Une ville, un réseau, la lune, un train et une souris

1 Connexité (Connectivity)

Exercice 1.1 (Algernon et le labyrinthe)

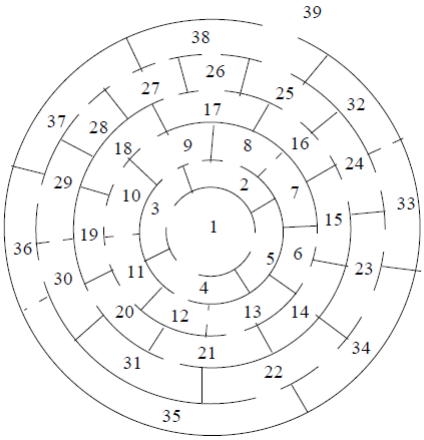


Figure 1 – Un des labyrinthes d'Algernon

Algernon, petite souris blanche de laboratoire, apprend à sortir de labyrinthes.

1. Algernon est placée au centre du labyrinthe. Est-ce qu'elle pourra en sortir ? Est-ce qu'elle a plusieurs solutions ?
2. Pour les essais suivants, Algernon est placée à un nouvel endroit du labyrinthe à chaque fois. Peut-elle sortir du labyrinthe à tous les coups ?
3. Algernon est maline, elle a réussi à trouver un chemin depuis le centre du nouveau labyrinthe et s'en souvient. Les chercheurs ferment donc toutes les portes par lesquelles elle est passée. Peut-elle encore sortir du labyrinthe depuis le centre ?

Exercice 1.2 (Réseau de routeurs)

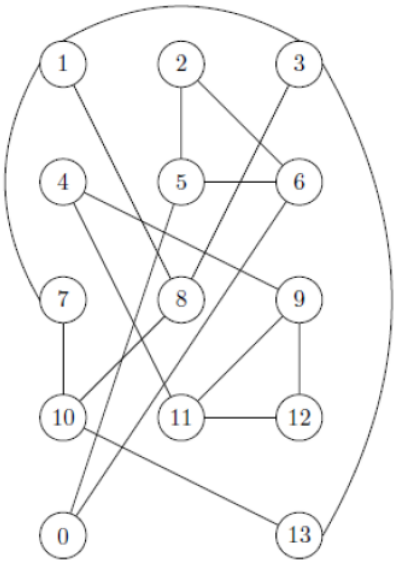


Figure 2 – Réseau de routeurs

Le fournisseur de service Internet (Internet Service Provider : ISP) *KrisNet* doit mettre en place son réseau de routeurs. Chaque routeur se trouve dans un *point d'interconnexion* pour avoir accès aux

routeurs des autres ISP. *KrisNet* désire établir son propre réseau (faire passer du trafic au travers du réseau d'un autre ISP étant facturé) par des liaisons *point à point* permettant de relier un routeur avec un autre. La direction de *KrisNet* a obtenu la solution présentée en figure 2.

1. L'architecte réseau doit vérifier que les connexions proposées permettent réellement de relier tous les routeurs entre eux.
   (a) Quelle propriété de la théorie des graphes ce réseau doit-il respecter ?
   (b) Rappeler toutes les définitions en relation avec cette propriété.
2. Graphe connexe (Connected graph) :
   (a) Donner le principe de vérification de la propriété de la question 1.
   (b) Appliquer ce principe au réseau de *KrisNet*. Ce réseau permet-il de connecter tous les routeurs ?
3. Composantes connexes (Connected components) :
   (a) Rappeler la définition de *composante connexe*.
   (b) Adapter le principe de l'algorithme précédent pour qu'il donne les composantes connexes d'un graphe et leur nombre.
   (c) Comment représenter les composantes connexes ?
4. Nouvelles liaisons :
   L'architecte réseau veut maintenant rajouter les liaisons manquantes pour éviter d'avoir des sous-réseaux qui ne seraient pas reliés aux autres. Comment déterminer les liaisons à ajouter pour rendre le graphe connexe ?

Exercice 1.3 (Indicateurs globaux de connexité – *Midterm - 2018*)

Les indicateurs globaux de connexité mesurent le degré de fragmentation d'un graphe en composantes connexes séparées les unes des autres. Soient N le nombre de sommets d'un graphe, k son nombre de composantes connexes et $n_1, \cdots, n_k$ le nombre de sommets de chacune des composantes connexes ($n_1 + n_2 + ... + n_k = N$). On peut définir deux indices de connexité dont les valeurs sont comprises entre 0 (graphe vide) et 1 (graphe connexe).

- Indice de connexité simple : $IC_1 = (N - k)/(N - 1)$
- Indice de connexité pondéré : $IC_2 = (n_1^2 + n_2^2 + ... + n_k^2)/(N^2)$

Par exemple sur le graphe de la figure 3 :

- $IC_1 = (14 - 3)/(14 - 1) = 0,9166..$
- $IC_2 = (4^2 + 4^2 + 6^2)/14^2 = 0,3469..$

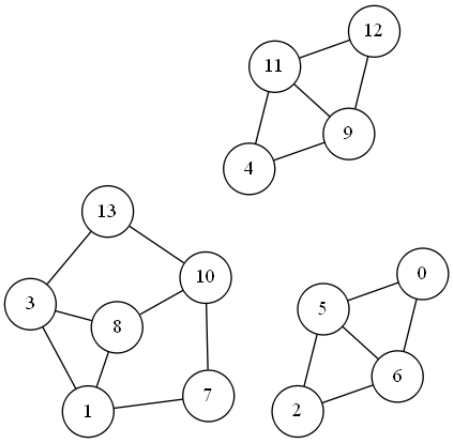


Figure 3 – Graphe à 3 composantes

1. En terme de probabilités, quelle information sur le graphe nous donne l'indice de connexité pondéré ?
2. Écrire les fonctions permettant de calculer les deux indices de connexités (simple et pondéré) d'un graphe (représenté par listes d'adjacences).

Exercice 1.4 (L'union fait la force)

KrisNet ne connaît pas le graphe : les seules informations qu'il a sont les routeurs (numérotés) et les connections existantes.

1. Comment savoir si deux routeurs sont interconnectés ?
2. Écrire une fonction qui donne la liste des connexions à ajouter pour relier tous les routeurs entre eux (sans construire de graphe).

Les exercices qui suivent sont tirés du contrôle de 2017, dans lequel `build` *qui applique l'union-find à une liste d'arêtes et retourne le vecteur de père était donné !* Voir `S4C4_2017_cor.py`

Exercice 1.5 (Journey To The Moon – *Midterm 2017*)

Les états membres de l'UN désirent envoyer 2 personnes sur la lune. Conformément à leur principe d'unité mondiale, il leur faut trouver deux astronautes de pays différents.

Il y a N astronautes entrainés, numérotés de 0 à $N-1$. Mais les responsables de la mission n'ont pas reçu d'informations sur la citoyenneté de chaque astronaute. La seule information dont ils disposent est que certaines paires d'astronautes appartiennent au même pays.

Votre tâche consiste à calculer de combien de façons ils peuvent choisir une paire d'astronautes appartenant à différents pays. Vous disposez de suffisamment de couples pour vous permettre d'identifier les groupes d'astronautes. Par exemple, si 1, 2, 3 sont des astronautes du même pays ; il suffit de mentionner que (1, 2) et (2, 3) sont des paires d'astronautes du même pays sans fournir d'informations sur une troisième paire (1, 3).

L est une liste de couples (A, B) tels que A et B, tous deux dans $[0, N-1]$, sont des astronautes du même pays.

Écrire la fonction `Moon(`N`,` L`)` qui calcule le nombre de paires différentes d'astronautes pouvant être sélectionnées.

Exercice 1.6 (Union-Find – *Midterm 2017*)

Soit un graphe $G =< S, A >$. À partir de n le nombre de sommets, et L la liste des arêtes de G (liste de couples de sommets), écrire la fonction `CCFromEdges(`n`,` L`)` qui retourne le couple (k, cc), avec k le nombre de composantes connexes de G, et cc le vecteur (une liste en Python) des composantes connexes de G.

Exercice 1.7 (Warshall – *Midterm 2017*)

Rappels :

- On appelle *fermeture transitive* d'un graphe non orienté G défini par le couple $< S, A >$, le graphe G^* défini par le couple $< S, A* >$ tel que pour toutes paires de sommets $x, y \in S$, il existe une arête $\{x, y\}$ dans G^* si-et-seulement-si il existe une chaîne entre x et y dans G.
- L'algorithme de Warshall calcule la matrice d'adjacence de la fermeture transitive d'un graphe.

Écrire la fonction `CCFromWarshall(`M`)` qui, à partir de M, matrice résultat de l'algorithme de Warshall appliquée au graphe G, retourne le couple (k, cc), avec k le nombre de composantes connexes de G, et cc le vecteur (une liste en Python) des composantes connexes de G.

Exercice 1.8 (Minimiser les liaisons)

Louer une liaison *point à point* coûte cher, le comptable demande donc à l'architecte réseau de trouver une solution garantissant que tous les routeurs sont reliés, mais en utilisant le minimum possible de liaisons.

1. Comment peut-on satisfaire le comptable : comment obtenir un réseau où tous les routeurs sont reliés en utilisant un minimum de liaisons ?
2. Un tel graphe est un *arbre*.
   (a) Que se passe-t'il si on ajoute une arête quelconque à un *arbre* ?

(b) Et si on lui enlève une arête quelconque ?

(c) Quelles sont les trois propriétés d'un graphe qui est un *arbre* ?

3. Algorithme :

(a) Est-il nécessaire de vérifier les trois propriétés de la question précédente pour déterminer si un graphe est un *arbre* ?

(b) En déduire les différents méthodes pour vérifier qu'un réseau est bien interconnecté et est constitué d'un nombre minimum de liaisons.

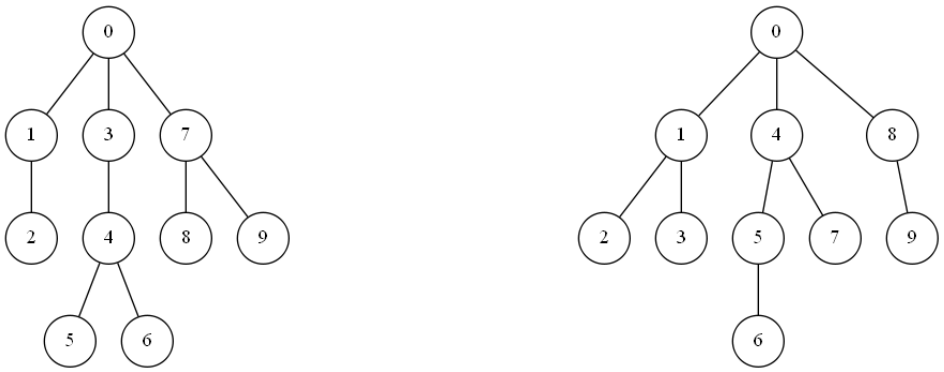
(c) Écrire une fonction qui détermine si un graphe est un arbre.

Exercice 1.9 (Bonus : Even Tree – *Midterm 2017*)

Nous travaillons ici avec des *arbres* (graphes simples connexes sans cycles). Le but est de trouver le nombre **maximum** d'arêtes que l'on peut retirer d'un arbre pour obtenir une forêt telle que chaque composante connexe de la forêt contienne un nombre **pair** de sommets.

Note : Le graphe donné est tel qu'il peut toujours être décomposé en composantes contenant un nombre pair de sommets.

Quelles arêtes peuvent être supprimées dans les arbres suivants ?



Dans les deux cas, le résultat est une forêt dont les composantes connexes sont de tailles paires.

Note : cela fonctionne quelque soit le sommet de départ.

Écrire la fonction `evenTree(`G`)` qui calcule le nombre maximum d'arêtes qui peuvent être enlevées de G selon les spécifications précédentes.

Une ville, un réseau, la lune, un train et une souris

2 Forte connexité (Strong Connectivity)

Exercice 2.1 (Γ^+, Γ^-)

On considère l'algorithme suivant :

```
mark ⊕ and ⊖ some vertex x of the digraph
 while feasible  do
     mark ⊕ every successor of a vertex marked ⊕
     mark ⊖ every predecessor of a vertex marked ⊖
 end while
```

1. Quelle est la complexité de cet algorithme ?
2. Que représente l'ensemble des sommets qui sont marqués à la fois ⊕ et ⊖ ?
3. Comment utiliser cet algorithme pour déterminer si un graphe orienté est fortement connexe ? Ou, s'il ne l'est pas, pour déterminer ses différentes composantes fortement connexes ?

Exercice 2.2 (Circulation à sens unique)

Monsieur Voie est chargé d'installer des sens uniques dans tout un quartier afin que la circulation soit plus fluide (les rues étant trop étroites, le stationnement et la circulation posent un problème). Voici le plan avec le sens de circulation de chaque rue. M. Voie se demande si on peut vraiment se déplacer partout dans le quartier sans en sortir.

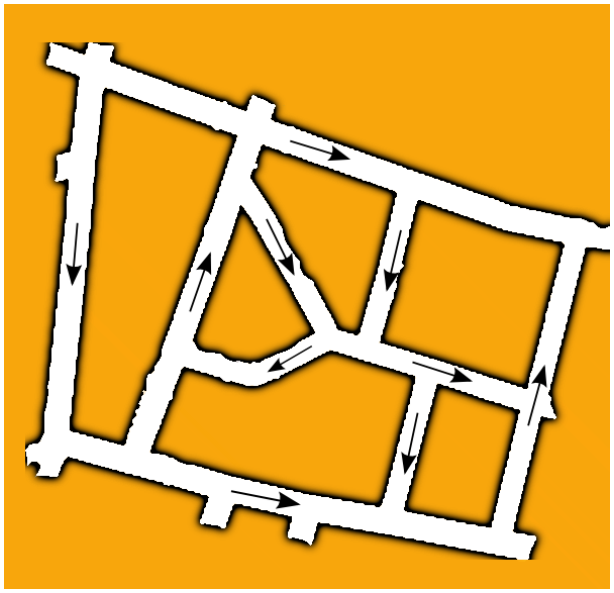


Figure 4 – Plan du quartier

1. Graphe fortement connexe (Strongly connected graph)
   (a) Modéliser le plan sous forme de graphe.
   (b) Le plan de M. Voie est-il fortement connexe ?
   (c) Quelles sont les rues qui doivent être mises en double sens ?

2. **Méthode 1 :** deux parcours
   (a) Qu'observe-t-on lorsque l'on considère le graphe transposé et la propriété que le graphe initial devrait respecter ?
   (b) Donner le principe de construction des composantes fortement connexes utilisant un double parcours et écrire la fonction correspondante.
   (c) Et si on veut uniquement tester la forte connexité du graphe ?

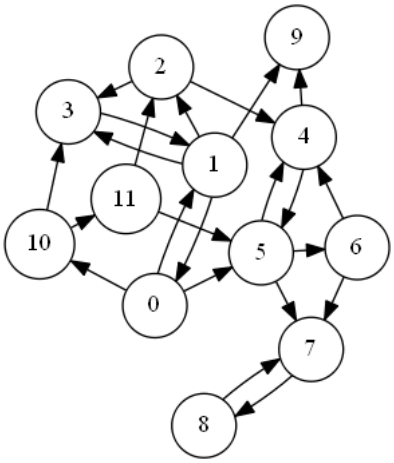


FIGURE 5 – Un autre plan

3. **Méthode 2 :** Tarjan
   (a) Appliquer l'algorithme de "Tarjan" pour trouver les composantes du graphe de la figure 5.
   (b) Écrire la fonction qui détermine les composantes fortement connexes d'un graphe orienté.
   (c) **Midterm S4 - 2018 :** Et si on veut uniquement tester la forte connexité du graphe ?

Exercice 2.3 (Train Électrique)

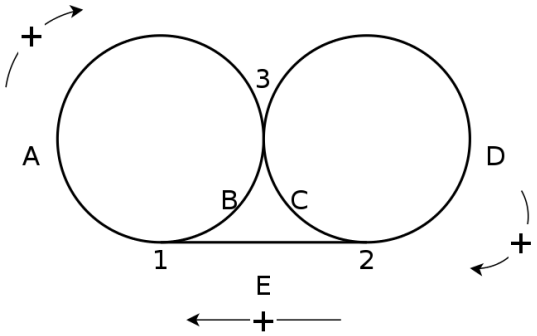


FIGURE 6 – Circuit de train électrique

La figure 6 présente un circuit de train électrique. Le train peut se déplacer soit en marche avant soit en marche arrière. Le circuit est découpé en 5 tronçons (A, B, C, D et E), que l'on peut emprunter dans les deux sens (+ et −) et dispose de 3 aiguillages.

Montrer que si le sens du train est constant (s'il est déterminé par l'alimentation du circuit par exemple) alors, à partir d'un certain temps, le train ne peut plus emprunter le tronçon E.

Faire exercice 2 du contrôle 4 de 2018

3 Pour finir

Exercice 3.1 (Résister aux pannes)

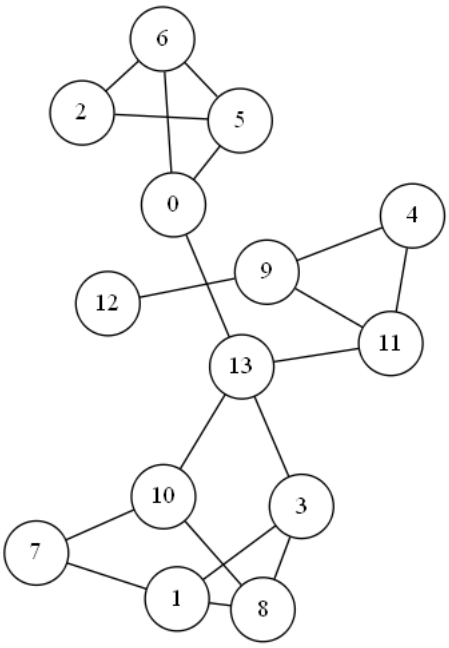


FIGURE 7 – Amélioration du réseau

Bien que minimiser le nombre de liaisons soit important, il convient à notre ISP d'avoir un réseau tolérant aux fautes. En effet, si un routeur se trouve coupé d'un autre, il devra emprunter une route alternative via les connexions externes des routeurs, or le trafic sur ces connexions (trafic de transit) a un coût relativement élevé. On s'intéresse donc à la résistance aux pannes, celles-ci peuvent être de deux sortes : panne d'un routeur et panne d'une liaison.

1. **Panne d'un routeur :**

   (a) Quel problème pose le réseau à nombre minimum de liaisons, en cas de panne de certains routeurs ? Quels sont ces routeurs ?

   (b) En considérant le réseau comme un graphe, comment appelle-t-on un sommet qui pose ce type de problème ?

   (c) Montrer qu'un tel sommet a au moins 2 fils s'il est racine d'une arborescence de parcours.

   (d) Montrer qu'un tel sommet nommé v, s'il n'est pas racine de l'arborescence, possède au moins un fils s tel qu'il n'existe aucun arc arrière partant de s ou d'un descendant de s et arrivant à un ancêtre de v..

   (e) Quelle information doit-on associer à chaque sommet pour représenter la propriété de la question précédente ? Comment calcule-t-on cette information ?

   (f) Donner le principe de détermination d'un tel sommet, en utilisant les réponses aux trois dernières questions.

   (g) Appliquer au graphe de la figure 7.

   (h) Écrire la fonction correspondant à ce principe.

2. **Panne d'une liaison :**

   (a) Quel problème pose le réseau à nombre minimum de liaisons en cas de liaison défectueuse ? En théorie des graphes, comment appelle-t-on une liaison qui pose ce type de problème.

   (b) Montrer qu'une telle liaison n'appartient à aucun cycle élémentaire du graphe.

   (c) Comment identifier toutes les liaisons ayant la propriété de la question 2a ?

   (d) De telles liaisons existent-elle dans le graphe de la figure 7 ?

   (e) Modifier le principe de la question 1f (et la fonction) : ajouter la détection de ces liaisons.

3. **Sous-réseau sûr :**

   (a) Comment appelle-t-on les sous-graphes qui ne contiennent pas de sommets du type de la question 1b ?

   (b) Comment peut-on identifier ces sous-graphes en utilisant les résultats déjà obtenus dans les questions précédentes ?

   (c) Déterminer les sous-graphes ayant la propriété de la question 3a, dans le graphe de la figure 7.

   (d) Modifier le principe de la question 2e, pour qu'il détermine aussi ces sous-graphes.

Exercice 3.2 (Algernon et le labyrinthe (suite))

Le nouveau labyrinthe d'Algernon est maintenant muni de portes ne pouvant s'ouvrir que dans un sens.

1. Le labyrinthe est très grand et contient des circuits. Comment savoir quels sont les différents chemins impossibles, sans parcourir tout le graphe à chaque fois ? Comment repérer les impasses dont elle ne pourra plus repartir ?

2. Algernon commence à décliner. Les chercheurs reprennent le labyrinthe initial (sans portes) pour y installer des nouvelles portes. Mais ils veulent s'assurer qu'Algernon pourra aller partout sans être bloquée.
   – Comment savoir quelles sont les portes qui doivent s'ouvrir dans les deux sens ?
   – Comment savoir dans quel sens doivent s'ouvrir les portes restantes ?