# Graphes : Parcours

Quentin Fortier

April 19, 2022

# Parcours de graphe

On a souvent besoin de parcourir les sommets d'un graphe un par un. Voici les deux principaux parcours, visitant les sommets de proche en proche :

# Parcours de graphe

On a souvent besoin de parcourir les sommets d'un graphe un par un. Voici les deux principaux parcours, visitant les sommets de proche en proche :

1. **Parcours en profondeur** (ou DFS pour Depth-First Search) : on visite les sommets le plus profondément possible avant de revenir en arrière.

# Parcours de graphe

On a souvent besoin de parcourir les sommets d'un graphe un par un. Voici les deux principaux parcours, visitant les sommets de proche en proche :

1. **Parcours en profondeur** (ou DFS pour Depth-First Search) : on visite les sommets le plus profondément possible avant de revenir en arrière.
2. **Parcours en largeur** (ou BFS pour Breadth-First Search) : on visite les sommets par distance croissante depuis un sommet de départ.

# Parcours de graphe

On a souvent besoin de parcourir les sommets d'un graphe un par un. Voici les deux principaux parcours, visitant les sommets de proche en proche :

1. **Parcours en profondeur** (ou DFS pour Depth-First Search) : on visite les sommets le plus profondément possible avant de revenir en arrière.
2. **Parcours en largeur** (ou BFS pour Breadth-First Search) : on visite les sommets par distance croissante depuis un sommet de départ.

Si le graphe est connexe, tous les sommets sont visités.
Sinon, on applique un parcours sur chacune des composantes connexes.

# Parcours en profondeur (DFS)

Un **parcours en profondeur** sur un graphe $G = (V, E)$ depuis un sommet $u$ consiste à visiter $u$ puis visiter récursivement chaque voisin de $u$ qui n'a pas déjà été vu.

# Parcours en profondeur (DFS)

Un **parcours en profondeur** sur un graphe $G = (V, E)$ depuis un sommet $u$ consiste à visiter $u$ puis visiter récursivement chaque voisin de $u$ qui n'a pas déjà été vu.

Il est nécessaire de se souvenir des sommets déjà visités pour éviter de faire une infinité d'appels récursifs (en revisitant toujours les mêmes sommets).
D'où l'utilisation d'une liste `visited` et d'une fonction auxiliaire récursive (de façon à avoir accès à `visited` dans tous les appels récursifs).

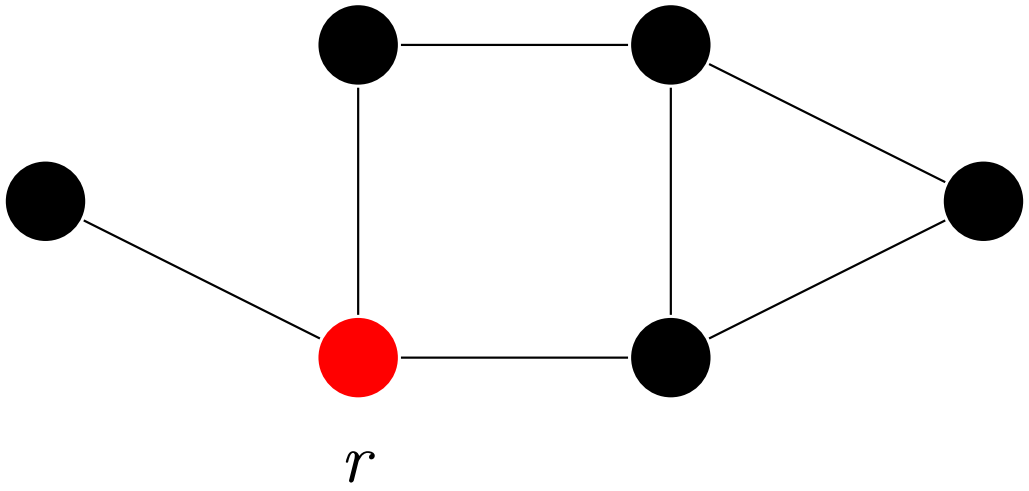
# Parcours en profondeur (DFS)

Parcours en profondeur sur un graphe $G$ représenté par liste d'adjacence, depuis un sommet $s$ :

```
def dfs(G, s):
    visited = [False]*len(G)
    def aux(u):
        if not visited[u]:
            visited[u] = True
            for v in G[u]:
                aux(v)
    aux(s)
```

`visited[u]` vaut `True` si `u` a déjà été visité

# Parcours en profondeur (DFS) : Exemple



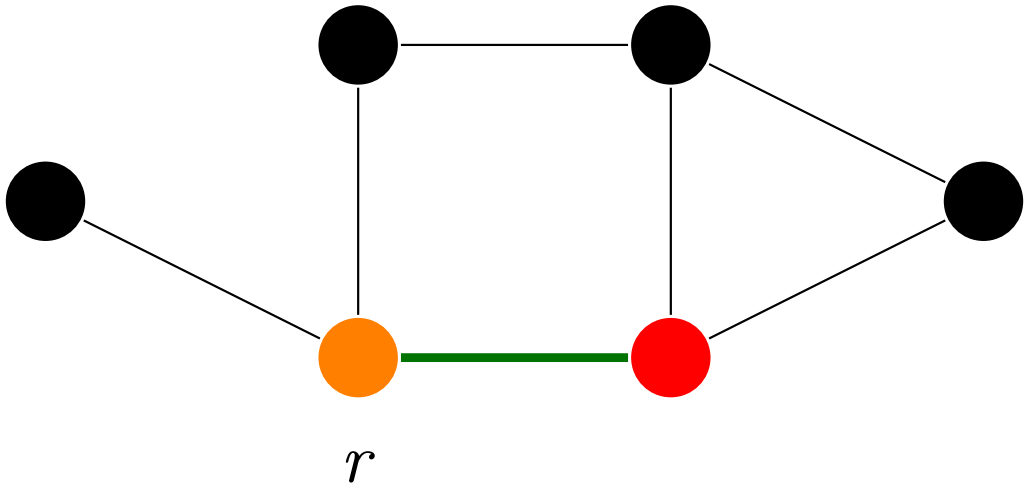

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



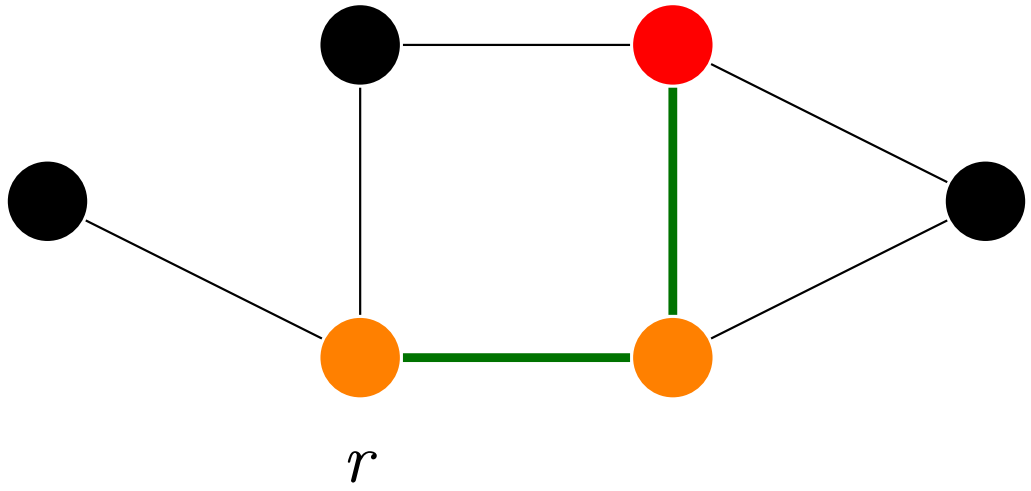

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



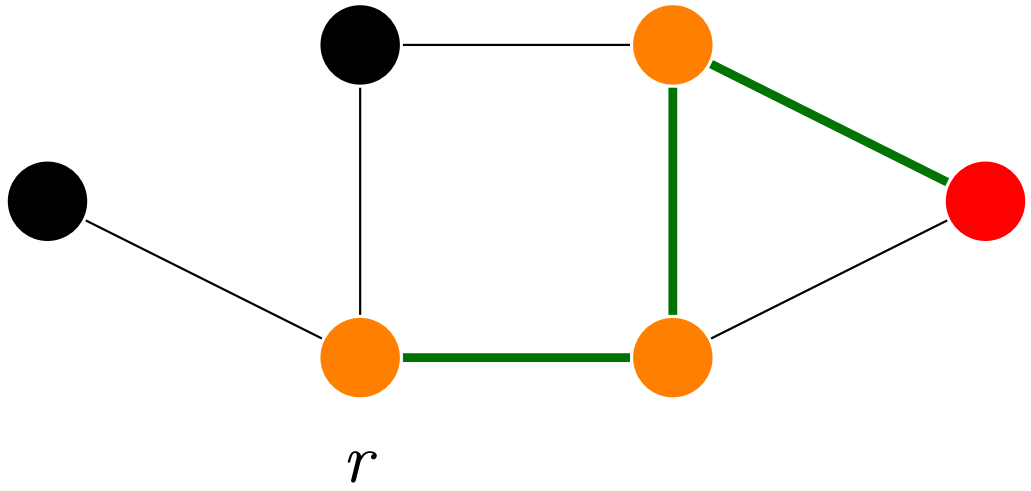

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



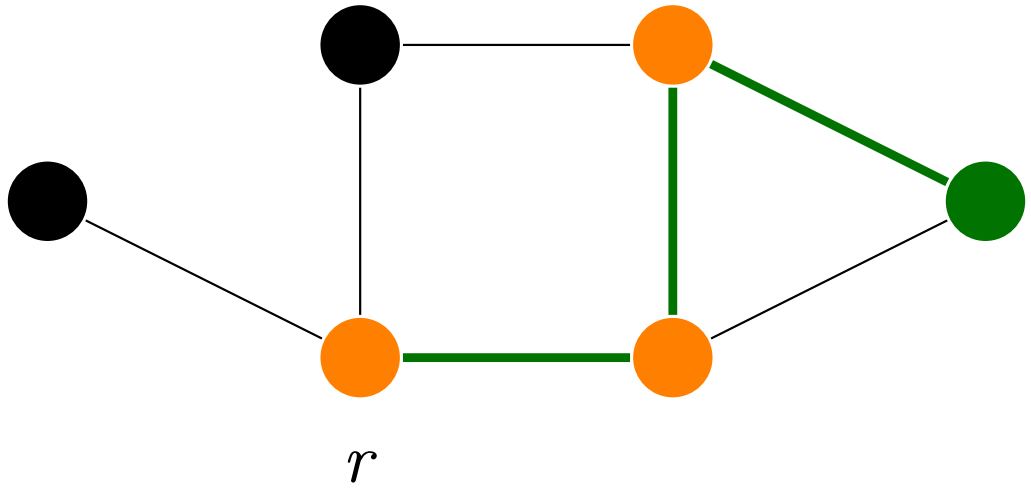

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



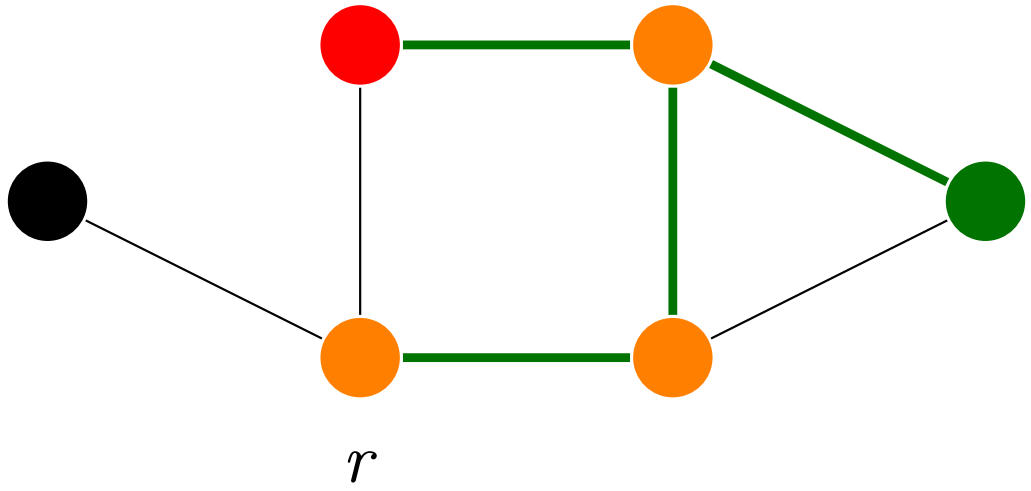

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



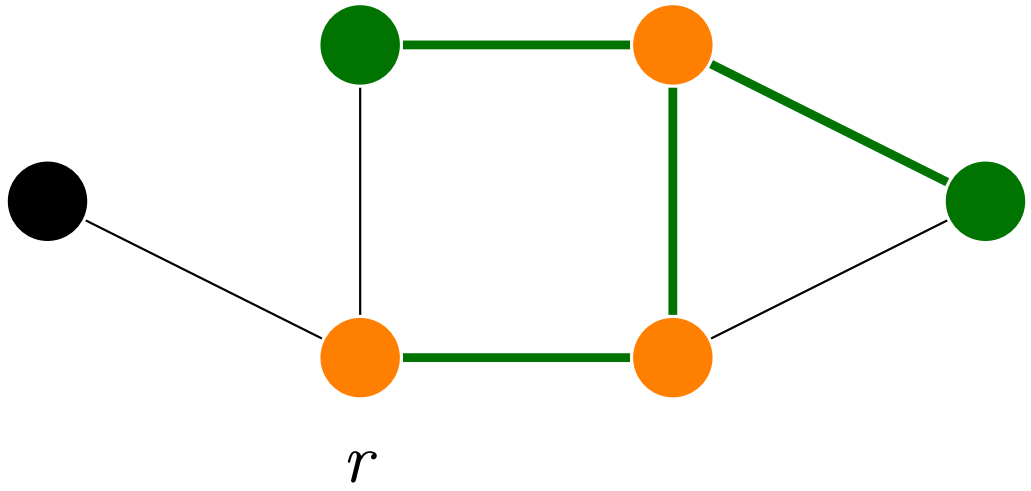

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



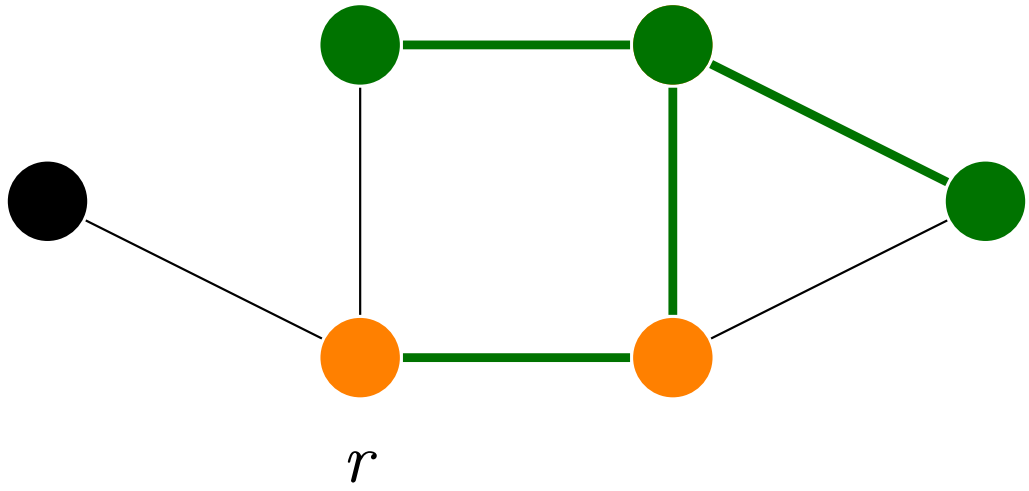

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



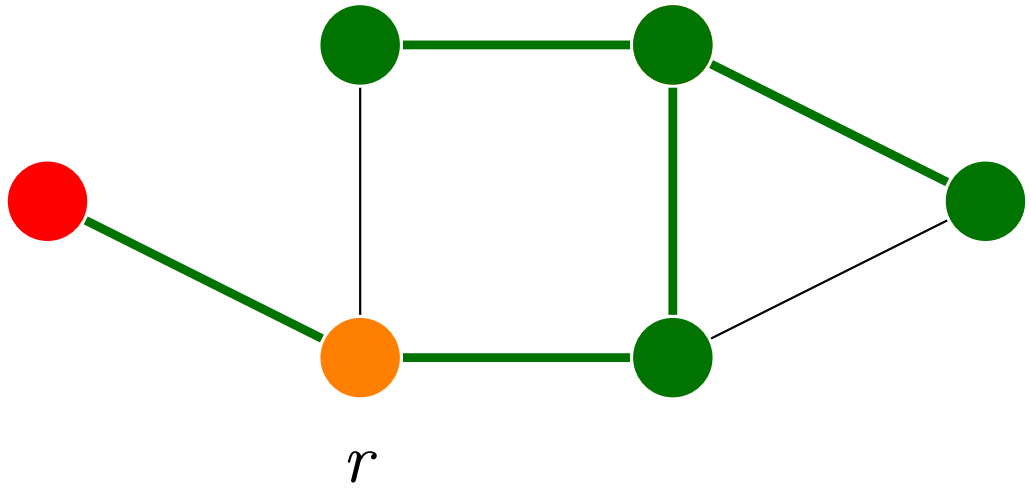

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



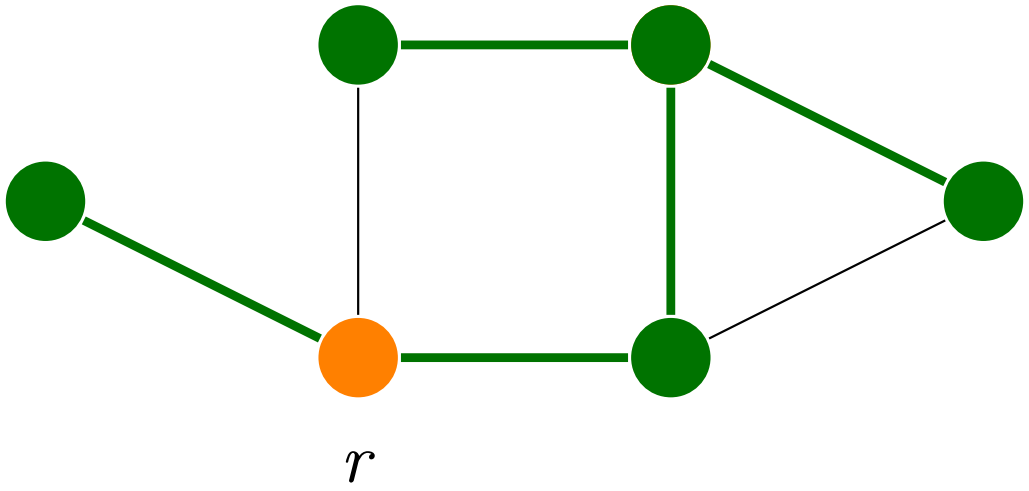

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple



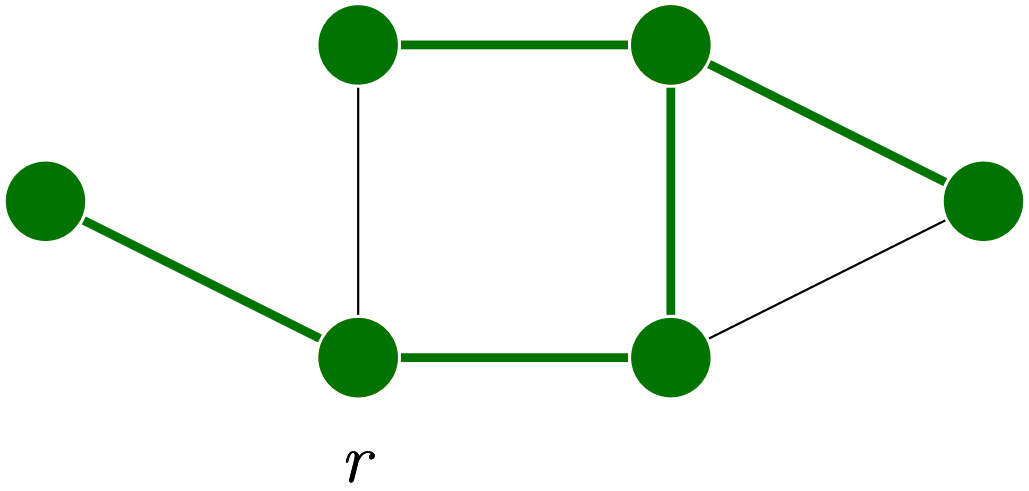

sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple




sommet pas encore visité

sommet en cours de traitement

appel récursif en pause

visite du sommet terminé

# Parcours en profondeur (DFS) : Exemple

**Exercice**

Adapter le code de la fonction ci-dessous pour afficher (avec `print`) les sommets dans l'ordre de leur visite dans le parcours en profondeur.

```
def dfs(G, s):
    visited = [False]*len(G)
    def aux(u):
        if not visited[u]:
            visited[u] = True
            for v in G[u]:
                aux(v)
    aux(s)
```

# Parcours en profondeur (DFS) : Exemple

**Exercice**

Écrire un parcours en profondeur pour un graphe représenté par matrice d'adjacence.

# Parcours en profondeur (DFS) : Application à la connexité

**Question**

Comment déterminer si un graphe **non orienté** est connexe?

# Parcours en profondeur (DFS) : Application à la connexité

**Question**

Comment déterminer si un graphe **non orienté** est connexe?

Il suffit de vérifier que le tableau `visited` ne contient que des `True`.

# Parcours en profondeur (DFS) : Application à la connexité

Si le graphe n'est pas connexe, on peut effectuer un parcours sur chacune des composantes connexes :

```
def dfs(G, s):
    visited = [False]*len(G)
    def aux(u):
        if not visited[u]:
            visited[u] = True
            for v in G[u]:
                aux(v)
    for u in range(n):
        aux(u)
```

# Parcours en profondeur (DFS) : Détection de cycle

**Question**

Comment déterminer si un graphe **non orienté** contient un cycle?

# Parcours en profondeur (DFS) : Détection de cycle

**Question**

Comment déterminer si un graphe **non orienté** contient un cycle?

On regarde si on revient sur un sommet déjà visité...

# Parcours en profondeur (DFS) : Détection de cycle

**Question**

Comment déterminer si un graphe **non orienté** contient un cycle?

On regarde si on revient sur un sommet déjà visité...et que ce n'est pas un fils qui revient sur son père !

# Parcours en profondeur (DFS) : Détection de cycle

**Question**

Comment déterminer si un graphe **non orienté** contient un cycle?

On regarde si on revient sur un sommet déjà visité...et que ce n'est pas un fils qui revient sur son père !

# Parcours en profondeur (DFS) : Détection de cycle

**Question**

Comment déterminer si un graphe **non orienté** contient un cycle?

On regarde si on revient sur un sommet déjà visité...et que ce n'est pas un fils qui revient sur son père !

déjà visité

# Parcours en profondeur (DFS) : Détection de cycle

**Question**

Comment déterminer si un graphe **non orienté** contient un cycle?

Il faut donc éviter de s'appeler récursivement sur son père :

```
def has_cycle(G, s):
    # renvoie True si G a un cycle atteignable depuis s
    visited = [False]*len(G)
    def aux(u, p):
        if not visited[u]:
            visited[u] = True
            for v in G[u]:
                if v != p and aux(v, u):
                    return True
        return False
    return aux(s, -1)
```

# Parcours en profondeur (DFS) : Avec pile

Au lieu d'utiliser la récursivité, on peut aussi stocker les prochains sommets à visiter dans une liste (utilisée comme une **pile**) :

```
def dfs(G, s): # G représenté par liste d'adjacence
    L = [s]
    visited = [False]*len(G)
    while len(L) > 0:
        u = L.pop()
        if not visited[u]:
            visited[u] = True
            for v in G[u]:
                L.append(v)
```