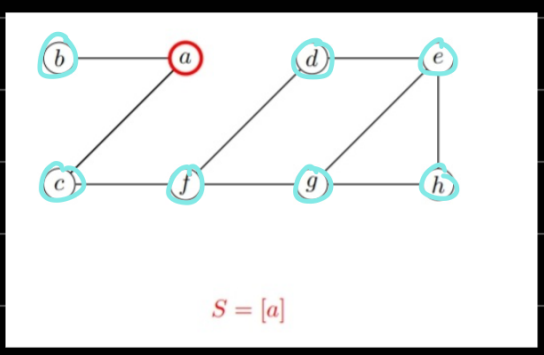
Parcours en profondeur (DFS):

Entrées : graphe $G = (V, E)$ et sommet $r \in V$
début
  créer pile(S)
  pour tous les $u \in V$ **faire**
    marqué$[u] \leftarrow$ False
  empiler(S, r)
  tant que $S \neq \emptyset$ **faire**
    $u \leftarrow$ dépiler(S)
    marqué$[u] \leftarrow$ Vrai
    pour tous les $uv \in E$ **faire**
      si marqué$[v]$ = Faux **alors**
        marqué$[v] \leftarrow$ Vrai
        empiler(S, v)

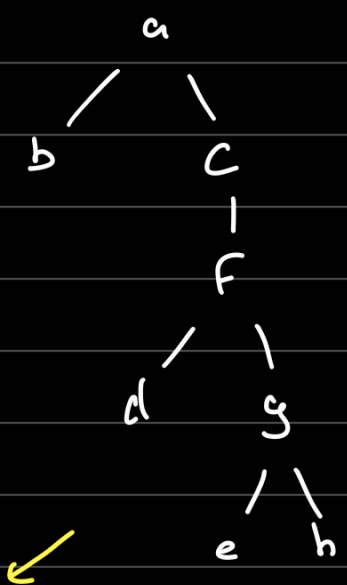
→ pas nécessaire

C'est le même algo que le parcours en largeur, mais avec une pile au lieu d'une file.

Exemple:



⇒



Arbre résultant du parcours en profondeur.

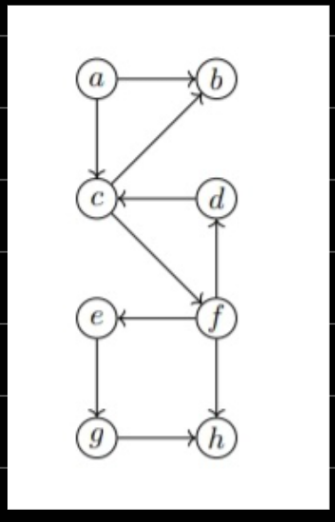
Version récursive :

```
Procédure explorer(G, u) :
    marqué[u] ← Vrai
    pour tous les (u, v) ∈ E(G) faire
        si marqué[v] = Faux alors
            explorer(G, v)
```

→ permet de gérer le cas des graphes non connexes

Si G est connexe, explorer(G, s) va explorer tout le graphe.

DFS dans les graphes orientés :




```
Entrées : graphe G = (V, E) et sommet r ∈ V
début
    créer pile(S)
    pour tous les u ∈ V faire
        marqué[u] ← False
    empiler(S, r)
    tant que S ≠ ∅ faire
        u ← dépiler(S)
        marqué[r] ← Vrai
        pour tous les (u, v) ∈ E faire
            si marqué[v] = Faux alors
                marqué[v] ← Vrai
                empiler(S, v)
```

→ On tient compte du sens.

Théorème des parenthèses :

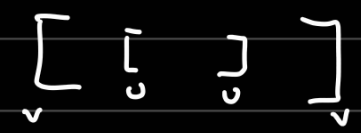
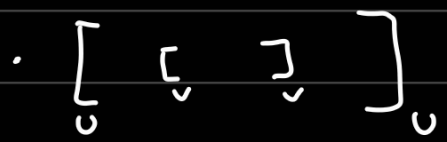
Soient u, v deux sommets quelconques

Alors, soit $[pre(u), post(u)] \subset [pre(v), post(v)]$

soit $[pre(u), post(u)] \supset [pre(v), post(v)]$

soit $[pre(u), post(u)] \cap [pre(v), post(v)] = \emptyset$

Exemple :

- 
- 
-