TD - parcours de graphe

1. Graphe non orienté :

0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0
1	1	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1	0	0
0	0	0	0	1	0	1	0	1	0
0	1	1	0	0	1	0	0	1	0
0	0	0	1	1	0	0	0	0	1
0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	1	0	0

0	2
1	2, 6
2	0, 1, 4, 6
3	4, 7
4	2, 3, 5, 7
5	4, 6, 8
6	1, 2, 5, 8
7	3, 4, 9
8	5, 6
9	7

Graphe orienté :

0	0	0	0	1	1	0	0	0	
1	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	1	0	0	
1	1	0	0	0	0	0	0	1	
0	1	0	0	0	0	0	0	0	
0	1	0	0	0	0	1	0	0	
0	0	1	0	0	0	0	0	0	
0	0	0	1	0	0	0	0	0	
0	0	0	0	0	0	1	0		

0	4, 5
1	0
2	6
3	0, 1, 8
4	1
5	1, 6
6	2
7	3
8	7

2.

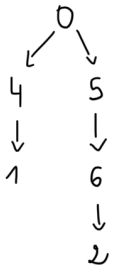
	file	vus	sommet actuel
Avant la boucle	← 5	{5}	x
1ère itération	← 4, 6, 8	{5, 4, 6, 8}	5
2ème itération	← 6, 8, 2, 3, 7	{5, 4, 6, 8, 2, 3, 7}	4
3ème itération	← 8, 2, 3, 7, 1	{5, 4, 6, 8, 2, 3, 7, 1}	6
4ème it.	← 2, 3, 7, 1	//	8
5ème it.	← 3, 7, 1, 0	{5, 4, 6, 8, 2, 3, 7, 1, 0}	2
6ème it.	← 7, 1, 0	//	3
7ème it	← 1, 0, 9	{5, 4, 6, 8, 2, 3, 7, 1, 0, 9}	7
8ème it.	← 0, 9	//	1
9ème it.	← 9	//	0
10ème it.	←	//	9

ordre de traitement ↓

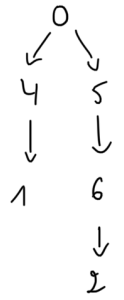
3. On trouve : 5, 4, 2, 0, 1, 6, 8, 3, 7, 9

4. On trouve : 5, 8, 6, 2, 0, 1, 4, 7, 9, 3
Ce n'est donc pas un parcours en profondeur.

5. Profondeur : 0, 4, 1, 5, 6, 2



Largeur : 0, 4, 5, 1, 6, 2



6. Un parcours de graphe (en profondeur ou en largeur) depuis un sommet x va explorer tous les sommets accessibles depuis x. Ici, tous les sommets sont accessibles depuis 3, 7 ou 8 : on peut donc prendre un de ces trois sommets comme départ.

7. Un graphe orienté admet un ordre topologique si et seulement s'il est <u>acyclique</u>. Ce n'est donc pas le cas ici. Lors du parcours en profondeur, si on retombe sur un sommet déjà vu, alors il y a un circuit (et donc pas d'ordre topologique).

8. 9. Si M est la matrice d'adjacence d'un graphe, alors le nombre de chemins de longueur p qui relient le sommet x au sommet y est égal au coefficient de M^p ligne x, colonne y (Rmq : se montre par récurrence sur p).

10. <u>Entrées</u> : - M la matrice d'adjacence d'un graphe $G=(S,A)$
- deux sommets de G : $x, y \in S$

<u>Algo</u> renvoyant vrai ssi y accessible depuis x :

$p \leftarrow 0$

Tant que $M^p_{x,y} = 0$ et $p < |S|$:

$p \leftarrow p+1$

Renvoyer $p \neq |S|$

<u>Explications :</u>

On regarde s'il existe un chemin de longueur 0 reliant x à y. Si oui, alors $M^0_{x,y} \neq 0$ donc on sort de la boucle et on renvoie vrai. Sinon, on regarde s'il existe un chemin de longueur 1 reliant x à y. Si oui, alors $M^1_{x,y} \neq 0$ donc on sort de la boucle et on renvoie VRAI. Ainsi de suite, on regarde s'il existe un chemin de longueur 2, 3, ... reliant x à y. D'après une propriété du cours, s'il existe un chemin de x à y alors il existe un chemin élémentaire de x à y (c.à.d. ne passant pas deux fois par le même sommet). Donc si on arrive à $p = |S|$ sans avoir trouvé de chemin, on sait que y n'est pas accessible depuis x : on sort de la boucle et on renvoie FAUX.

<u>Complexité</u>

On suppose que pour calculer M^p, on effectue simplement une multiplication de M^{p-1} (déjà calculée à l'itération précédente) avec M (ce qui est plus optimal que de recalculer entièrement M^p à chaque itération).
Cette multiplication a <u>généralement</u> un coût de $|S|^3$.
(avec une implémentation simple)

Le reste des opérations effectuées à chaque itération ont une complexité en $O(1)$. Avant et après la boucle, les opérations sont également en $O(1)$. On a donc dans le pire des cas la complexité suivante pour l'algo : $O(1) + \sum_{i=0}^{|S|-1} \left(O(|S|^3) + O(1) \right)$

en dehors de la boucle — nb max d'itérations — dans la boucle

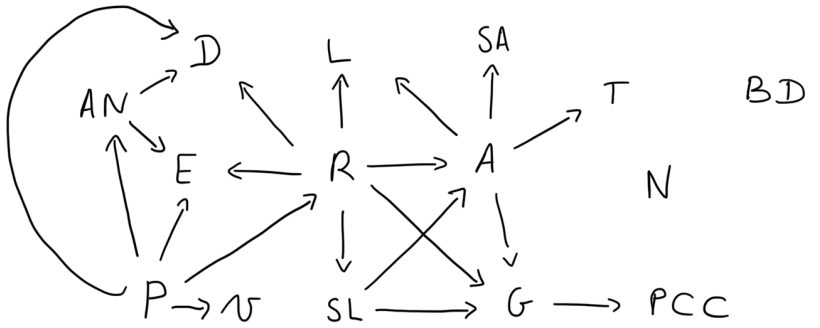
Donc une complexité en $O(|S|^4)$.

11. On peut prendre un parcours en profondeur ou en largeur, avec départ = x. Il suffit d'ajouter la ligne suivante à la toute fin de l'algo (c.à.d après la boucle dans le parcours en largeur, ou après l'appel à explorer(départ) dans le parcours en profondeur) : RENVOYER $y \in vus$.

Complexité : $O(|S| + |A|)$ en supposant avoir la liste d'adjacence même si le nombre d'arêtes est maximal (c.à.d. de l'ordre de $|S|^2$), c'est nettement mieux que l'approche de la question précédente.

12. On utilise un parcours de graphe pour être efficace. Il suffit de lancer un parcours (profondeur ou largeur) depuis n'importe quel sommet et de vérifier à la toute fin que $|vus| = |S|$.

13. A = arbres G = graphes T = algo du texte L = logique
SA = structures à l'aide d'arbres SL = structures linéaires
D = décomposition d'un problème U = validation de programmes
BD = bases de données N = repr. des nombres R = récursivité
E = exploration exhaustive P = bases de programmation
AN = analyse d'algos PCC = plus court chemin



Tri topologique :

BD < N < P < R < SL < A < G < PCC < T < SA < L < AN < D < E < U

(ce n'est pas la seule possibilité)

Pour obtenir un tri topologique d'un GO acyclique, il suffit de lancer un parcours en profondeur
- depuis chaque sommet de degré entrant nul, et
- de ranger les sommets par ordre de fin de parcours décroissantes.

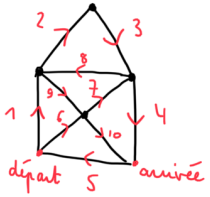
14. Il s'agit de trouver un cycle hamiltonien dans le complémentaire du graphe. Par exemple : B, C, H, A, F, G, E, D.

15. On dessine $G = (S, A)$ avec $S = \{A, B, C, D, E, F, G\}$ et $\{x, y\} \in A$ ssi x et y se sont rencontrés. Le nombre chromatique $\chi(G)$ du graphe est le nombre de places assises nécessaires. Ici on trouve $\chi(G) = 4$.

16. 17. Un GNO possède un <u>chemin eulérien</u> ssi il possède exactement 0 ou 2 sommets de degré impair.
C'est le cas de la figure donnée (les deux sommets en bas sont de degré 3, les autres sont de degré 2 ou 4).

On peut trouver un chemin eulérien en prenant comme sommet de départ un des 2 sommets de degré impair (et l'arrivée sera l'autre).

Exemple :



Il n'y a cependant pas de <u>cycle eulérien</u> car il faudrait pour cela qu'il n'y ait aucun sommet de degré impair.

On peut le montrer par l'absurde (s'il existe un sommet de degré impair, alors on pourra à un moment du chemin "arriver" sur ce sommet mais pas en "repartir" ou inversement).