28 – Parcours en profondeur d'un graphe

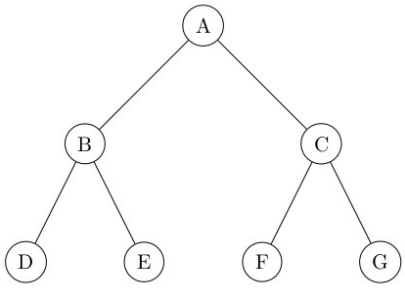
Parcours en largeur : utilise une _____ **Parcours en profondeur** : utilise une _____

1 - Rappel : parcours en profondeur d'un arbre

Nous en avons vu trois :

1. Le parcours **préfixe** (RGD) : on lit la racine puis sous-arbre gauche puis droite.
2. Le parcours **infixe** (GRD) : on part dans le sous-arbre gauche, la racine puis droite.
3. Le parcours **suffixe** ou postfixe (GDR) : sous-arbre gauche,droite puis racine.

Exemple en préfixe sur cet arbre binaire :



2 - Parcours en profondeur d'un graphe

2.1 Principe du parcours en profondeur dans un graphe G = (S, A)

Nous allons utiliser les **attributs suivants** :

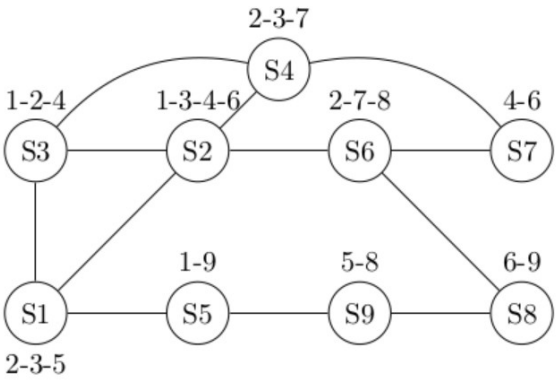
- un attribut **couleur** pour préciser l'état du sommet lors de l'exploration :
  1. **BLANC** : non découvert
  2. **GRIS** : le sommet a été visité
  3. **NOIR** : le sommet a été traité
- un attribut **parent** qui contient le sommet depuis lequel on a découvert ce sommet. VIDE au départ de l'exploration.

Algorithme du parcours_en_profondeur :

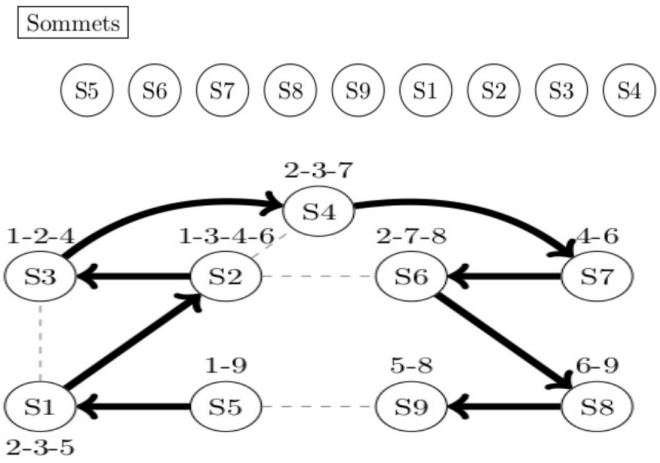
1. **INITIALISER** : Pour chaque sommet **u** de S :
   - on place BLANC dans couleur et
   - on place VIDE dans parent
2. **PARCOURIR** : Pour chaque sommet **u** de S :
   - Si **u** est BLANC
     - **visiter_en_profondeur(u)**

Algorithme de visiter_en_profondeur(u)

1. **couleur** de **u** passe à GRIS
2. Pour chaque sommet **v** de la liste d'adj. de **u** :
   - Si **v** est BLANC
     - **parent** de **v** devient **u**
     - **visiter_en_profondeur(v)**
3. **couleur** de **u** passe à NOIR puisqu'on a testé toutes ses arêtes ou arcs
4. 2.2 Exemple de déroulement de l'algorithme de parcours en profondeur



2.2 Exemple de déroulement de l'algorithme de parcours en profondeur (voir votre feuille)

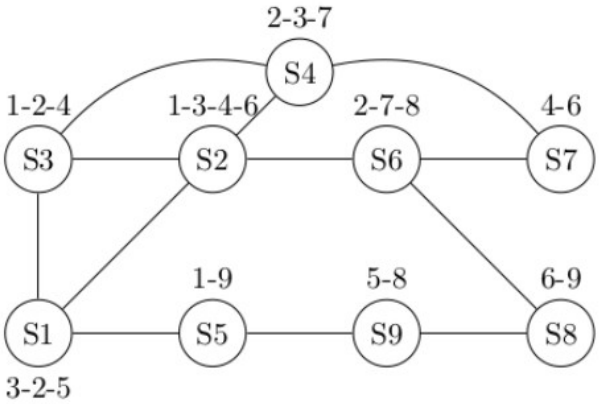


Arbre de parcours →

✎ **01°** Sur l'Arbre de parcours obtenu, quel est l'enchaînement de sommets obtenu de S5 à S6 ?

✎ **02°** S'agit-il d'un chemin minimum en terme de nombre d'arêtes ?

✎ **03°** Regardons si l'ordre des sommets dans les listes d'adjacence a une importante. Réaliser le parcours en profondeur sur ce "nouveau" graphe. Comparer votre Arbre de parcours à l'ancien.



Sommets

✎ **04°** On utilise une fois chaque sommet comme centre de recherches. Combien de fois utilise-t-on les arêtes ?

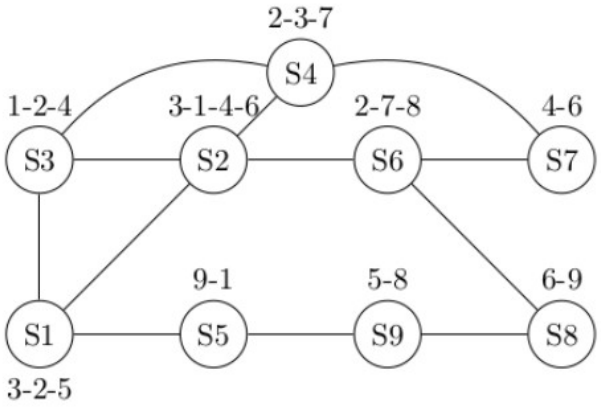
✎ **05°** Nous allons chercher la complexité de cet algorithme en prenant comme référence le nombre de fois où on teste si un sommet est BLANC. Notons a le nb d'arêtes. Quelle est la complexité ?

Le coût total d'exécution dépend en réalité du nombre de sommets et du nombre d'arêtes. A cause de la boucle initiale sur les sommets, l'algorithme assure qu'on explore chaque sommet même en cas de graphe non convexe.

On peut donc écrire $\Theta(s+a)$

On rappele que pour l'algorithme de parcours en largeur, on écrit simplement $O(s+a)$ car si le graphe est non convexe, on ne passe pas forcément par tous les sommets et toutes les arêtes.

✎ **06°** D'autres listes d'adjacences : vous devriez voir qu'on peut avoir un arbre non filiforme même avec un graphe connexe.

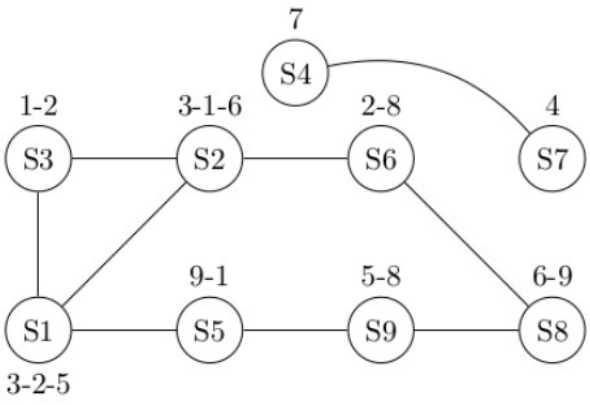


Sommets

✎ **07°** Cet algorithme peut-il trouver le plus court chemin en terme d'arêtes ? A quoi peut-il servir ?

✎ **08°** Quelle va être la forme de la forêt obtenue avec ce graphe ? (la seule différence vient du fait que les sommets 4 et 7 sont isolés des autres)

Sommets



3 - Algorithme de parcours en profondeur

ALGORITHME PARCOURS EN PROFONDEUR

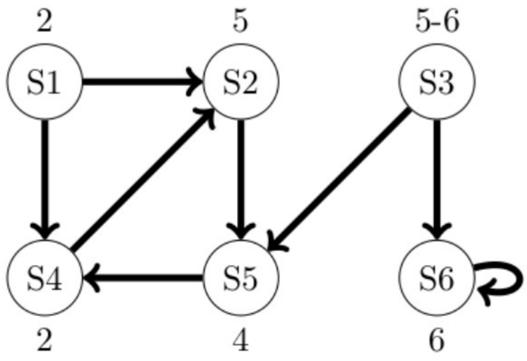
```
→ on initialise tous les sommets
POUR chaque sommet u appartenant au graphe g
    u.couleur ← BLANC
    u.parent ← VIDE
Fin Pour

→ on lance le traitement des sommets un par un
POUR chaque sommet u appartenant au graphe g
    SI u.couleur est BLANC
        → si on arrive ici, c'est que le sommet n'a pas encore été découvert
        visiter_en_profondeur(u)
    Fin Si
Fin Pour

Renvoyer VIDE (∅)
```

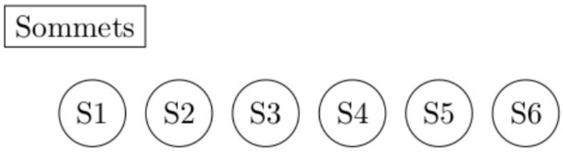
ALGORITHME de visiter_en_profondeur(u)

```
u.couleur ← GRIS
POUR chaque sommet v de la liste d'adjacence du sommet u
    SI v.couleur est BLANC
        → si on arrive ici, c'est que le sommet n'a pas encore été visité
        v.parent ← u.
        visiter_en_profondeur(v)
    Fin Si
Fin Pour
→ ici, tous les sommets adjacents ont été tentés
u.couleur ← NOIR
Renvoyer VIDE (∅)
```

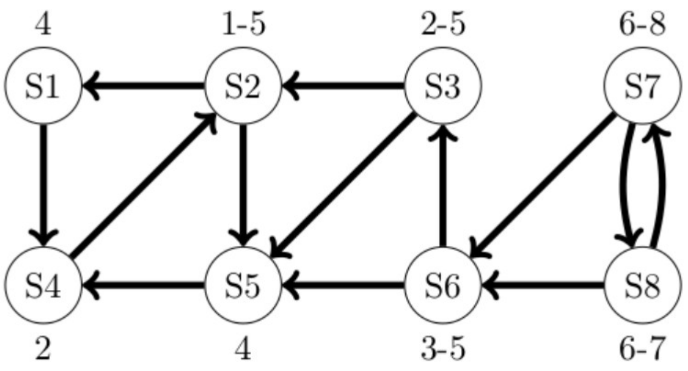
4 - Exercices

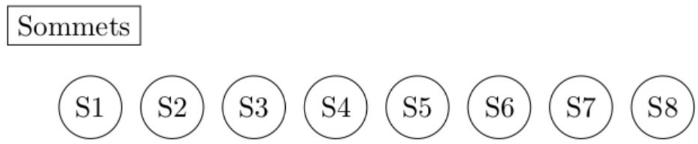
✎ **09°** Explorer en profondeur ce graphe orienté pour tracer son arbre ou sa forêt de parcours.



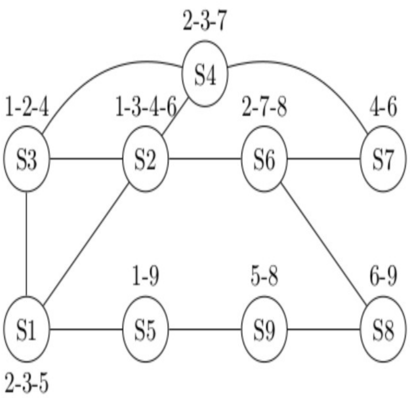


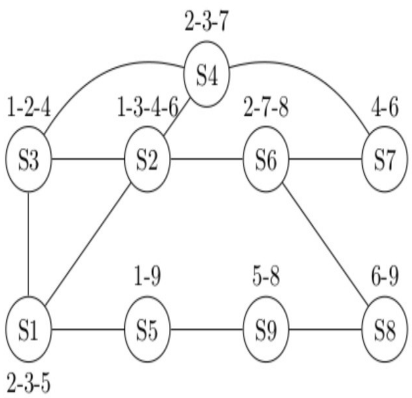
✎ **10°** Explorer en profondeur ce graphe orienté pour tracer son arbre ou sa forêt de parcours.

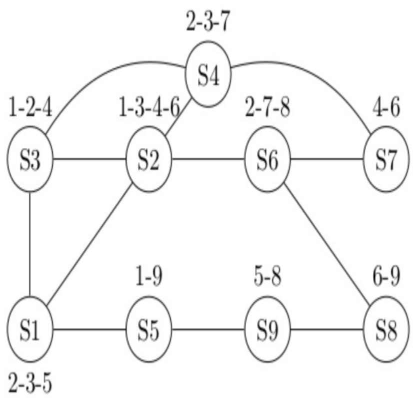


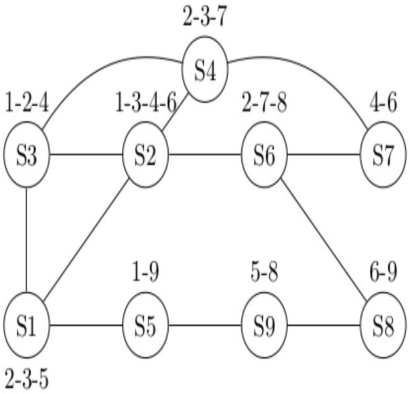


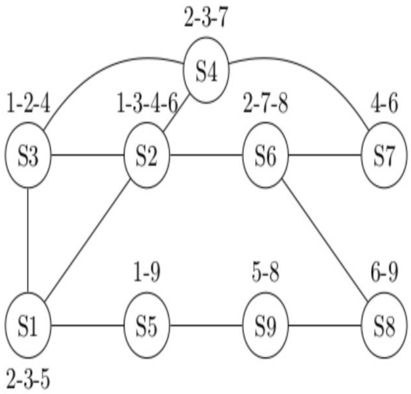
<u>Etapes du déroulement du graphe de l'exemple :</u>

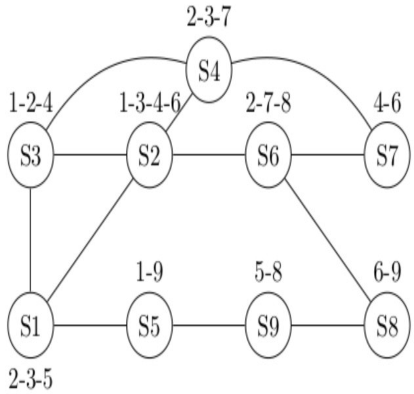


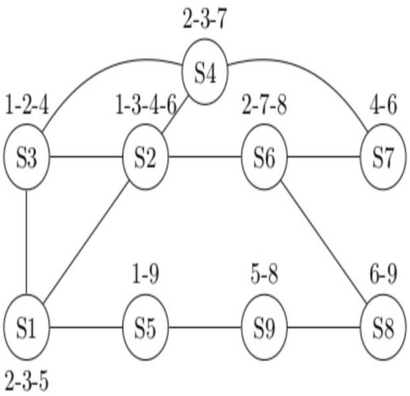


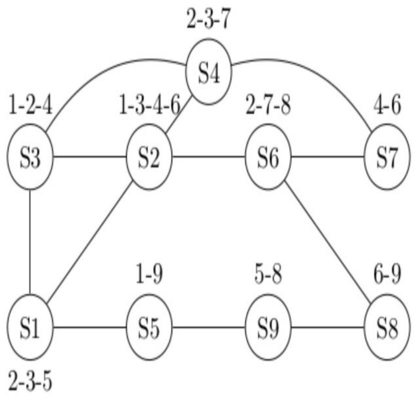


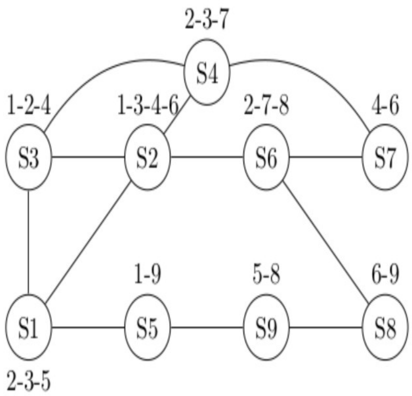

2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5

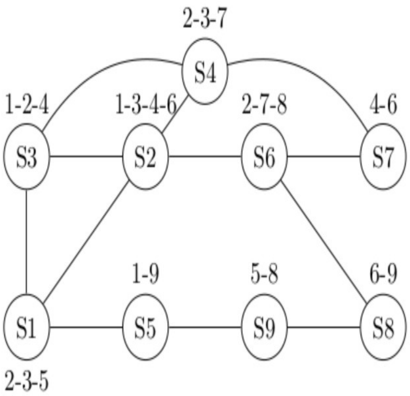

2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5

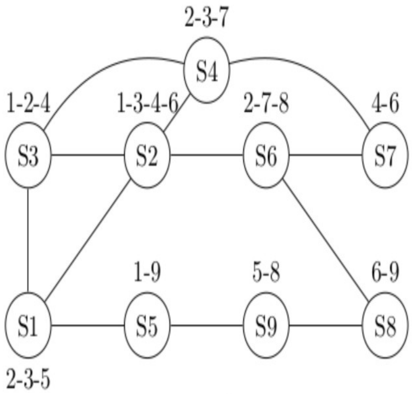

2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5

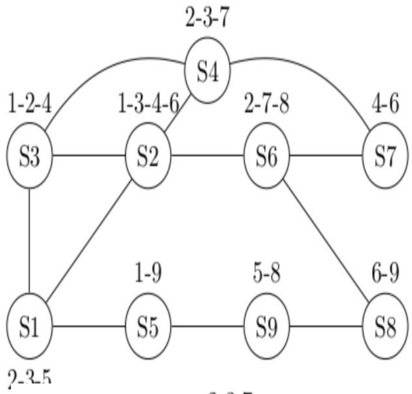

2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5

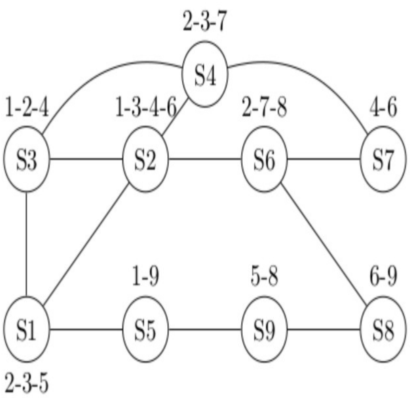

2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5

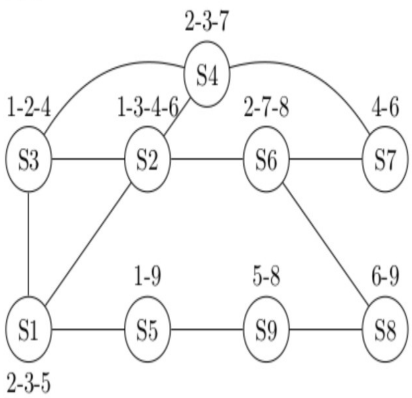

2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5

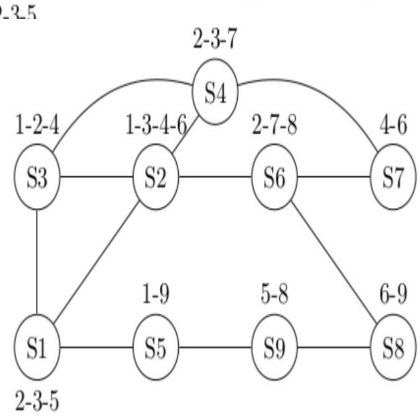

2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5


2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5


2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8


2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5


2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5


2-3-7
S4
1-2-4
1-3-4-6
2-7-8
4-6
S3
S2
S6
S7
1-9
5-8
6-9
S1
S5
S9
S8
2-3-5

PARCOURS EN PROFONDEUR VERSION 1 : avec les couleurs

```
# CONSTANTES ==========================================================================
BLANC = 0
GRIS = 1
NOIR = 2

# FONCTIONS ===========================================================================
def parcours_profondeur(g:'dict[int, list]') -> dict:
    """Renvoie les informations (les parents) du parcours en profondeur"""
    # INITIALISATION
    parcours = {}                     # Création du dictionnaire (qui sera un dict de dict)
    for u in g.keys():                # Pour chaque sommet du graphe
        parcours[u] = {}              # La clé u de parcours a comme valeur associée un dict
        parcours[u]['couleur'] = BLANC # Initialisation de la couleur
        parcours[u]['parent'] = None   # Initialisation du parent
    # PARCOURIR
    for u in g.keys():                          # Pour chaque sommet du graphe
        if parcours[u]['couleur'] == BLANC:     # S'il est encore Non découvert
            visiter_profondeur(u, g, parcours) # Visite le sommet
    return parcours

def visiter_profondeur(u:int, g:'dict[int, list]', parcours:dict) -> None:
    """Visite un sommet u en particulier"""
    parcours[u]['couleur'] = GRIS             # Signale que u est découvert
    for v in g[u]:                            # Pour chaque sommet v de la liste d'adj de u
        if parcours[v]['couleur'] == BLANC:   # Si v est non découvert
            parcours[v]['parent'] = u         # u est son parent
            visiter_profondeur(v, g, parcours) # on lance la visite de v
    parcours[u]['couleur'] = NOIR             # u est totalement exploré

def donner_chemin(sommet_initial, sommet_final, parcours:dict) -> list:     voir le site

# PROGRAMME PRINCIPAL =================================================================
if __name__ == '__main__':
    graphe = { 5: [9, 1], 6: [2, 7, 8], 7: [4, 6], 8: [6, 9], 9: [5, 8], 1: [3, 2, 5],
               2: [3, 1, 4, 6], 3: [1, 2, 4], 4: [2, 3, 7] }
    parcours = parcours_profondeur(graphe)
    print(parcours)
    print("9 vers 2 : ", donner_chemin(9, 2, parcours))
    print("9 vers 5 : ",donner_chemin(9, 5, parcours))
```

VERSION 2 : sans couleur

```
def parcours_profondeur(g):
    # INITIALISATION
    d = []

    # PARCOURIR
    for u in g.keys():
        if not(u in d):
            visiter_profondeur(u, g, d)
    return d

def visiter_profondeur(u, g, d):
    """Visite un sommet u en particulier"""
    d.append(u)
    for v in g[u]:
        if not(v in d):
            visiter_profondeur(v, g, d)
    # u est exploré
```

VERSION 3 : impératif (Pile)

```
def parcours_profondeur(g):
    # INITIALISATION
    d = [] # Tableau des sommets découverts

    # PARCOURIR
    for u in g.keys():

        pile = []
        pile.append(u)

        # Equivalent impératif de visiter(u)
        while pile: # TQ pile pas vide
            s = pile.pop() # On dépile
            if not(s in d):
                d.append(s) # s est découvert
                    for v in g[s]:
                        pile.append(v)
    return d
```