TD2

Exercice 2.1 (exercice de cours)

$F = \emptyset$ (initialisation)
$\forall v$, v.visité = FALSE
Tant que $V(F) \neq V(G)$
- Choisir un sommet non visité v
- $T = BFS(G, v)$
- Marquer tous les sommets de T comme visités (peut être interne à BFS)
- $F = F \cup T$

Le résultat est une forêt : si une arête relie deux arbres T_1 et T_2, $T_1 \cup T_2$ est connexe et le BFS l'aurait parcouru en une seule fois

La forêt est couvrante puisque c'est le critère d'arrêt.

Dans le cas non-orienté, chaque BFS découvre exactement la composante connexe du sommet de départ choisi. La taille de la forêt est donc toujours égale au nombre de composantes connexes de G.

Dans le cas orienté, ce n'est plus le cas :

$$v_1 \to v_2 \to v_3 \; - - - - \; v_{n-1} \to v_n$$

En débutant par v_1, on aura une forêt couvrante à un arbre.
En débutant par v_n, puis v_{n-1}, ..., elle aura n arbres

Exercice 2.2 (exercice de cours)

1 - Descendants : 5, 6, 7

Ascendants : 1, 2, 3, 5, 6

2 - DFS ou BFS orienté, en ne considérant une fois que les voisins extérieurs et une fois que les voisins intérieurs.

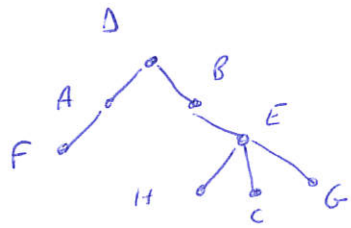
Remarque : Plutôt que d'implémenter un nouvel algorithme pour les ascendants, on peut retourner les arêtes du graphe et appliquer celui des ~~ascendants~~ descendants.

Complexité en $O(m)$.

Exercice 2.7

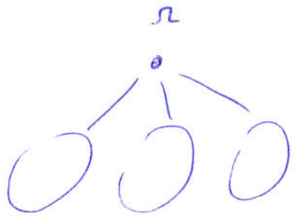
1. D et E sont des séparateurs.

2.



D est de degré 2 dans cet arbre, et supprimer D dans G crée deux composantes connexes.

3.



les sommets situés sur une même branche émanant de r sont dans la même composante connexe de $G-r$. En effet, les arêtes de T sont des arêtes de G et tous ses sommets peuvent donc être reliés par des chemins.

Supposons qu'il existe une arête entre deux telles branches. Alors il existerait une ~~arêtes~~ arête présente dans G et non dans T qui ne relie pas deux sommets dont l'un est le descendant de l'autre dans T. Ceci est impossible (cours) dans un DFS.

Chaque branche émanant de r correspond donc à une composante connexe de $G-r$.

On en déduit que

$\boxed{\text{$r$ est un séparateur si et seulement si il est de degré 1 dans ~~les~~ DFS enraciné en r}}$

4. Séparateurs $= \emptyset$

Pour tout $v \in V$

  $T = \text{DFS}(G, v)$

  Si degré de v dans $T > 1$

    Séparateurs $=$ Séparateurs $\cup \{v\}$

On lance autant de DFS qu'il y a de sommets donc l'algorithme est de complexité $O(nm)$

Exercice 2. 4.

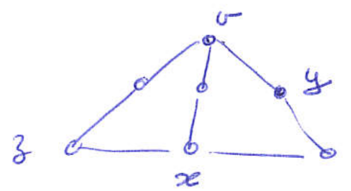
1. Cf cours : S'il existait un chemin plus court, il y aurait une arête qui "saute" un niveau dans $G \setminus T$, ce qui est impossible dans un BFS.

Algorithme : BFS + choisir le dernier sommet parcouru

2. Lancer l'algorithme précédent en chaque sommet.
Complexité $\mathcal{O}(nm)$

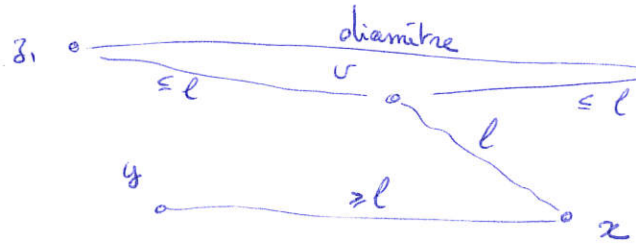
3. * Il s'agit de deux BFS $\rightarrow \mathcal{O}(E)$. Récupérer le niveau de y dans le dernier BFS est également linéaire (on peut même l'intégrer directement au BFS).

*



x bien à distance max de v, y à distance max de x.
Pourtant, $d(x,y)=2$ et $d(y,z)=3$

*



Soit (z_1, z_2) les extrémités d'un chemin diamétral. Par maximalité de x dans le BFS enraciné en v, $d(v,z_1) \leq d(v,x)$ et $d(v,z_2) \leq d(v,x)$

Par inégalité triangulaire, $\mathrm{diam}(G) = d(z_1, z_2) \leq 2\, d(v,x)$.

Par maximalité de y dans le BFS enraciné en x, $d(v,x) \leq d(y,x)$

Donc $\mathrm{diam}(G) \leq 2\, d(x,y)$. L'autre inégalité découle de la déf. du diamètre.

Exercice 2.5

1. On considère un DFS T.

Proposition Tout cycle de G contient une arête entre deux sommets u et v tels que u est un descendant de v dans T

Dém : Tout cycle contient une arête de $G \setminus T$ puisque T est acyclique. D'après le cours sur la structure des DFS, une telle arête relie deux sommets dont l'un est le descendant de l'autre. $\square$

Algorithme : On fait un DFS.

Si à un moment, ~~l~~un voisin v du sommet courant u apparaît comme déjà visité, on renvoie le cycle composé de l'arête vu et du chemin de u à v dans T

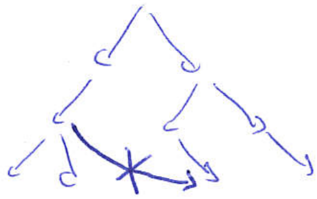
Si le DFS se termine sans que ce soit le cas, le graphe est acyclique

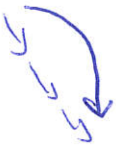
2. On considère à nouveau un DFS T.

Proposition Tout cycle orienté de G contient une arête $\overrightarrow{uv}$ telle que u est un descendant de v dans T

Dém : On suppose que T est construit de haut en bas, et de gauche à droite.

D'après le cours, il ne peut pas y avoir d'arête dans $G \setminus T$ qui va d'une branche à l'autre, de gauche à droite

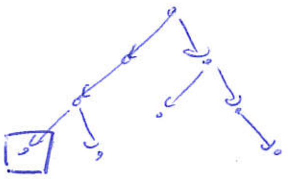


Il n'y a que des arêtes dans $G \setminus T$ qui descendent , qui remontent  ou qui vont de droite à gauche

Un cycle sans arête qui remonte est donc impossible puisqu'on ne pourrait aller que vers le bas (via T ou $G \setminus T$) et vers la gauche. Tout cycle contient donc une arête qui remonte de $G \setminus T$. □

On peut reprendre l'algorithme du cas non-orienté, à part que si v est déjà marqué comme visité, il faut vérifier s'il appartient aux ancêtres de u avant de retourner un cycle.

Si tous les sommets ont un degré extérieur ≥ 1, on considère le sommet en bas à gauche de T.



L'arête qui en sort ne peut pas descendre ni aller vers la droite. Elle remonte forcément, créant donc un cycle.

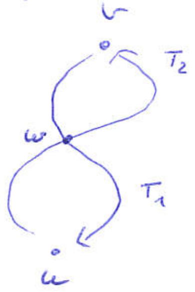
Exercice 2. 6

Algorithme : 1) On lance un BFS orienté T sur G, enraciné en v, et pour tout u on stocke son niveau $n_1(u)$.

2) On crée G' obtenu en changeant l'orientation des arêtes de G. On lance un BFS T' enraciné en v sur G' et pour tout u on stocke son niveau $n_2(u)$

3) On sélectionne $u \neq v$ tel que $n_1(u) + n_2(u)$ est minimum. ~~L'union~~ On retourne le cycle formé par le chemin de v à u dans T_1 et celui de u à v dans T_2.

Validité : Le résultat est bien une marche orientée. Pour vérifier que c'est un cycle, il faut que les deux chemins dont on fait l'union n'ait pas de sommet communs. Mais si c'était le cas, le sommet commun contredirait la minimalité de u :



$$n_1(w) + n_2(w) < n_1(u) + n_2(u)$$

Le cycle est bien un plus court cycle car si ce n'est pas le cas, on contredirait à nouveau la minimalité de u.

Exercice 2.7

Le faire deux fois, avec Prim et Kruskal. Le poids min est de 12.

Exercice 2.8

Solution 1 : On reprend l'algorithme de Kruskal, mis à part qu'à chaque étape on ne rajoute l'arête e que si

1) elle ne crée pas de cycle
2) elle n'est pas la deuxième arête incidente à un sommet de U.

La complexité est la même que celle de Kruskal mais la démonstration est longue (il faut adapter celle de Kruskal en vérifiant ce qui se passe pour les sommets de U et à la fin vérifier qu'on a tout couvert. Ça se fait mais c'est long)

Solution 2 : Soit H le graphe obtenu en soustrayant les sommets de U

1) On cherche un arbre couvrant T_H de poids minimal de H
2) $\forall u \in U$, on le relie à T_H par l'arête de poids minimal entre u et $V(G \setminus U)$.

Complexité : celle de l'algo choisi pour l'étape 1. La deuxième étape est en effet en $\mathcal{O}(m)$

Validité : Le résultat est clairement un arbre couvrant tel que les sommets de U sont de degré 1. Soit T le résultat

Alors $$\omega(T) = \omega(T_H) + \sum_{u \in U} \omega(e_u)$$

où $$e_u = \underset{e \text{ de } u \text{ à } G \setminus U}{\operatorname{argmin}} \omega(e)$$

Soit S un arbre couvrant ~~de poids minimal~~ tel que les sommets de U sont des feuilles. Alors le graphe S_U obtenu en supprimant les sommets de U de S est encore un arbre, qui couvre H. D'où

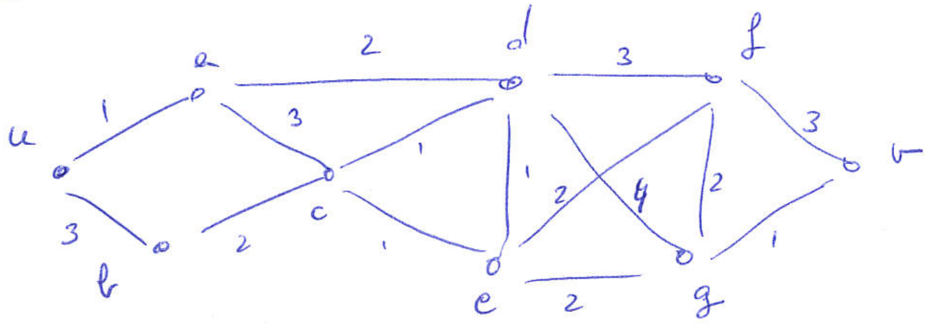
$$\omega(S_U) \geq \omega(T_H)$$

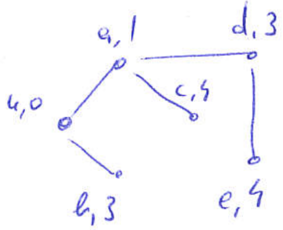
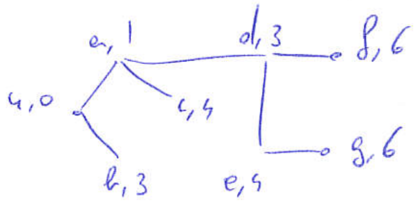
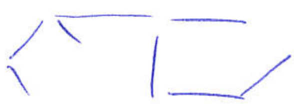
De plus, l'arête de S entre u et S_U est une arête entre u et $G \setminus U$ donc de poids supérieur ou égal à $\omega(e_u)$

Finalement $$\omega(S) \geq \omega(T_H) + \sum_{u \in U} \omega(e_u)$$
$$\geq \omega(T)$$

Donc T est bien minimal.

Exercice 2.9



Sommets considérés	Sommet ajouté	~~Sommet~~ Graphe découverts
{u, 0}	{u, 0}	{u, 0}
{(a, 1), (b, 3)}	{a, 1}	{(u,0), (a, 1)}
(b,3) (c,4) (d,3)	(b,3) (d,3)	(u,0), (a, 1), (b,3), (d, 3)
(c,4) (e,4) (f,6) (g,7)	(c,4) (e,4)	
(f,6) (g,6)	(f,6) (g,6)	
(v, 7)	(v, 7)	

Plus court chemin de longueur 7.

Exercice 2.10

Lancer Dijkstra à partir de chaque sommet et garder le plus grand.

Complexité en $n \times$ complexité de l'implementation choisie.

Au mieux $n(m + n \lg n)$