Informatique

TP X2

# Suppléments sur les graphes (simples)

## Connexité, cyles et arbres

**Résumé**

Dans TP, nous allons voir deux utilisations importantes des parcours de graphes : le calcul des « composantes connexes » d'un graphe d'une part, et le calcul « d'arbres couvrants » d'autre part.

En principe, ces deux utilisations peuvent être implémentées aussi bien à partir d'un parcours en profondeur d'abord que d'un parcours en largeur d'abord. Mais les résultats peuvent dépendre du parcours choisi.

# 1 Connexité

## 1.1 Définitions

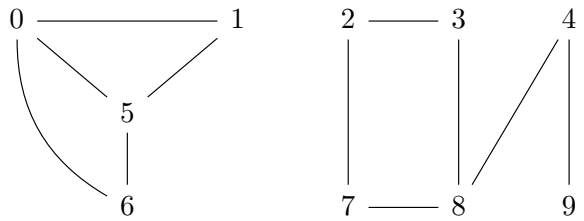
Soit $G = (V, E)$ un graphe simple. Une **composante connexe** de $G$ est un ensemble $C \subseteq V$ de sommets de $G$ tel que

(i) pour tous $u, v \in C$, il existe une chaîne entre $u$ et $v$ dans $G$, et

(ii) pour tout $u \in C$ et tout $v \in V \setminus C$, il n'existe pas de chaîne entre $u$ et $v$ dans $G$.

**Remarque 1.1.** *Soit $C$ une composante connexe de $G$. La condition (ii) implique que toute chaîne entre deux sommets de $C$ ne contient que des sommets de $C$.*

**Question 1.** *Donner les composantes connexes du graphe suivant :* Q.1

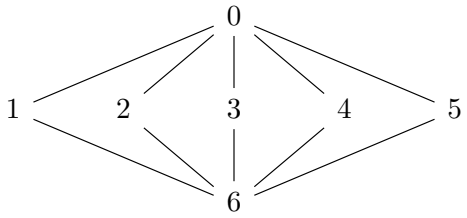



**Question 2.** *Donner le* **nombre** *de composantes connexes du graphe suivant :* Q.2



Un graphe simple $G = (V, E)$ est **connexe** si $V$ est une composante connexe de $G$ (ou, de manière équivalente, si tout sommet de $G$ est accessible à partir de n'importe quel autre sommet de $G$).

**Exemple 1.2.** *Les graphes des Questions 1 et 2 ne sont pas connexes. Le graphe suivant est connexe :*



### 1.2 Implémentations Python

**Question 3.** *Écrire une fonction* Python *qui prend en argument un graphe $G$ (représenté par liste d'adjacence) et qui renvoie la liste des toutes les composantes connexes de $G$.* Q.3

**Question 4.** *Écrire une fonction* Python *qui prend en argument un graphe $G$ et qui renvoie le nombre de composantes connexes de $G$.* Q.4

**Question 5.** *Écrire une fonction* Python *qui prend en argument un graphe $G$. Cette fonction doit renvoyer* `True` *si le graphe $G$ est connexe, et renvoyer* `False` *sinon.* Q.5

## 2 Chaînes simples, cycles et arbres

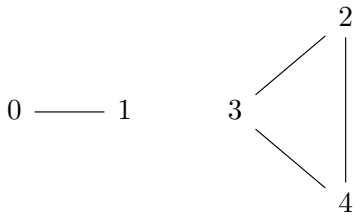
Soit $G = (V, E)$ un graphe.

**Chaînes simples.** On dit qu'une arête $\{u, v\} \in E$ **apparaît** dans une chaîne $v_0, \dots, v_k$ s'il existe (au moins) un $i < k$ tel que ($u = v_i$ et $v = v_{i+1}$) ou ($v = v_i$ et $u = v_{i+1}$).

**Exemple 2.1.** *Considérons les chaînes*

$$0, 1 \qquad 0, 1, 0 \qquad 2, 3, 4 \qquad \textit{et} \qquad 2, 3, 4, 2$$

*dans le graphe suivant :*




*L'arête $\{0, 1\}$ apparaît une fois dans la chaîne $0, 1$ et deux fois dans la chaîne $0, 1, 0$ (une fois dans le sens $0 \to 1$, et une fois dans le sens $1 \to 0$).*

*L'arête $\{2, 4\}$ n'apparaît pas dans la chaîne $2, 3, 4$, mais elle apparaît dans la chaîne $2, 3, 4, 2$ (dans le sens $4 \to 2$).*

Une chaîne $v_0, \dots, v_k$ est **simple** si toute arête apparaît au plus une fois dans $v_0, \dots, v_k$. Autrement dit, une chaîne $v_0, \dots, v_k$ est simple si toutes ses arêtes sont distinctes (si $i \neq j$, alors $\{v_i, v_{i+1}\} \neq \{v_j, v_{j+1}\}$).

**Question 6.** *Parmi les chaînes de l'Exemple 2.1, lesquelles sont simples ?* Q.6

**Cycles.** Un **cycle** est une chaîne simple $v_0, \dots, v_k$ telle que $k \geq 1$ et $v_0 = v_k$. Autrement dit, un cycle est une chaîne simple de longueur $\geq 1$ entre un sommet et lui même.

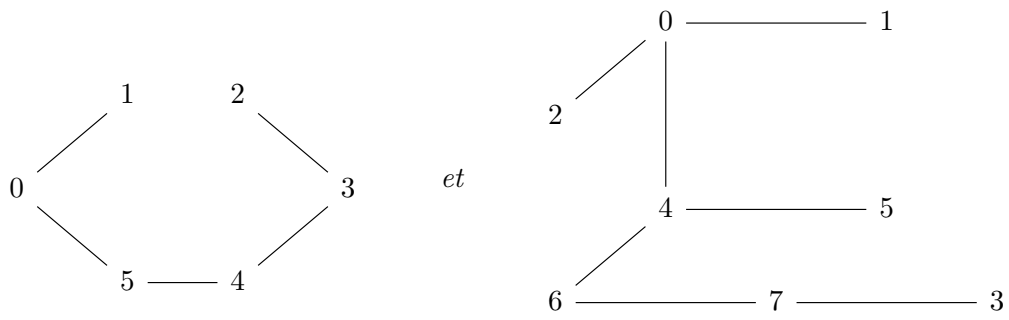
Un graphe est **acyclique** si aucune de ses chaînes (simple) n'est un cycle.

**Question 7.** *Parmi les chaînes de l'Exemple 2.1, lesquelles sont des cycles ?* Q.7

**Remarque 2.2.** *Un cycle doit comporter au moins trois sommets distincts.*

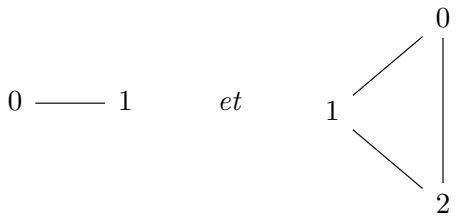
**Arbres.** Un **arbre** est un graphe (simple) connexe et acyclique.

**Exemple 2.3.** *Les deux graphes suivants sont des arbres :*




**Question 8.** *Donner des listes d'adjacences pour chacun des deux arbres de l'Exemple 2.3.* Q.8

**Question 9.** *Indiquer lesquels des graphes suivants sont des arbres :* Q.9

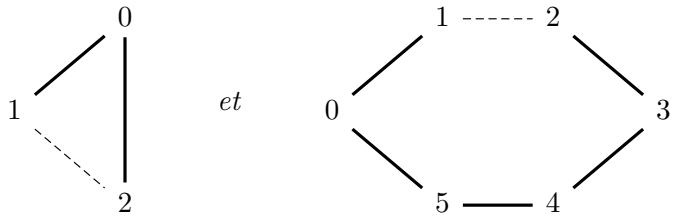



## 3 Arbres couvrants

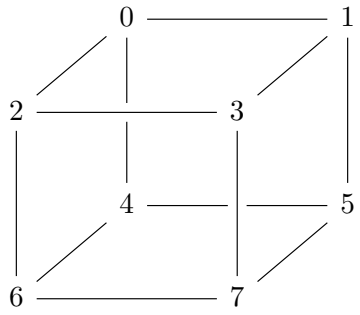
Soit $G = (V, E)$ un graphe (simple) connexe. Un **arbre couvrant** de $G$ est un arbre $A = (V, T)$ avec $T \subseteq E$. Autrement dit, un arbre couvrant de $G$ est un arbre qui a exactement les mêmes sommets que $G$ et dont toutes les arêtes sont des arêtes de $G$.

Par extension, si $G = (V, E)$ est un graphe non nécessairement connexe, et si $C \subseteq V$ est une composante connexe de $G$, alors un arbre couvrant de $C$ est un arbre couvrant du graphe de sommets $C$, et dont les arêtes sont les $\{u, v\} \in E$ tels que $u, v \in C$.

**Exemple 3.1.** *Voici deux arbres couvrants, chacuns pour un graphe vu précédemment (les arêtes des arbres couvrants sont en* ***gras****, les autres arêtes sont en pointillés) :*




**Question 10.** *Donner un arbre couvrant pour le cube* Q.10




**Question 11.** *Considérons le graphe de la Question 1. Donner un arbre couvrant de la composante connexe du sommet* 0 *et un arbre couvrant de la composante connexe du sommet* 2. Q.11

### 3.1 Construction d'arbres couvrants

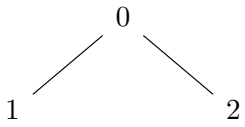
Nous allons maintenant utiliser les parcours de graphes pour obtenir des arbres couvrants.

L'idée est la suivante. Soit $G = (V, E)$ un graphe simple, avec $V = \{0, \dots, |V| - 1\}$, et soit $u$ un sommet de $G$. On cherche à construire un arbre couvrant de la composante connexe de $u$ dans $G$. Pour ce faire, on se donne une liste `parent`, de longeur $|V|$. Cette liste `parent` est initialisée à `None`. On parcourt $G$ depuis le sommet $u$, et lors de ce parcours, à chaque fois qu'on explore une liste de voisins `lst`$[a]$, pour chaque sommet $b$ (non encore visité) de cette liste, on met `parent`$[b]$ à $a$. Autrement dit, lorsqu'on traverse pour la première fois une arête $\{a, b\} \in E$ dans le sens $a \to b$, on indique dans `parent`$[b]$ que le sommet $b$ a été découvert en venant du sommet $a$.

En fin d'algorithme, on obtient une liste `parent` qui contient uniquement `None` et des entiers parmi $0, \dots, n-1$, où $n$ est la longueur de `parent`. Si `parent`$[i] \neq i$ pour tout $i = 0, \dots, n-1$, alors une telle liste `parent` définit un graphe $P$ de sommets $\{0, \dots, n-1\}$ et tel que

$$\{a, b\} \text{ est une arête de } P \qquad \text{si et seulement si} \qquad (\texttt{parent}[a] = b \ \text{ ou } \ \texttt{parent}[b] = a)$$

**Exemple 3.2.** *Pour* `parent` $= [\texttt{None}, 0, 0]$*, le graphe* $P$ *est*



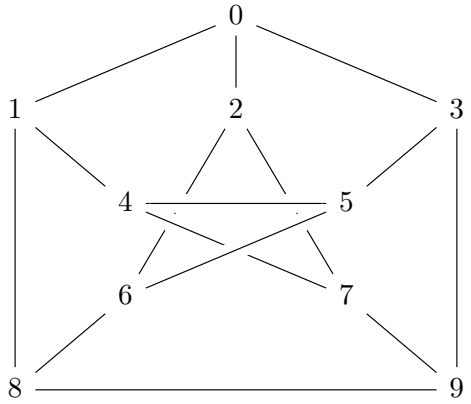

**Question 12.** *Donner le graphe* $P$ *pour* `parent` $= [\texttt{None}, 0, 0, 7, 0, 4, 4, 6]$. Q.12

**Question 13.** *Écrire une fonction* Python *qui prend arguement une liste* `parent` *comme ci-dessus, et qui renvoie une liste d'adjacence pour le graphe* $P$ *correspondant.* Q.13

**Question 14.** *Écrire une fonction* Python *qui prend en arguments un graphe* $G$ *et un sommet* $u$ *de* $G$*. Cette fonction doit renvoyer une liste* `parent` *comme ci-dessus, et telle que la composante connexe de* $u$ *dans* $P$ *est un arbre couvrant de la composante connexe de* $u$ *dans* $G$*.* **Pour simplifier, on pourra supposer que** $G$ **est connexe, auquel cas** $P$ **doit être un arbre couvrant de** $G$**.** Q.14

**Question 15.** *Utilisez votre fonction de la Question 14 pour donner un arbre couvrant du graphe* Q.15

*de Petersen :*




### 3.2 Détection de cycles

Nous allons maintenant voir que l'algorithme de la Question 14 donne une méthode très simple pour détecter si un graphe (simple) est acyclique. Pour simplifier, on ne considère que des graphes connexes. Rappelons qu'un tel graphe est acylique si, et seulement si, c'est un arbre.

Soit donc $G = (V, E)$ un graphe simple connexe. et considérons la liste `parent` renvoyée par l'algorithme de la Question 14, à partir d'un sommet quelconque de $G$. Soit $P$ le graphe induit par `parent`. Comme $G$ est connexe, $P$ est un arbre couvrant de $G$. De plus, on peut voir que $P$ a exactement $|V| - 1$ arêtes.

Comme toutes les arêtes de $P$ sont des arêtes de $G$, si $|E| = |V| - 1$ alors $G = P$, et $G$ est acyclique. Supposons maintenant que $G$ est acyclique, et considérons une arête $\{u, v\} \in E$ de $G$. Si cette arête n'est pas dans $P$, alors la rajouter à $P$ crée un cycle (en effet, il existe une chaîne simple entre $u$ et $v$ dans $P$, et cette chaîne ne contient pas l'arête $\{u, v\}$ par hypothèse). Ainsi, toute arête de $G$ est une arête de $P$, et $|E| = |V| - 1$.

Nous avons donc le résultat suivant.

**Thérème 3.3.** *Un graphe (simple) connexe $G = (V, E)$ est acyclique si, et seulement si, $|E| = |V| - 1$.*

**Question 16.** *Écrire une fonction* Python *qui prend en argument un graphe $G$. Cette fonction doit renvoyer* `True` *si $G$ est acyclique, et renvoyer* `False` *sinon.* **On supposera que $G$ est connexe. Le nombre d'opérations doit être linéaire en la taille de $G$.** Q.16