Soit $G = (S, A)$ un graphe orienté, $n = |S|$ et $p = |A|$.

I Ordre de parcours

Définition : Parcours en profondeur préfixe, postfixe

On peut ordonner les sommets de G lors d'un parcours en profondeur (DFS) complet :

- Ordre préfixe : on ajoute un sommet au début de son appel récursif (avant ses voisins).
- Ordre postfixe : on ajoute un sommet à la fin de son appel récursif (après ses voisins).

Exercice 1.
Montrer que l'inverse d'une liste de parcours préfixe n'est pas forcément un parcours postfixe.

Définition : Ordre topologique

Un ordre topologique (ou : tri topologique) de G est une liste $v_1, v_2, \dots, v_n$ des sommets de G telle que si $(v_i, v_j) \in A$, alors $i < j$.

Théorème

Si G est acyclique alors l'inverse d'une liste de parcours postfixe est un ordre topologique de G.

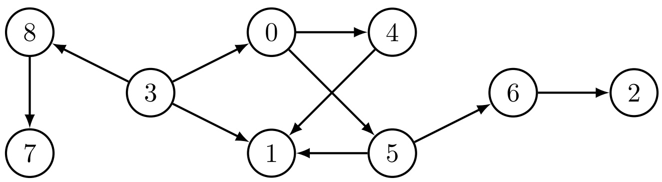
Preuve :

```
let postfixe_inverse (g : int list array) =
    let n = Array.length g in
    let vus = Array.make n false in
    let l = ref [] in
    let rec dfs u =
        if not vus.(u) then (
            vus.(u) <- true;
            List.iter (fun v -> dfs v) g.(u);
            l := u :: !l
        ) in
    for u = 0 to n - 1 do
        dfs u
    done;
    !l
```

Complexité : $O(n + p)$ pour un graphe représenté par liste d'adjacence car chaque arête est parcourue au plus deux fois (O($2p$) = O(p)) et chaque sommet est visité une fois (O(n)).
Remarque : on aurait aussi pu ajouter un `List.rev` à la fin.

Exercice 2.

1. Donner le parcours postfixe du graphe ci-dessous, en choisissant le sommet de plus petit numéro s'il y a plusieurs choix possibles.
2. En déduire un ordre topologique de ce graphe.



Théorème

G possède un ordre topologique si et seulement s'il est acyclique.

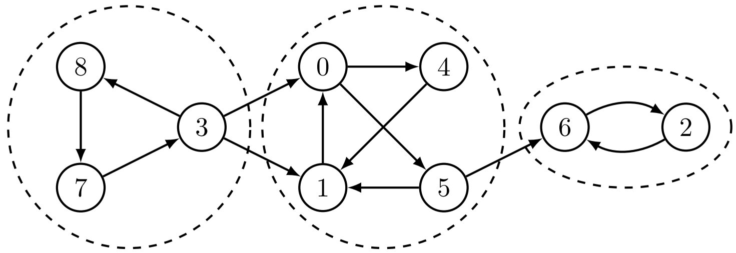
Preuve :

Remarque : Si G n'est pas acyclique, l'inverse d'un parcours postfixe donne un ordre topologique des composantes fortement connexes de G.

Exercice 3.
Soit G acyclique pondéré par $w : A \longrightarrow \mathbb{R}$ et $u \in S$. On suppose G représenté par liste d'adjacence. Montrer que l'on peut calculer la distance $d(u, v)$ de u à tous les sommets $v \in S$ en $O(n + p)$.

II Algorithme de Kosaraju

L'algorithme de Kosaraju permet de trouver les composantes fortement connexes d'un graphe orienté.



Idée : Si on fait un parcours de graphe depuis 6 ou 2, on obtient la composante fortement connexe $\{2, 6\}$.
On lance un DFS depuis chaque sommet dans l'ordre inverse du parcours postfixe de G^T (graphe obtenu en inversant les arcs de G). L'ensemble des sommets visités à chaque DFS forme une composante fortement connexe.

Algorithme de Kosaraju

Entrée : Un graphe connexe $G = (S, A)$
Sortie : Les composantes fortement connexes de G

$G^T \leftarrow$ graphe transposé de G
$L \leftarrow$ liste inverse du parcours postfixe de G^T
$C \leftarrow$ tableau de taille n initialisé à -1
$k \leftarrow 0$
Pour $u \in L$ **:**
  Si $C[u] = -1$ **:**
    $C[u] \leftarrow$ k
    DFS(u)
    $k \leftarrow$ k + 1
Renvoyer C `/*` $C[i]$ `= numéro de la composante fortement connexe de` i `*/`

Complexité de l'algorithme de Kosaraju :

- Calcul de G^T : ______
- Liste inverse du parcours postfixe de G^T : ______
- Initialisation de C : ______
- DFS complet : ______

D'où une complexité totale en : ______

Exercice 4.
Appliquer l'algorithme de Kosaraju au graphe ci-dessus.

Exercice 5.

1. Écrire une fonction `transpose : int list array -> int list array` qui prend en argument un graphe G et renvoie son graphe transposé G^T.
2. Écrire une fonction `kosaraju : int list array -> int array` qui prend en argument un graphe G et renvoie un tableau C tel que $C[i]$ est le numéro de la composante fortement connexe de i.