TD13: Graphes

MP2I Lycée Pierre de Fermat

Dans tout le TD, on considère des graphes **finis**. Certaines questions sont marquées d'un **(*)**, ce qui signifie que vous trouverez à la fin du TD une indication pour cette question si vous bloquez.

Définition 1. Soit $G = (S, A)$ un graphe et $S' \subseteq S$. Le sous-graphe de G induit par S' est $G' = (S', A \cap S'^2)$. C'est donc le graphe obtenu en ne gardant que les sommets de S', et que les arêtes dont les deux bouts sont dans S'.

Exercice 1.

Degrés, hypercubes

Q1. Montrer qu'un graphe non-orienté sans boucles possède un nombre pair de sommets de degré impair.

On considère l'hypercube de dimension n, noté H_n qui est un graphe dont l'ensemble des sommets est $\{0,1\}^n$, et dans lequel deux sommets $(a_1, \dots, a_n)$ et $(b_1, \dots, b_n)$ sont reliés si et seulement si ils diffèrent d'exactement une coordonnée.

Q2. Donner le nombre de sommets de H_n, ainsi que le degré de ses sommets. En déduire le nombre d'arêtes de H_n.

Exercice 2.

Line graph

On considère un graphe $G = (S, A)$ non-orienté. Le **line graph** de G, noté $L(G)$ est définit comme suit.

- L'ensemble des sommets de $L(G)$ est l'ensemble des arêtes de G, A.
- Deux sommets a, b de $L(G)$ sont reliés par une arête s'il existe un sommet $u \in S$ tel que les arêtes a et b sont incidentes à u.

Q1. Dessiner le graphe dont la matrice d'adjacence est la suivante, puis dessiner son line graph.

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Q2. Dessiner le line graphe de K_4 (le graphe complet à 4 sommets, c'est à dire le tétraèdre).

Q3. Montrer que si un graphe est régulier, de degré d, alors son line-graph est régulier, et donner son degré.

Q4. **(*)** Si $G = (S, A)$ est un graphe non-orienté, avec n sommets et m arêtes, déterminer le nombre de sommets et d'arêtes de $L(G)$.

Exercice 3.

Graphes connexes

On considère des graphes non-orientés, sans boucles. Soit $G = (S, A)$ un graphe non orienté, et notons $n = |S|, m = |A|$. On souhaite montrer que $m \geq n - 1$.

Q1. Dessiner quelques graphes connexes pour vous convaincre du résultat.

Q2. Que dire pour $n = 1$?

Q3. Soit $G = (S, A)$ connexe possédant au moins 2 sommets, et soit u un sommet quelconque de G.

On considère le sous-graphe G' de G induit par $S \setminus \{u\}$. On note $C_1, \ldots, C_p$ ses composantes connexes. On note $G_1, \ldots, G_p$ les sous-graphes induits de G correspondants. On note n_i le nombre de sommets de G_i, m_i le nombre d'arêtes de G_i pour $i \in [\![1, p]\!]$

Q4. Que peut-on dire de p par rapport à $\mathbf{deg}(u)$?

Q5. Donnez un lien entre m, $\mathbf{deg}(u)$ et les m_i.

Q6. Donnez un lien entre n et les n_i.

Q7. En utilisant les résultats précédents, montrer par récurrence **forte** sur n que $m \geq n - 1$.

Exercice 4.

Graphes acycliques

Définition 2. Un graphe non-orienté $G = (S, A)$ est **acyclique** s'il ne contient aucun cycle.

On souhaite montrer que pour $G = (S, A)$ un graphe non-orienté acyclique, en notant $n = |S|$ et $m = |A|$, on a $m \leq n - 1$.

Q1. Dessiner quelques graphes acycliques pour vous convaincre du résultat.

Q2. (*) Montrer qu'un graphe acyclique non vide possède un sommet de degré au plus 1.

Q3. Montrer par récurrence faible sur n que $m \leq n - 1$.

Exercice 5.

Arbres

Définition 3. Un graphe est un **arbre** s'il est connexe ET acyclique.

Soit $G = (S, A)$ un graphe non-orienté. Notons $n = |S|$ et $m = |A|$. Montrer que les trois propriétés suivantes sont équivalentes :

(i) G est un arbre

(ii) G est connexe et $m = n - 1$.

(iii) G est acyclique et $m = n - 1$.

Exercice 6.

Démo du cours : parcours en largeur

Définition 4. Soit $G = (S, A)$ un graphe orienté. La **distance** entre deux sommets $u \in S$ et $v \in S$ est la longueur du plus court chemin les reliant, ou $+\infty$ si aucun chemin ne va de u à v.

On considère le graphe $G_0 = (S_0, A_0)$ ci-dessous :

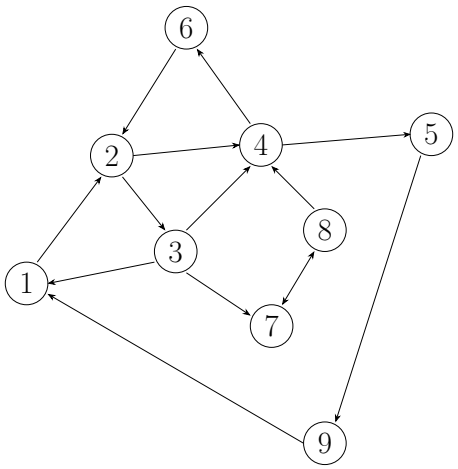

FIGURE 1 – Graphe G_0

Sur ce graphe par exemple, la distance entre les sommets 5 et 2 est de 3 car il existe un chemin de taille 3 ($5 \to 9 \to 1 \to 2$) mais aucun chemin strictement plus court.

Q1. A la main, donner les distances entre le sommet 1 et chaque sommet du graphe

Le but de cet exercice est de démontrer ce que vous avons observé empiriquement en cours : le parcours en largeur permet de calculer la distance d'un sommet à tous les autres, et l'arborescence du parcours permet de reconstruire des plus courts chemins. On propose l'algorithme suivant :

Algorithme 1 : Parcours en largeur : calcul des distances

Entrée(s) : $G = (S, A)$ un graphe, $s \in S$
Sortie(s) : (d, P) avec d tableau des distances de s à chaque sommet de G et P tableau des prédécesseurs de l'arborescence de parcours.

```
1  d ← dictionnaire vide // d[u] contient la distance entre s et u
2  Initialiser d[u] = +∞ pour chaque u ∈ S;
3  P ← dictionnaire vide // P[u] contient le prédécesseur de u
4  F ← file vide;
5  d[s] ← 0;
6  enfiler(F, s);
7  tant que F non vide faire
8      u ← defiler(F);
9      pour v voisin de u faire
10         si d[v] = +∞ alors
               // sommet jamais croisé
11             d[v] ← d[u] + 1;
12             P[v] ← u;
13             enfiler(F, v);
14 retourner (d, P)
```

Q2. Appliquer l'algorithme sur G_0 en partant du sommet 1. Noter les tableaux d et P obtenus.

Montrons la correction de cet algorithme. On veut montrer d'une part que d contient bien les distances du sommet source à tout sommet, et que P contient l'information permettant de reconstruire des chemins minimaux.

Q3. Montrer l'invariant suivant : "pour tout $u \in S \setminus \{s\}$, si $d[u] \neq +\infty$, alors il existe un chemin de s à u de longueur $d[u]$, dont la dernière arête est $(P[u], u)$".

Q4. En déduire qu'en sortie de l'algorithme, pour tout $u \in S$, $d[u] \geq \delta(u)$

Montrons maintenant que $d[u]$ atteint exactement $\delta(u)$. Pour commencer, nous allons montrer que l'ordre des sommets dans F est extrêmement restrictif : si F contient $u_1, \ldots, u_k$ dans l'ordre de la tête à la queue, alors $d[u_1] \leq \cdots \leq d[u_k]$, et $d[u_k] = d[u_1]$ ou $d[u_1] + 1$.

Q5. Montrer que la propriété annoncée est bien un invariant de boucle.

Supposons, par l'absurde, qu'il existe $u \in S$ tel que $\delta(u) < d[u]$ en sortie. Soit $x_0 x_1 \ldots x_p$ un chemin minimal de s à u. En particulier, $p = \delta(u)$.
On considère $E = \{i \in [\![0, p]\!] \mid \delta(x_i) < d[x_i]\}$ l'ensemble des sommets du chemin pour lesquels l'algorithme s'est trompé.

Q6. Justifier que $\min(E)$ existe et ne vaut pas 0.

On considère $v = P[x_i]$.

Q7. Donner un lien entre $d[v]$ et $d[x_i]$, puis un lien entre $\delta(x_{i-1})$ et $\delta(x_i)$.

Q8. En déduire que $d[v] < d[x_{i-1}]$, puis trouver une absurdité.

Exercice 7.

Démo du cours : Tri topologique

On rappelle l'algorithme plus efficace proposé en cours pour calculer un tri topologique :

Algorithme 2 : Tri topologique

Entrée(s) : $G = (S, A)$ graphe orienté sans cycle
Sortie(s) : L tri topologique de G

```
1  n ← |S|;
2  L ← liste vide;
3  P ← pile vide;
4  d⁻ ← dictionnaire vide ;
   // Initialiser d⁻
5  pour u sommet de G faire
6      d⁻[u] ← 0;
7  pour u sommet de G faire
8      pour v successeur de u faire
9          d⁻[v] = d⁻[v] + 1;
   // Empiler les sources
10 pour u sommet de G faire
11     si d⁻[u] = 0 alors
12         empiler(P, u);
   // Parcours
13 tant que P non vide faire
14     u ← depiler(P);
15     Ajouter u à la fin de L;
16     pour v voisin de u faire
17         d⁻[v] = d⁻[v] − 1;
18         si d⁻[v] = 0 alors
19             empiler(P, v);
20 retourner L
```

On considère les deux propriétés suivantes :

$$\forall u \in S, d^-[u] = |\{v \in S \mid (v,u) \in A \text{ et } v \notin L\}| \tag{1}$$

$$\forall u \in S, d^-[u] = 0 \iff (u \in P \text{ ou } u \in L) \tag{2}$$

Q1. Montrer que ces deux propriétés sont bien des invariants de la boucle while.

Nous allons utiliser ces deux propriétés pour montrer la correction de l'algorithme. Pour cela, nous voulons montrer que L énumère bien tous les sommets de S, et que l'ordre de chaque arête est bien respecté.
On se place en sortie de la boucle while.

Q2. Montrer que si $(x, y) \in A$ et x, y apparaissent tous les deux dans L, alors x apparaît avant y dans L. On pourra considérer l'instant où y a été ajouté dans L.

Q3. (*) Montrer que s'il existe un sommet $u \in S$ tel que u n'apparaît pas dans L, alors il existe un sommet $u' \in S$ (éventuellement égal à u) n'apparaissant pas dans L **et** dont tous les prédécesseurs sont dans L.

Q4. En déduire qu'il n'existe pas de sommet $u \in S$ n'apparaissant pas dans L, et conclure sur la correction de l'algorithme.

Exercice 8.

Matrice d'incidence

Soit $G = (S, A)$ un graphe orienté sans boucles, avec $n = |S|$ et $m = |A|$. On note $S = \{s_1, \ldots, s_n\}$ et $A = \{a_1, \ldots, a_m\}$ La matrice d'incidence de G est $M = (m_{ij})_{1 \leq i \leq n, 1 \leq j \leq m}$ définie par :

$$\forall i \in [\![1, n]\!], \forall j \in [\![1, m]\!], m_{ij} = \begin{cases} 1 \text{ si l'arête } a_j \text{ arrive sur } s_i \\ -1 \text{ si l'arête } a_j \text{ part de } s_i \\ 0 \text{ sinon} \end{cases}$$

La matrice d'incidence indique ainsi les relations entre les différentes arêtes et sommets d'un graphe.

Q1. Dessiner le graphe dont la matrice d'incidence est :

$$\begin{pmatrix} -1 & 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Q2. Quelle est la complexité spatiale nécessaire pour cette représentation ?

Q3. Donner un algorithme permettant de déterminer si deux sommets (u, v) d'un graphe sont voisins, à partir de cette représentation. Quelle est sa complexité ?

Q4. Donner un algorithme permettant de récupérer la liste des successeurs d'un sommet u dans cette représentation. Quelle est sa complexité ?

Q5. Soit $G = (S, A)$ un graphe, et M sa matrice d'incidence. Décrire la matrice d'incidence du graphe obtenu en inversant le sens de toutes les arêtes de G.

Q6. Soit $G = (S, A)$ un graphe, et M sa matrice d'incidence. On note ${}^tM$ sa transposée. Que contient la matrice $M\,{}^tM$?

Exercice 9.

Chemins eulériens et hamiltoniens

On considère des graphes non orientés, sans boucles.

Un chemin dans un graphe G est **hamiltonien** s'il passe une fois et une seule par chaque sommet de G. On dit qu'un cycle est hamiltonien s'il passe une fois et une seule par chaque sommet, sauf le sommet qui sert de départ et d'arrivée, qui est visité deux fois.

Un chemin est **eulérien** s'il passe une fois et une seule par chaque arête.

Q1. Dans les graphes suivants, dire s'il existe des chemins eulériens / cycles eulériens / chemins hamiltoniens / cycles hamiltoniens, et les donner lorsqu'ils existent :

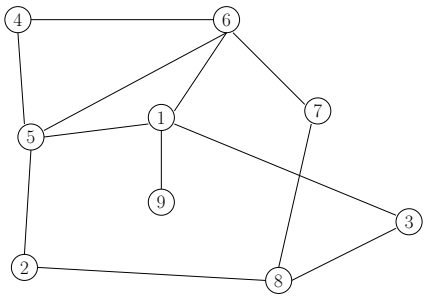

FIGURE 2 – G_1

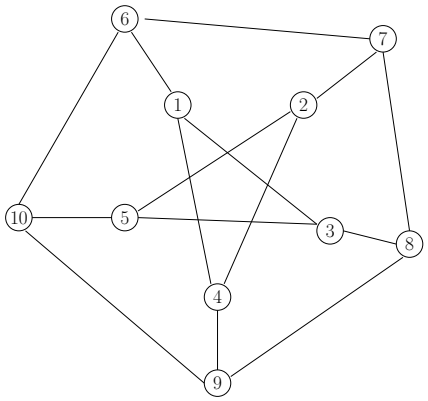

FIGURE 3 – G_2

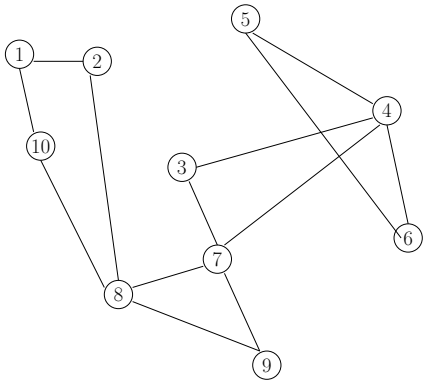

FIGURE 4 – G_3

Q2. Montrer que si un graphe admet un cycle eulérien, alors il est connexe, et tous ses sommets sont de degré pair.

Montrons maintenant que c'est une condition nécessaire et suffisant : un graphe admet un cycle eulérien si et seulement si il est connexe et que tous ses sommets sont de degré pair (on dira que G est de degrés pairs).

Q3. Montrer qu'un graphe connexe de degrés pairs admet un cycle d'au moins 3 sommets. Déduire de votre preuve un algorithme permettant de trouver un tel cycle.

Q4. Soit $G = (S, A)$ un graphe connexe de degrés pairs, et C un cycle. Montrer que le graphe G' obtenu en supprimant de G les arêtes de C est toujours de degrés pairs.

Q5. (*) Déduire des deux questions précédentes un algorithme permettant de construire un cycle eulérien dans un graphe de degrés pairs.

Q6. On considère maintenant des **chemins** eulériens. Donner un critère nécessaire et suffisant pour qu'un graphe admette un chemin eulérien n'étant pas un cycle. Démontrer ce critère en réutilisant le résultat montré sur les cycles eulériens.

Exercice 10. *Vertex-cover*

Pour un graphe non-orienté $G = (S, A)$, le problème de couverture des arêtes par les sommets, noté `Vertex-Cover`, est de trouver un sous-ensemble de sommets $C \subseteq S$ tel que toute arête (u, v) a un sommet au moins dans C.
Un algorithme glouton simple consiste à choisir le sommet de plus grand degré, à l'ajouter à C, à le supprimer de G (ainsi que ses arêtes incidentes), et à recommencer l'opération tant qu'il reste des arêtes :

Algorithme 3 : `gloutonVC`(G)

Entrée(s) : $G = (S, A)$ un graphe non-orienté
Sortie(s) : Une vertex-cover des arêtes de G

1 $C \leftarrow \emptyset$;
2 **tant que** ***il existe un sommet*** u ***degré*** > 0 ***dans*** G **faire**
3 $\quad u \leftarrow$ sommet de G de degré maximal;
4 $\quad C \leftarrow C \cup \{u\}$;
5 $\quad$ Supprimer u de S et ses arêtes de A;
6 **retourner** C

Q1. Montrer à l'aide d'un contre-exemple que cet algorithme n'est pas optimal.

On suppose que l'on implémente les graphes par tableaux de listes d'adjacences.

Q2. Quel est le coût de supprimer un sommet u d'un graphe $G = (S, A)$ représenté par tableaux de listes d'adjacences, en fonction de $n = |S|$ et $m = |A|$?

Q3. Quelle est la complexité de l'algorithme `gloutonVC` ?

Afin d'améliorer la complexité de l'algorithme, on utilise une structure permettant de stocker l'état du graphe au cours de l'algorithme, sans le modifier directement. Plus précisément, on stocke dans un tableau T le degré de chaque sommet. Au départ, la case $T[i]$ contient le degré de i dans le graphe G, et lorsque l'on supprime un sommet à la ligne 4 de l'algorithme, on **modifie** T sans toucher G.

Q4. Avec cette nouvelle structure, quel est le coût pour supprimer un sommet u ?

Q5. Réécrire l'algorithme `gloutonVC` en explicitant l'utilisation du tableau des degrés, et évaluer la complexité.

Indications

— Exercice 4, **Q2.** : on pourra raisonner par l'absurde, et arriver à construire une suite infinie de sommets distincts.
— Exercice 2, **Q4.** A quoi correspond un sommet de G dans $L(G)$?
— Exercice 6, **Q8.** Considérer l'ordre dans lequel les sommets sont dépilés, et le moment où $d[x_i]$ a été modifié.
— Exercice 7, **Q3.** On pourra raisonner par l'absurde et construire un cycle dans G.
— Exercice 9, **Q5.** : Raisonner récursivement, pour construire le cycle eulérien morceau par morceau.