

# Algo 28 - Parcours en profondeur d'un graphe



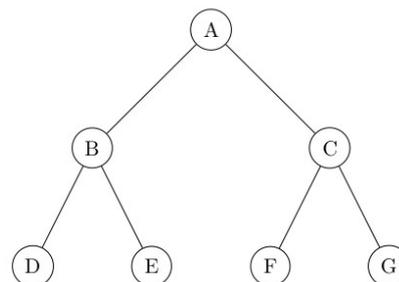
**Parcours en largeur** : utilise une \_\_\_\_\_ **Parcours en profondeur** : utilise une \_\_\_\_\_

## 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 [version en français]

### 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.

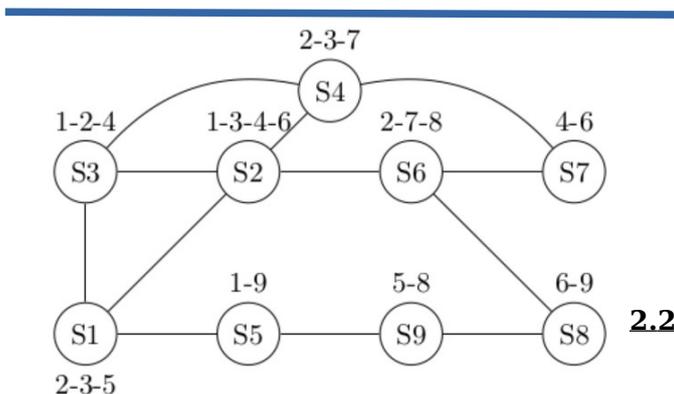
**Algorithme du parcours\_en\_profondeur :**

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

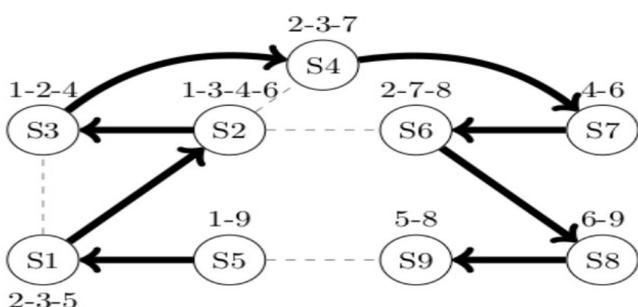
### 2.2 Exemple de déroulement de l'algorithme de parcours en profondeur (voir votre feuille)



Sommets



Arbre de parcours



### 3 - Algorithme de parcours en profondeur [version pseudo-code]

#### 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 ( $\emptyset$ )

#### 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

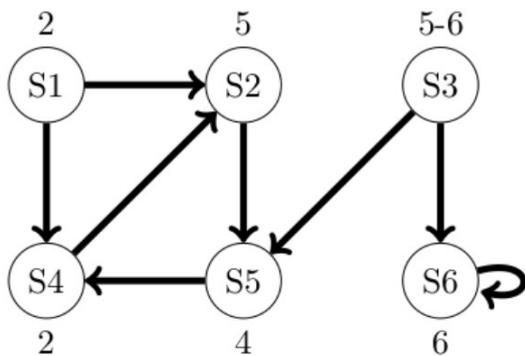
→ ici, tous les sommets adjacents ont été tentés

**u.couleur** ← NOIR

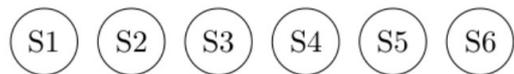
Renvoyer VIDE ( $\emptyset$ )

### 4 - Exercices

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



Sommets



### 5 - Une implémentation en Python

```
1 def parcours_profondeur(g:'dict[int, list]') -> list:
2     """Renvoie la tableau des sommets découverts lors du parcours en profondeur"""
3
4     # INITIALISATION
5     decouverts = [] # Tableau des sommets découverts
6
7     # PARCOURIR
8     for u in g.keys(): # Pour chaque sommet u du graphe
9         if not(u in decouverts): # Si u n'est encore découvert
10            visiter_profondeur(u, g, decouverts) # Visite le sommet u
11     return d
12
13 def visiter_profondeur(u:int, g:'dict[int, list]', decouverts:list) -> None:
14     """Visite un sommet u en particulier"""
15
16     decouverts.append(u) # Signale que u est découvert
17     for v in g[u]: # Pour chaque sommet v de la liste d'adj de u
18         if v not in d: # Si v est non découvert
19             visiter_profondeur(v, g, decouverts) # on lance la visite de v
20     # u est totalement exploré mais on ne l'implémente pas en le mémorisant
```

## PARCOURS EN PROFONDEUR VERSION 2 : avec les couleurs

BLANC = 0

GRIS = 1

NOIR = 2

```
def parcours_profondeur(g:'dict[int, list]') -> dict:
    """Renvoie les informations (les parents) du parcours en profondeur"""

    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

    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: Idem largeur

# 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 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
```