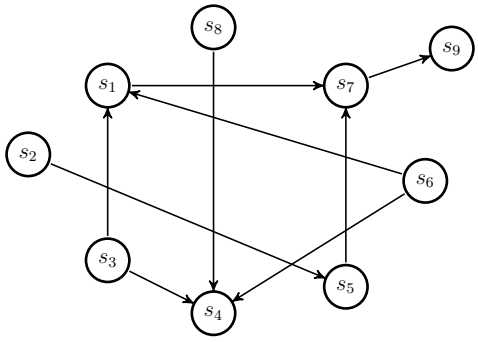
Exercice 1

Modifier l'algorithme `PP` pour calculer la longueur du plus long chemin dans un graphe orienté sans circuit.

Exercice 2

Appliquer l'algorithme de *tri topologique* basé sur l'algorithme `PP` au graphe ci-dessous en suivant l'ordre lexicographique :



Exercice 3

On considère l'algorithme suivant :

```
1 pour chaque sommet u de X[G] faire:
2     deg_entrant[u] <- calculer_degre_entrant(u)
3 tant qu'il existe un sommet u tel que deg_entrant[u] == 0 faire:
4     afficher(u)
5     deg_entrant[u] <- deg_entrant[u] - 1
6     pour chaque v de Adj[u] faire:
7         deg_entrant[v] <- deg_entrant[v] - 1
```

i. Démontrer que tout graphe orienté fini, non vide, sans circuit possède un sommet sans prédécesseur.

ii. Que fait cet algorithme lorsqu'on l'applique à un graphe orienté sans circuit ?

iii. Comment l'utiliser pour détecter un circuit ?

iv. Quelle est sa complexité suivant qu'on utilise, pour représenter le graphe, une matrice d'adjacence ou des listes de successeurs ?

Rappel :

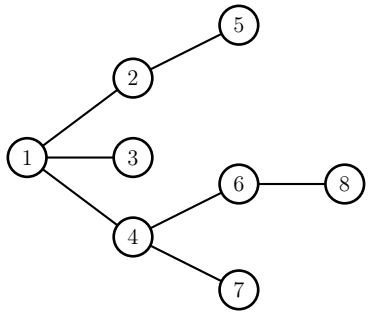
```
0   PP(G)
1     pour chaque sommet u de X faire
2         couleur[u] <- BLANC
3         pere[u] <- nil
4     temps <- 0
5     pour chaque sommet u de X faire
6         si couleur[u] = BLANC
7             alors Visiter_PP(u)
```

```
0   Visiter_PP(u)
1     couleur[u] <- GRIS
2     d[u] <- temps <- temps + 1
3     pour chaque v de Adj[u] faire
4         si couleur[v] = BLANC
5             alors pere[v] <- u
6                   Visiter_PP(v)
7     couleur[u] <- NOIR
8     f[u] <- temps <- temps + 1
```

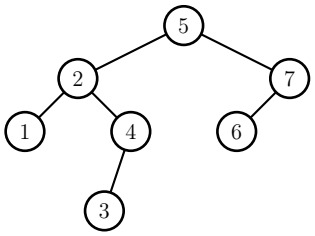
Exercices complémentaires non traités en TD

Exercice 4

Un arbre `A` sera caractérisé par sa racine, `racine(A)`, et la liste `fils(A)` des fils de la racine (qui sont des arbres).



Arbre A_1



Arbre B_1

i. Modifier les algorithmes `PP` et `PL` de façon à les adapter aux arbres. Les algorithmes devront remplir un tableau `p[s]` donnant l'ordre de visite du sommet `s` dans le parcours. Pour `PL` remplir aussi un tableau `d[s]` donnant la distance à `racine(A)`.

Lors d'un parcours en profondeur d'un arbre chaque sommet est rencontré une première fois, avant tous ses fils, et une dernière fois, après tous ses fils, ce qui induit les deux ordres classiques sur les sommets : *préfixe* (ou préordre) et *postfixe* (ou postordre).

ii. Indiquer les ordres préfixe et postfixe des sommets de l'arbre A_1 dans un parcours en profondeur :

Un arbre binaire `B` est un arbre dont chaque sommet a au plus deux successeurs. Il sera représenté par `racine(B)`, et ses successeurs `fils_gauche(B)`, `fils_droit(B)` (qui sont des arbres).

Dans un parcours `PP` d'un arbre binaire, chaque sommet est rencontré trois fois (avant son fils gauche, entre les deux fils et après son fils droit), ce qui induit un troisième ordre classique sur les sommets : l'ordre *infixe* (ou symétrique).

iii. Donner l'ordre préfixe, postfixe et infixe de l'arbre binaire B_1.

iv. Décrire trois versions de l'algorithme `PP` pour un arbre binaire de façon à remplir le tableau `p[s]` suivant les trois ordres.