

Algorithmique (AL5)

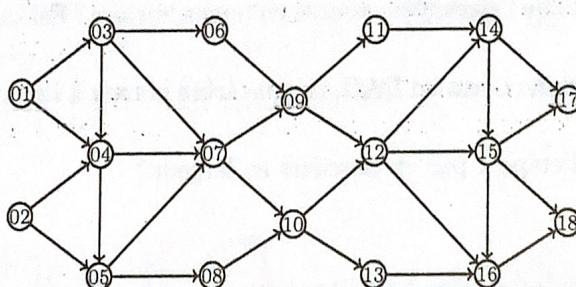
TD n° 4 : Tri topologique, Composantes fortement connexes

I) Tri topologique

Exercice 1 : Graphes Orientés Acycliques et Tri Topologique

Un graphe orienté acyclique (en anglais Directed Acyclic Graph, ou DAG) est un graphe orienté qui ne contient aucun cycle orienté.

- Montrer que si C est un cycle dans un graphe orienté G , alors dans toute exécution de parcours en profondeur de G , un des arcs de C sera un arc retour (on pourra utiliser la caractérisation des arcs retour par les fonctions *pre* et *post*). En déduire une modification de l'algorithme de parcours en profondeur qui permet de décider si un graphe orienté est un DAG.
- On rappelle qu'un *tri topologique* d'un graphe orienté $G = (V, E)$ est une énumération (s_1, s_2, \dots, s_n) des éléments de V telle que, pour tous i, j , on ait $(s_i, s_j) \in E \Rightarrow i < j$. Montrer que si un graphe orienté admet un tri topologique, alors il est nécessairement acyclique.
- Montrer la réciproque en prouvant que si on effectue un parcours en profondeur d'un DAG et qu'on trie les sommets par ordre décroissant de la valeur de la fonction $\text{post}(v)$, on obtient un tri topologique du graphe.
- Ceci fournit donc un algorithme pour construire le tri topologique d'un DAG. Quelle est sa complexité ?
- Appliquer l'algorithme sur le graphe suivant.



- Montrer que si un graphe orienté est tel que tout sommet a au moins un voisin entrant, alors il contient un cycle.
- On vient donc de montrer qu'un DAG possède nécessairement une source (un sommet de degré entrant 0). En se basant sur cette observation proposer un autre algorithme permettant de construire le tri topologique d'un DAG. Quelle est sa complexité (on suppose que le graphe est représenté par liste d'adjacence) ?
- On aurait aussi pu montrer que tout DAG possède nécessairement un puits (degré sortant 0) et en déduire un algorithme pour le tri topologique qui construit l'ordre par la fin. Aurait-on obtenu une meilleure complexité ?
- Améliorer la complexité de l'algorithme de la question 7 en construisant, puis en maintenant une liste de tous les sommets qui peuvent être la prochaine source choisie. Quelle est la nouvelle complexité ?

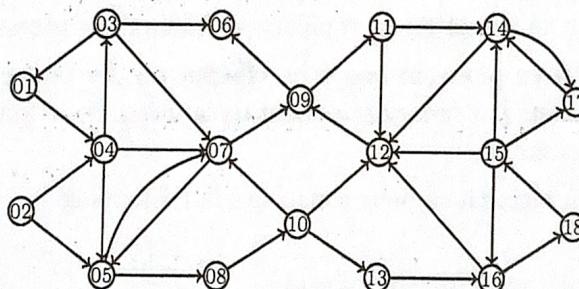
II) Composantes fortement connexes

Exercice 2 : Algorithme de calcul de CFC

- Rappelez ce qu'est une composante fortement connexe dans un graphe dirigé.

On rappelle l'*algorithme de Kosaraju* qui permet de calculer les composantes fortement connexes d'un graphe dirigé :

- Faire un parcours en profondeur sur G pour obtenir une liste L de sommets de G triés par ordre décroissant de date *post* ;
 - Construire G^t (le graphe transposé de G) ;
 - Faire un parcours en profondeur sur G^t en choisissant les sommets par ordre décroissant de la valeur *post* obtenue à l'étape précédente.
 - Chaque arbre de la forêt obtenue couvre exactement une CFC.
- Quelle est la relation entre cet algorithme et l'algorithme vu en cours ?
 - Appliquez l'algorithme ci-dessus au graphe suivant. En cas de choix arbitraire, on commencera par les sommets de numéro plus petit.



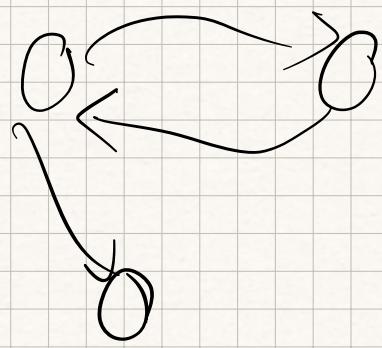
- Comment faire pour que l'algorithme tourne en temps linéaire ? Est-ce surprenant qu'on puisse le faire ?
- Montrez que si le graphe G est un DAG, chaque arbre obtenu à l'étape 3 est trivial (n'a qu'un seul sommet).
- Peut-on remplacer l'étape 1 par un parcours en largeur ?
- Et l'étape 3 ?

Exercice 3 : Chemins et cycles hamiltoniens

Définitions :

- On appelle *chemin hamiltonien* sur un graphe (dirigé ou non) un chemin passant par chaque sommet du graphe une fois et une seule.
- Un *circuit hamiltonien* est un cycle passant par chaque sommet du graphe une fois et une seule.
- Un *tournoi* est un graphe dirigé tel que pour toute paire de sommets u, v soit $(u, v) \in E$ soit $(v, u) \in E$.

- Si un graphe dirigé possède un cycle hamiltonien, que dire de ses composantes fortement connexes ?
- Réciproquement, si un graphe est fortement connexe, est-il nécessaire qu'il ait un cycle hamiltonien ?
- On a vu exercice 2 qu'un graphe orienté acyclique (DAG) admet un tri topologique. Montrer qu'un DAG n'a qu'un seul tri si et seulement s'il possède un chemin hamiltonien.
- Montrer qu'un tournoi G admet un chemin hamiltonien.
- Montrer qu'un tournoi fortement connexe possède un cycle hamiltonien.



graphe dirigé

Un graphe orienté acyclique (en anglais Directed Acyclic Graph, ou DAG) est un graphe orienté qui ne contient aucun cycle orienté.

On rappelle qu'un *tri topologique* d'un graphe orienté $G = (V, E)$ est une énumération (s_1, s_2, \dots, s_n) des éléments de V telle que, pour tous i, j , on ait $(s_i, s_j) \in E \Rightarrow i < j$.

Connexe : graphe non dirigée

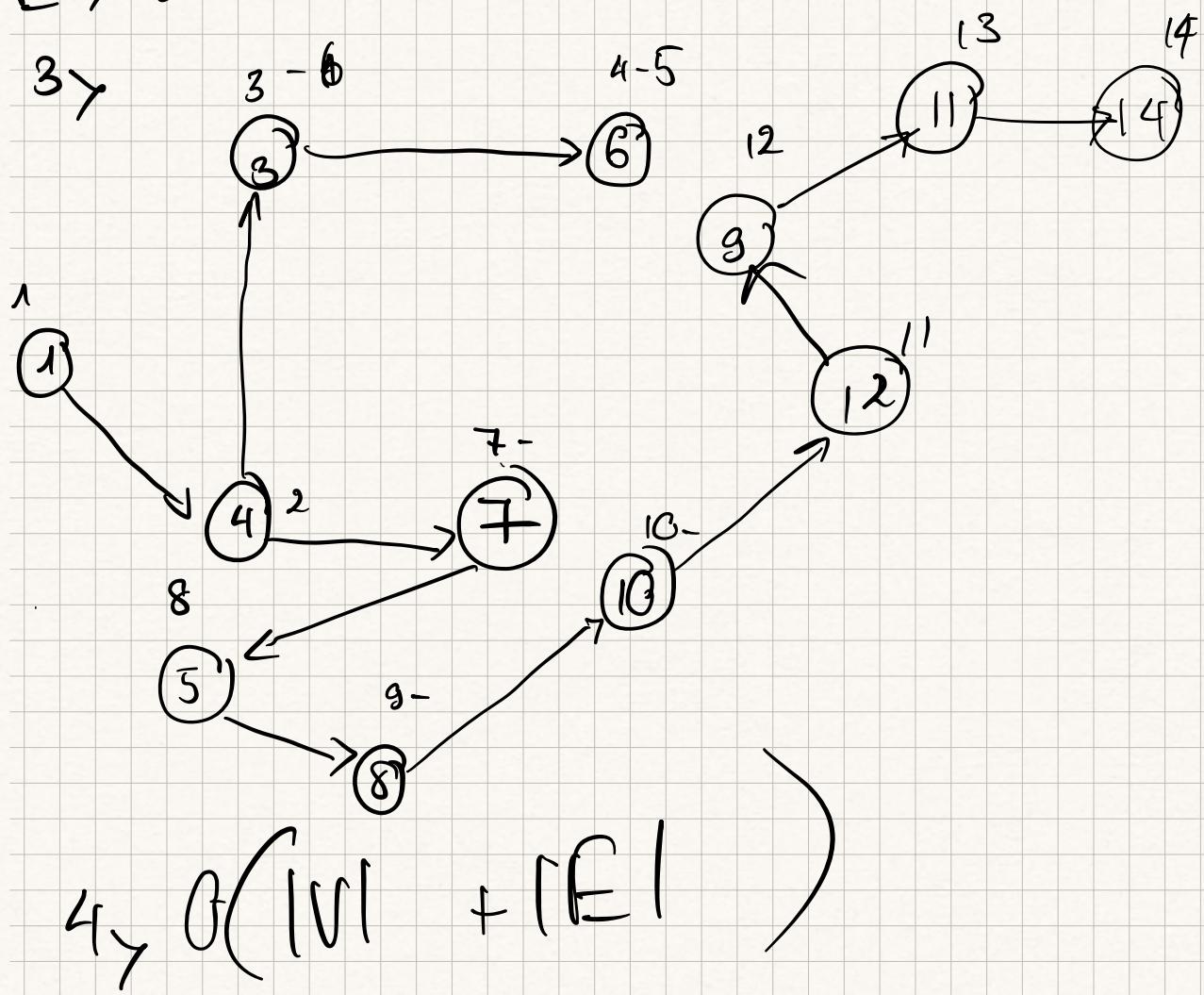
1. Lorsqu'on fait un parcours en profondeur, si on rencontre un arc qui pointe vers un sommet d_j visité (et qui n'est pas encore terminé), alors cet arc est un arc retour, et le graphe a un cycle. Si à la fin du parcours, on a trouvé aucun arc retour, alors le graphe est un DAG

2. Oui, uniquement les graphes acycliques (DAG) peuvent être triés topologiquement. Si un graphe a un cycle, un tel topologique n'est pas possible

3

4γ

Eγ 2



II) Composantes fortement connexes

Exercice 2 : Algorithme de calcul de CFC

1. Rappellez ce qu'est une composante fortement connexe dans un graphe dirigé.

On rappelle l'*algorithme de Kosaraju* qui permet de calculer les composantes fortement connexes d'un graphe dirigé :

1. Faire un parcours en profondeur sur G pour obtenir une liste L de sommets de G triés par ordre décroissant de date *post* ;
2. Construire G^t (le graphe transposé de G) ;
3. Faire un parcours en profondeur sur G^t en choisissant les sommets par ordre décroissant de la valeur *post* obtenue à l'étape précédente.
4. Chaque arbre de la forêt obtenue couvre exactement une CFC.

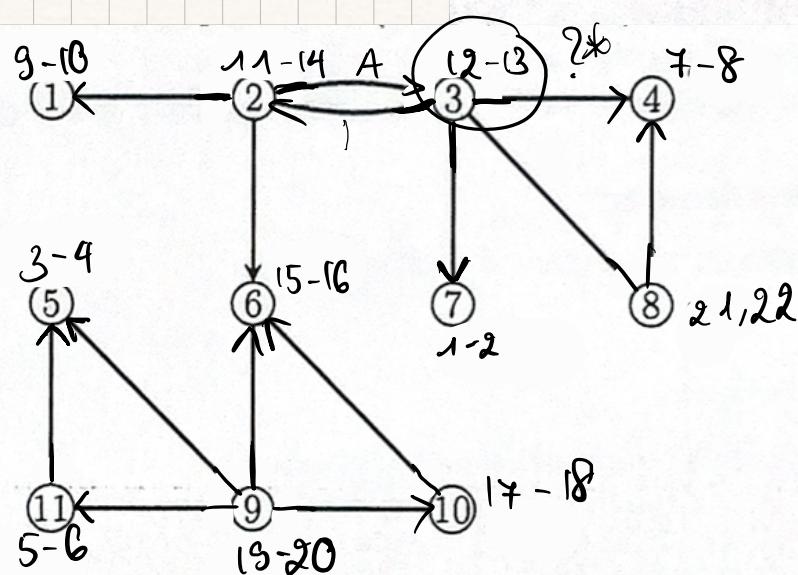
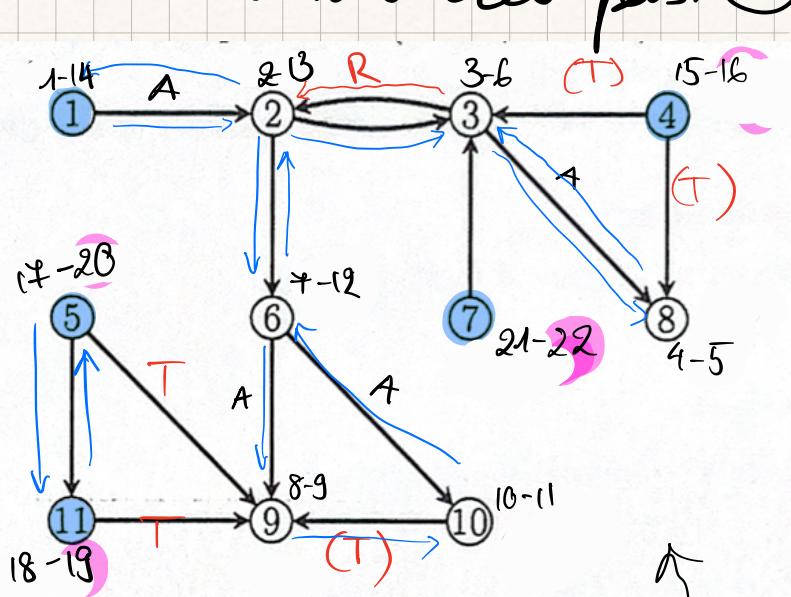
1^y Un DFS sur G on récupère la liste de sommets triés par ordres décroissantes des post (L)

$$L = 7, 5, 11, 4, 1, 2, 6, 10, 9, 3, 8$$

pré - post plus grand \rightarrow plus petit

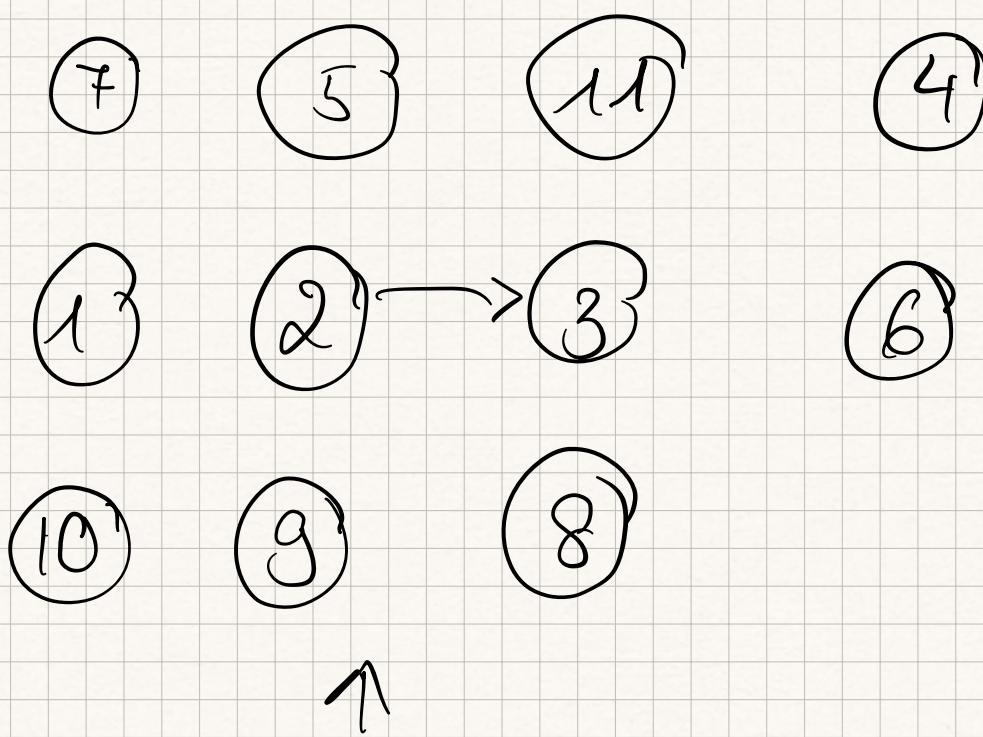
DFS

Dans



* On transpose car pour les flèches origine toutes les flèches changent de sens sauf 2 \leftarrow 3
Donc on transpose ici

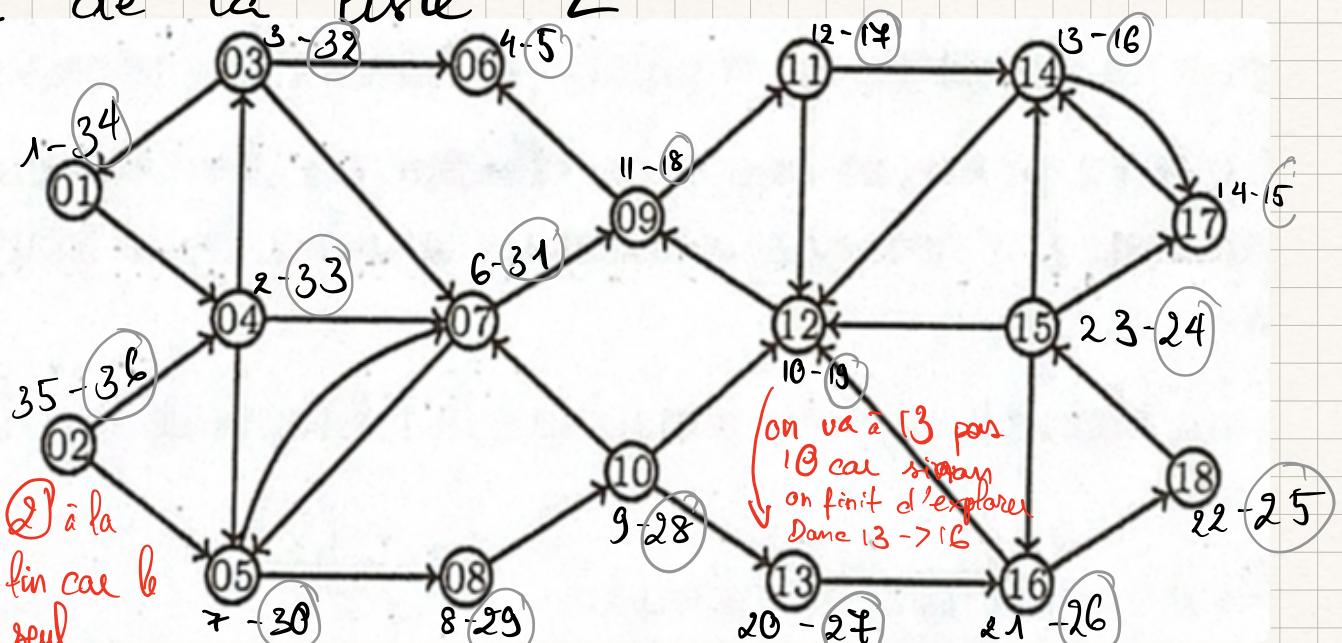
2) Construire G^k ($a \rightarrow n$ deoient $v \rightarrow u$)



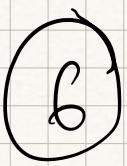
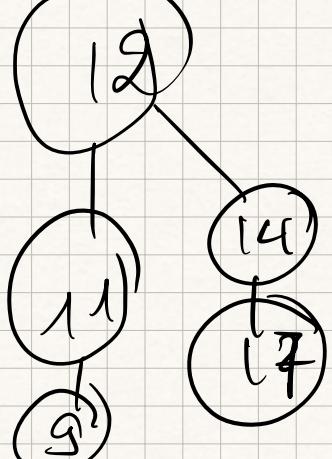
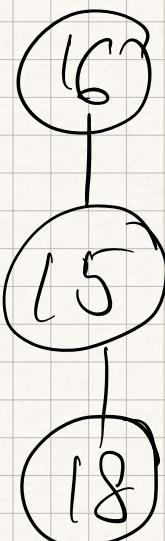
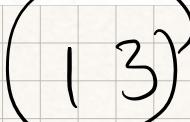
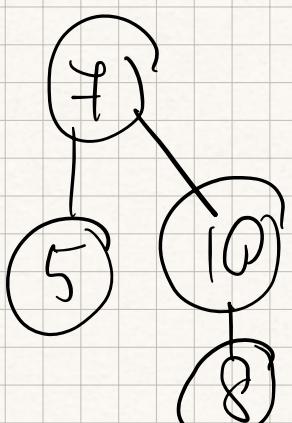
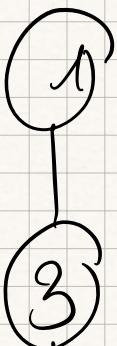
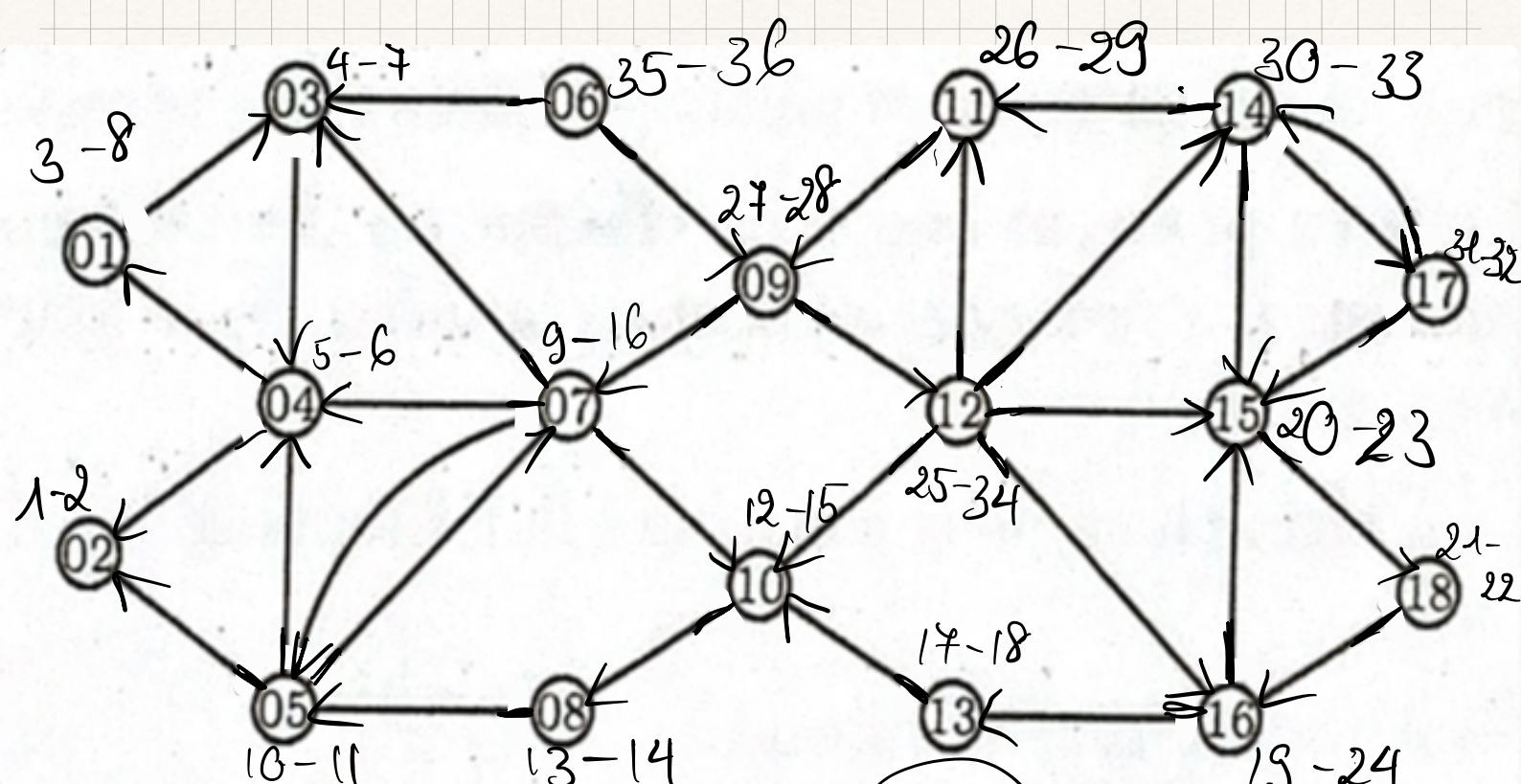
4) Est ce que à l'étape 3-4 la DFS était important ?

La seule contrainte est que notre parcours obéisse à l'ordre de L . Un parcours en largeur convient aussi

3) Fais un DFS sur G^t , choix des sommets ordre de la liste L



$L = \{ [2, 1, 4, 3], [4, 5, 8, 10], [13, 16, 18, 15], [12, 9, 11] \}$
 14, 17, 6



A_y Arbre de la forêt obtenue = les CFC

DFS : liste d'adjacente

Représente l'ordre de post

via une liste chaînée (ajout en temps c)

$G \rightarrow G^K$: temps linéaire

5x Si G est un DAG

Si on parcourt une racine u

S'il existe $u \rightarrow v$ une arête de u vers v
dans G^K

il y avait $v \rightarrow u$ dans G

$\text{post}(u) < \text{post}(v)$

dans v
n'a jamais

Rappel : DAG \rightarrow pas d'arc retour DES de parcs
avant