kot x partage la rumeur au kot y. (Note: Dans les fichiers de tests, on numérote les kots de 1 à n contrairement à l'input de la fonction *solve* où la correction -1 est déjà faîte.)

L'organisation des tests est donc déjà orchestrée dans les différents fichiers fournis, vous n'avez plus qu'à compléter la fonction *solve* dans le fichier *solve_ hw.py*.

[2]Une implémentation récursive de cet algorithme augmente excessivement la taille de la pile d'exécution.

Consignes

Vous devez soumettre votre fichier solution complété *solve_hw.py* sur l'activité Inginious[3]. Votre solution doit avoir une complexité linéaire en le nombre d'arêtes des graphes d'entrées pour passer les tests. (Note: Pas de texte "print" dans le fichier.)

Le langage de programmation est **Python 3** (version 3.5).

Deadline : vendredi 15 octobre à 22h.

Questions : Au tp, sur le forum moodle, ou par mail brieuc.pinon@uclouvain.be.

[3]https://inginious.info.ucl.ac.be