Parcours de graphe

Sébastien Godillon

SJTU SPEIT - Y3S1 2023/2024

Plan

1. Algorithme de parcours en largeur
   - Présentation, exemple et marquage
   - Exploration à l'aide d'une file
   - Description de l'algorithme
   - Implémentation et applications

2. Algorithme de parcours en profondeur
   - Présentation, exemple et marquage
   - Exploration à l'aide d'une pile
   - Description de l'algorithme
   - Implémentation et applications

Parcours de graphe

Un parcours de graphe est un algorithme consistant à explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.

Parcours de graphe

Un parcours de graphe est un algorithme consistant à explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.

Il existe de nombreux algorithmes de parcours qui permettent de résoudre plusieurs problèmes classiques de théorie des graphes, par exemples :

- étudier la connexité et déterminer les composantes connexes,
- calculer le plus court chemin entre deux sommets,
- chercher des cycles,
- trouver un arbre couvrant,
- etc.

Parcours de graphe

Un parcours de graphe est un algorithme consistant à explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.

Il existe de nombreux algorithmes de parcours qui permettent de résoudre plusieurs problèmes classiques de théorie des graphes, par exemples :

- étudier la connexité et déterminer les composantes connexes,
- calculer le plus court chemin entre deux sommets,
- chercher des cycles,
- trouver un arbre couvrant,
- etc.

Les deux algorithmes de parcours les plus connus sont :

1. l'algorithme de parcours en largeur,
2. l'algorithme de parcours en profondeur.

Plan

Algorithme de parcours en largeur

L'algorithme de parcours en largeur (ou BFS pour *Breadth-First Search* en anglais) consiste à :

Algorithme de parcours en largeur

L'algorithme de parcours en largeur (ou BFS pour *Breadth-First Search* en anglais) consiste à :

- partir d'un sommet,

Algorithme de parcours en largeur

L'algorithme de parcours en largeur (ou BFS pour *Breadth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer tous ses voisins,

Algorithme de parcours en largeur

L'algorithme de parcours en largeur (ou BFS pour *Breadth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer tous ses voisins,
- explorer tous les voisins de ces voisins (sauf le sommet de départ),

Algorithme de parcours en largeur

L'algorithme de parcours en largeur (ou BFS pour *Breadth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer tous ses voisins,
- explorer tous les voisins de ces voisins (sauf le sommet de départ),
- explorer tous les voisins non-explorés des sommets déjà explorés,

Algorithme de parcours en largeur

L'algorithme de parcours en largeur (ou BFS pour *Breadth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer tous ses voisins,
- explorer tous les voisins de ces voisins (sauf le sommet de départ),
- explorer tous les voisins non-explorés des sommets déjà explorés,
- puis etc.

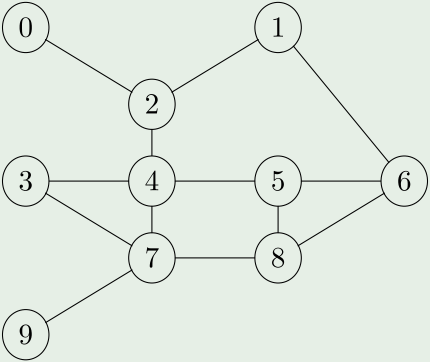
Algorithme de parcours en largeur

L'algorithme de parcours en largeur (ou BFS pour *Breadth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer tous ses voisins,
- explorer tous les voisins de ces voisins (sauf le sommet de départ),
- explorer tous les voisins non-explorés des sommets déjà explorés,
- puis etc.
- jusqu'à ce qu'il n'existe plus de voisins non-explorés.

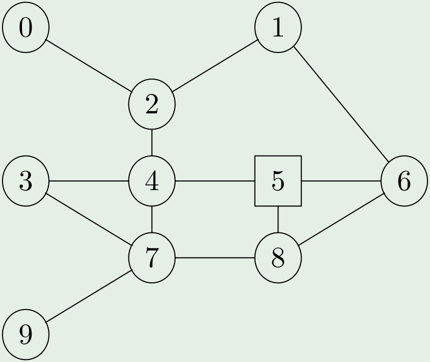
Algorithme de parcours en largeur

Exemple



Algorithme de parcours en largeur

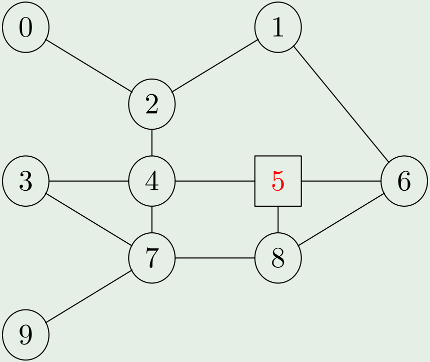
Exemple



$\boxed{\text{Choix}}$ d'un sommet de départ

Algorithme de parcours en largeur

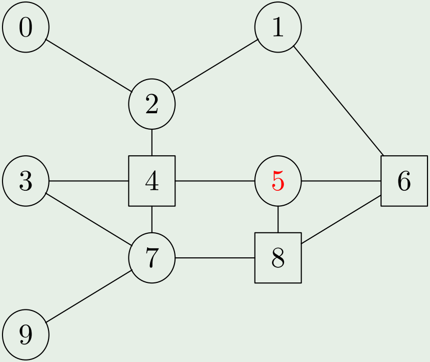
Exemple



Marquage du sommet de départ

Algorithme de parcours en largeur

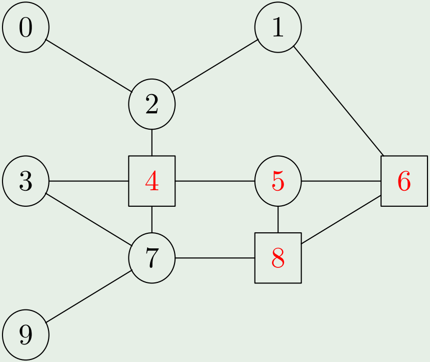
Exemple



Exploration des voisins du sommet de départ

Algorithme de parcours en largeur

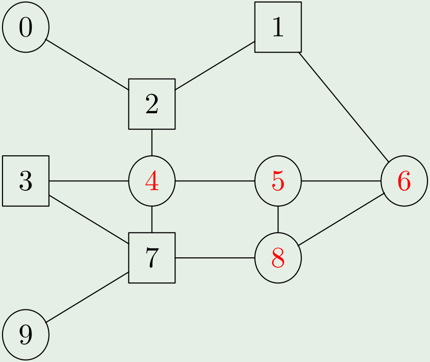
Exemple



Marquage des voisins du sommet de départ

Algorithme de parcours en largeur

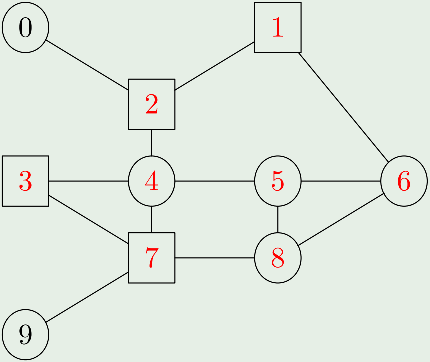
Exemple



Exploration des voisins non-marqués des sommets marqués

Algorithme de parcours en largeur

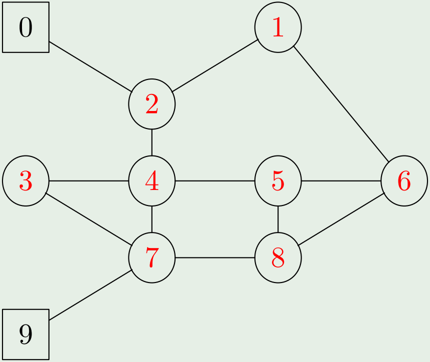
Exemple



Marquage des sommets explorés

Algorithme de parcours en largeur

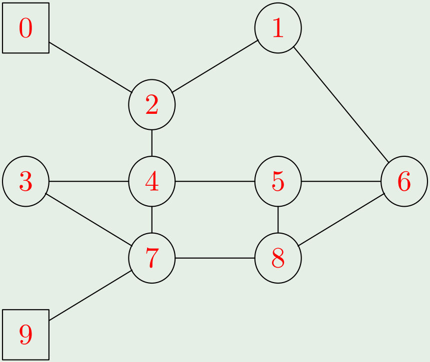
Exemple



$\boxed{\text{Exploration}}$ des voisins non-marqués des sommets marqués

Algorithme de parcours en largeur

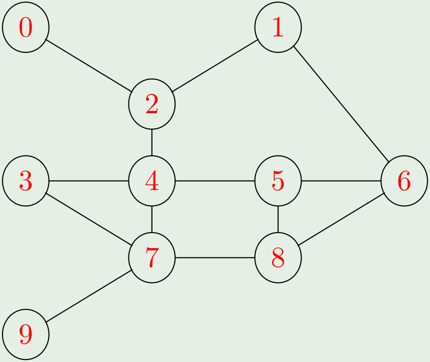
Exemple



Marquage des sommets explorés

Algorithme de parcours en largeur

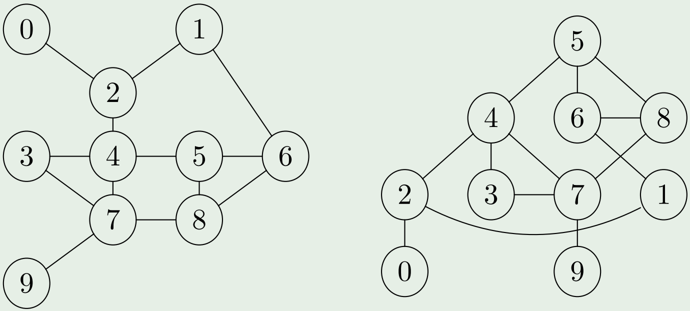
Exemple



Fin

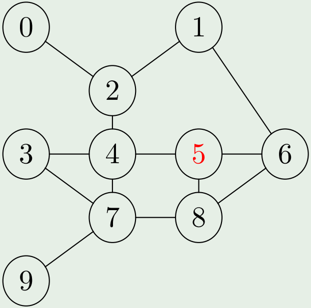
Algorithme de parcours en largeur

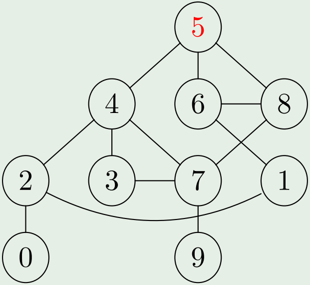
Exemple : Représentation équivalente



Algorithme de parcours en largeur

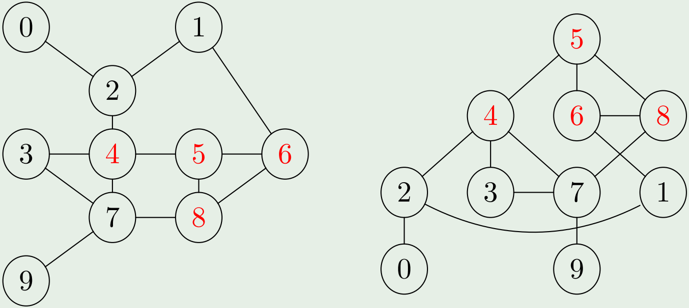
Exemple : Représentation équivalente





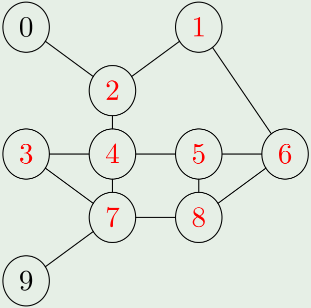
Algorithme de parcours en largeur

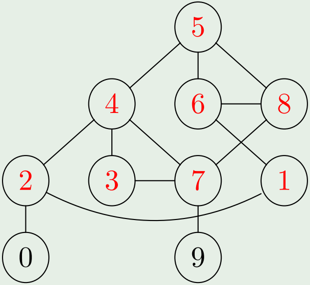
Exemple : Représentation équivalente



Algorithme de parcours en largeur

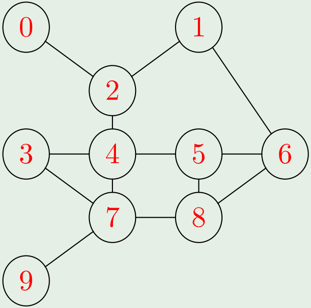
Exemple : Représentation équivalente

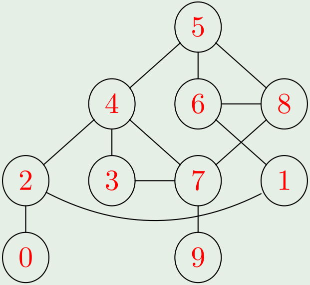




Algorithme de parcours en largeur

Exemple : Représentation équivalente





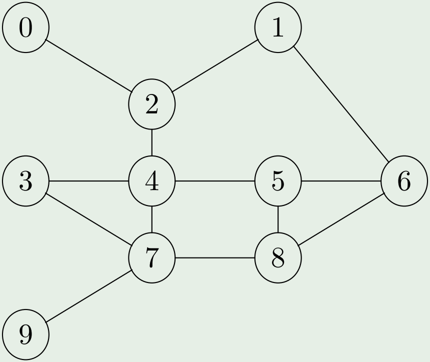
Algorithme de parcours en largeur

Pour distinguer les sommets marqués des sommets non-marqués, on peut utiliser une liste de booléens :

$$\texttt{Marques[s]} = \begin{cases} \texttt{True} \text{ si le sommet } \texttt{s} \text{ est marqué,} \\ \texttt{False} \text{ si le sommet } \texttt{s} \text{ n'est pas marqué.} \end{cases}$$

Algorithme de parcours en largeur

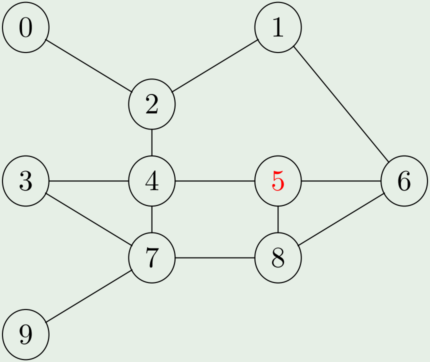
Exemple : Modélisation du marquage




```
Marques=[False,False,False,False,False,
         False,False,False,False,False]
```

Algorithme de parcours en largeur

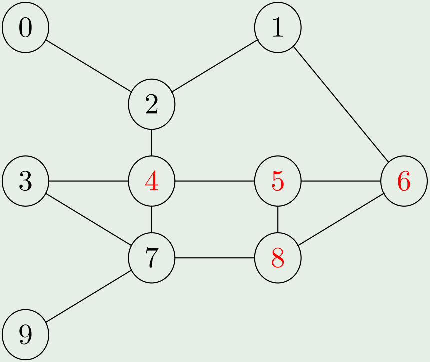
Exemple : Modélisation du marquage




```
Marques=[False,False,False,False,False,
         True,False,False,False,False]
```

Algorithme de parcours en largeur

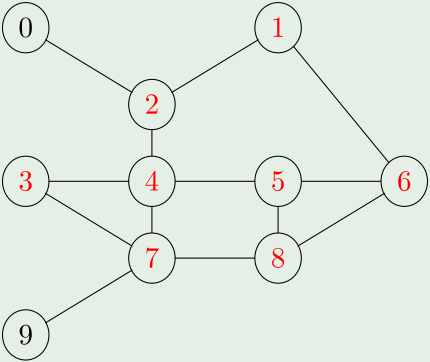
Exemple : Modélisation du marquage




```
Marques=[False,False,False,False,True,
              True,True,False,True,False]
```

Algorithme de parcours en largeur

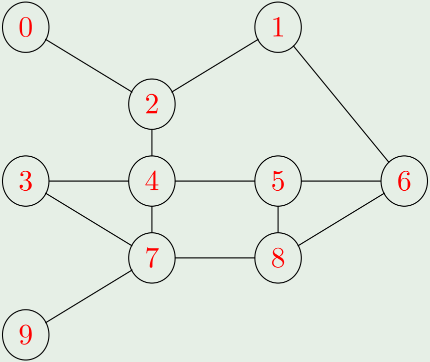
Exemple : Modélisation du marquage




```
Marques=[False,True,True,True,True,
             True,True,True,True,False]
```

Algorithme de parcours en largeur

Exemple : Modélisation du marquage



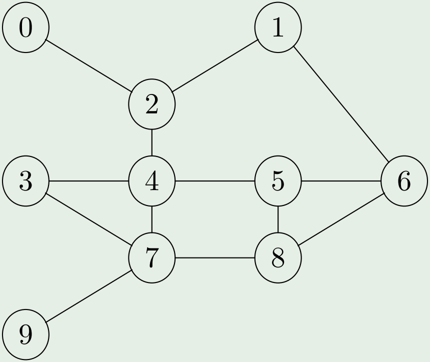

```
Marques=[True,True,True,True,True,
         True,True,True,True,True]
```

Algorithme de parcours en largeur

Pour modéliser l'exploration des sommets, on peut aussi utiliser une liste.

Algorithme de parcours en largeur

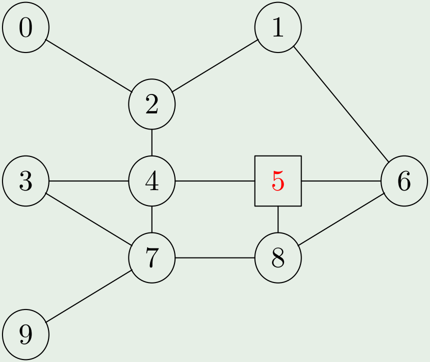
Exemple : Description de l'exploration



`Explores=[]`

Algorithme de parcours en largeur

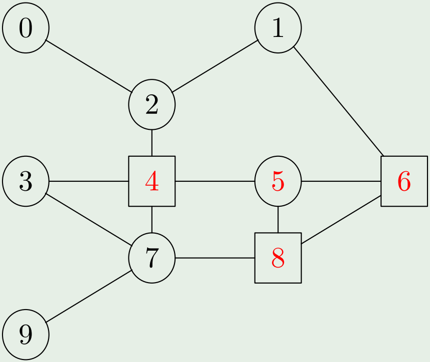
Exemple : Description de l'exploration



`Explores=[5]`

Algorithme de parcours en largeur

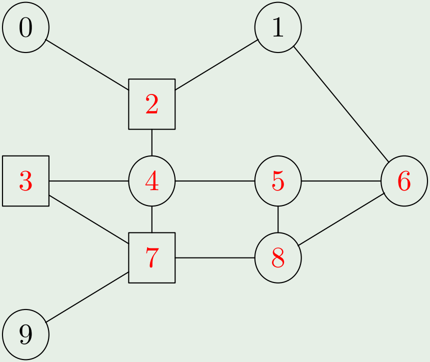
Exemple : Description de l'exploration



`Explores=[5,4,6,8]`

Algorithme de parcours en largeur

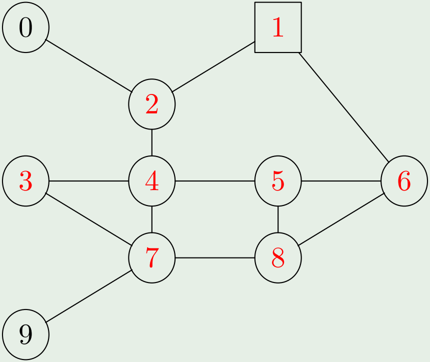
Exemple : Description de l'exploration



`Explores=[5,4,6,8,2,3,7]`

Algorithme de parcours en largeur

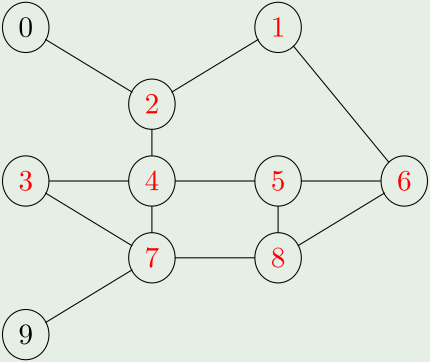
Exemple : Description de l'exploration



Explores=[5,4,6,8,2,3,7,1]

Algorithme de parcours en largeur

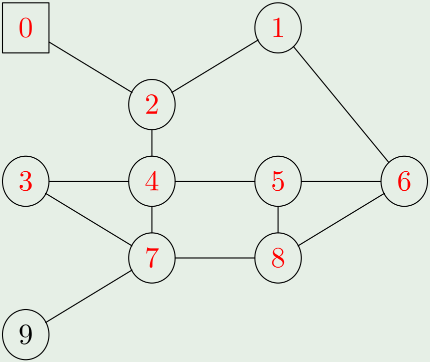
Exemple : Description de l'exploration



`Explores=[5,4,6,8,2,3,7,1]`

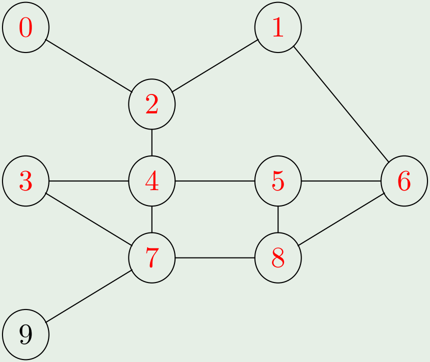
Algorithme de parcours en largeur

Exemple : Description de l'exploration



Explores=[5,4,6,8,2,3,7,1,0]

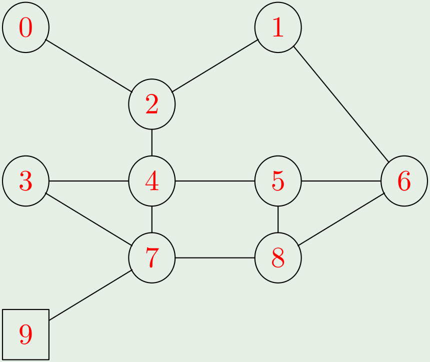
Exemple : Description de l'exploration



`Explores=[5,4,6,8,2,3,7,1,0]`

Algorithme de parcours en largeur

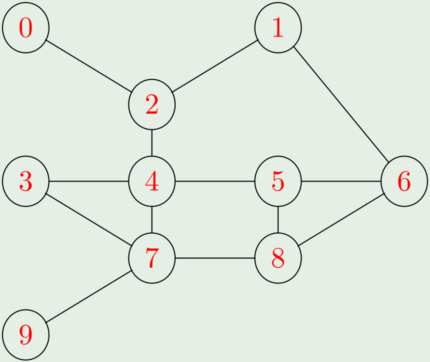
Exemple : Description de l'exploration



`Explores=[5,4,6,8,2,3,7,1,0,9]`

Algorithme de parcours en largeur

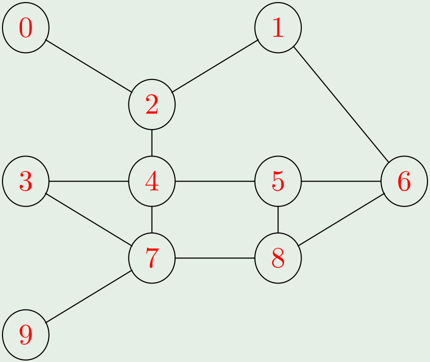
Exemple : Description de l'exploration



`Explores=[5,4,6,8,2,3,7,1,0,9]`

Algorithme de parcours en largeur

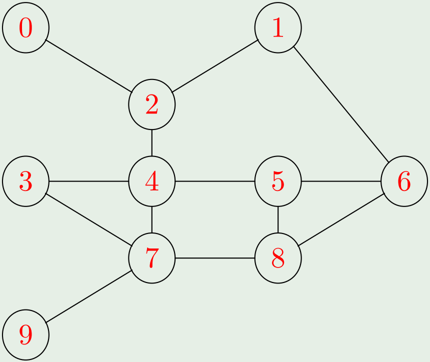
Exemple : Description de l'exploration



`Explores=[5,4,6,8,2,3,7,1,0,9]`

Algorithme de parcours en largeur

Exemple : Description de l'exploration



`Explores=[5,4,6,8,2,3,7,1,0,9]`

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

```
Explores=[]
```

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

```
Explores=[]
          [5]
```

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

```
Explores=[]
         [5]
         [5,4,6,8]
```

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

```
Explores=[]
         [5]
         [5,4,6,8]
         [5,4,6,8,2,3,7]
```

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	`[]`
`[5]`	`[5]`
`[5,4,6,8]`	`[4,6,8]`
`[5,4,6,8,2,3,7]`	`[6,8,2,3,7]`

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	`[]`
`[5]`	`[5]`
`[5,4,6,8]`	`[4,6,8]`
`[5,4,6,8,2,3,7]`	`[6,8,2,3,7]`
`[5,4,6,8,2,3,7,1]`	`[8,2,3,7,1]`

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

```
Explores=[]
         [5]
         [5,4,6,8]
         [5,4,6,8,2,3,7]
         [5,4,6,8,2,3,7,1]
         [5,4,6,8,2,3,7,1]
```

Ou plus simplement :

```
[]
[5]
[4,6,8]
[6,8,2,3,7]
[8,2,3,7,1]
[2,3,7,1]
```

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	`[]`
`[5]`	`[5]`
`[5,4,6,8]`	`[4,6,8]`
`[5,4,6,8,2,3,7]`	`[6,8,2,3,7]`
`[5,4,6,8,2,3,7,1]`	`[8,2,3,7,1]`
`[5,4,6,8,2,3,7,1]`	`[2,3,7,1]`
`[5,4,6,8,2,3,7,1,0]`	`[3,7,1,0]`

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	`[]`
`[5]`	`[5]`
`[5,4,6,8]`	`[4,6,8]`
`[5,4,6,8,2,3,7]`	`[6,8,2,3,7]`
`[5,4,6,8,2,3,7,1]`	`[8,2,3,7,1]`
`[5,4,6,8,2,3,7,1]`	`[2,3,7,1]`
`[5,4,6,8,2,3,7,1,0]`	`[3,7,1,0]`
`[5,4,6,8,2,3,7,1,0]`	`[7,1,0,9]`

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	[]
[5]	[5]
[5,4,6,8]	[4,6,8]
[5,4,6,8,2,3,7]	[6,8,2,3,7]
[5,4,6,8,2,3,7,1]	[8,2,3,7,1]
[5,4,6,8,2,3,7,1]	[2,3,7,1]
[5,4,6,8,2,3,7,1,0]	[3,7,1,0]
[5,4,6,8,2,3,7,1,0]	[7,1,0,9]
[5,4,6,8,2,3,7,1,0,9]	[1,0,9]

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	[]
[5]	[5]
[5,4,6,8]	[4,6,8]
[5,4,6,8,2,3,7]	[6,8,2,3,7]
[5,4,6,8,2,3,7,1]	[8,2,3,7,1]
[5,4,6,8,2,3,7,1]	[2,3,7,1]
[5,4,6,8,2,3,7,1,0]	[3,7,1,0]
[5,4,6,8,2,3,7,1,0]	[7,1,0,9]
[5,4,6,8,2,3,7,1,0,9]	[1,0,9]
[5,4,6,8,2,3,7,1,0,9]	[0,9]

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	`[]`
`[5]`	`[5]`
`[5,4,6,8]`	`[4,6,8]`
`[5,4,6,8,2,3,7]`	`[6,8,2,3,7]`
`[5,4,6,8,2,3,7,1]`	`[8,2,3,7,1]`
`[5,4,6,8,2,3,7,1]`	`[2,3,7,1]`
`[5,4,6,8,2,3,7,1,0]`	`[3,7,1,0]`
`[5,4,6,8,2,3,7,1,0]`	`[7,1,0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[1,0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[9]`

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	`[]`
`[5]`	`[5]`
`[5,4,6,8]`	`[4,6,8]`
`[5,4,6,8,2,3,7]`	`[6,8,2,3,7]`
`[5,4,6,8,2,3,7,1]`	`[8,2,3,7,1]`
`[5,4,6,8,2,3,7,1]`	`[2,3,7,1]`
`[5,4,6,8,2,3,7,1,0]`	`[3,7,1,0]`
`[5,4,6,8,2,3,7,1,0]`	`[7,1,0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[1,0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[]`

Algorithme de parcours en largeur

Exemple : Évolution de la liste d'exploration

	Ou plus simplement :
`Explores=[]`	`[]`
`[5]`	`[5]`
`[5,4,6,8]`	`[4,6,8]`
`[5,4,6,8,2,3,7]`	`[6,8,2,3,7]`
`[5,4,6,8,2,3,7,1]`	`[8,2,3,7,1]`
`[5,4,6,8,2,3,7,1]`	`[2,3,7,1]`
`[5,4,6,8,2,3,7,1,0]`	`[3,7,1,0]`
`[5,4,6,8,2,3,7,1,0]`	`[7,1,0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[1,0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[0,9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[9]`
`[5,4,6,8,2,3,7,1,0,9]`	`[]`
	$\longleftarrow$

Algorithme de parcours en largeur

Définition

Une **file** est une structure de données (par exemple une liste) construite sur le principe «premier entré, premier sorti» (ou **FIFO** pour *First In First Out* en anglais).

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

1. **Créer** une file vide.

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

1. **Créer** une file vide.
2. **Mettre** s dans la file et **marquer** s.

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

1. **Créer** une file vide.
2. **Mettre** s dans la file et **marquer** s.
3. **Tant que** la file n'est pas vide :

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

1. **Créer** une file vide.
2. **Mettre** s dans la file et **marquer** s.
3. **Tant que** la file n'est pas vide :
    4. **Retirer** le premier sommet de la file.

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

1. **Créer** une file vide.
2. **Mettre** s dans la file et **marquer** s.
3. **Tant que** la file n'est pas vide :
    4. **Retirer** le premier sommet de la file.
    5. **Pour tout** v voisin non-marqué du sommet retiré :

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

1. **Créer** une file vide.
2. **Mettre** s dans la file et **marquer** s.
3. **Tant que** la file n'est pas vide :
    4. **Retirer** le premier sommet de la file.
    5. **Pour tout** v voisin non-marqué du sommet retiré :
        6. **Mettre** v à la fin de la file et **marquer** v.

Algorithme de parcours en largeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en largeur

1. **Créer** une file vide.
2. **Mettre** s dans la file et **marquer** s.
3. **Tant que** la file n'est pas vide :
   4. **Retirer** le premier sommet de la file.
   5. **Pour tout** v voisin non-marqué du sommet retiré :
      6. **Mettre** v à la fin de la file et **marquer** v.
   7. **Retourner** à 3.

Algorithme de parcours en largeur

Convergence

Puisque tout sommet marqué n'est plus exploré, chaque sommet est donc exploré au plus une fois. Par conséquent, l'algorithme de parcours en largeur s'arrête après un nombre fini d'instructions.

Algorithme de parcours en largeur

Exemple d'implémentation à l'aide d'une liste d'adjacence :

```python
def bfs_liste(L,s):
    n=len(L) # ordre du graphe
    Marques=[False]*n # initialisation du marquage
    File_explores=[s] # étapes 1 et 2
    Marques[s]=True # étape 2
    while File_explores!=[]: # étapes 3 et 7
        t=File_explores.pop(0) # étape 4
        for v in L[t]: # étape 5
            if not(Marques[v]): # étape 5
                File_explores.append(v) # etape 6
                Marques[v]=True # étape 6
    # à compléter
```

Algorithme de parcours en largeur

Exemple d'implémentation à l'aide d'une matrice d'adjacence :

```
def bfs_matrice(M,s):
    n=len(M)
    Marques=[False]*n
    File_explores=[s]
    Marques[s]=True
    while File_explores!=[]:
        t=File_explores.pop(0)
        for v in range(n): # !!
            if M[t][v]!=0 and not(Marques[v]): # !!
                File_explores.append(v)
                Marques[v]=True
    # à compléter
```

Algorithme de parcours en largeur

Application : Tester la connexité de G

G est connexe si et seulement si tous ses sommets sont marqués après un parcours en largeur partant de n'importe quel sommet.

Algorithme de parcours en largeur

Exemple d'implémentation d'un test de connexité à l'aide d'un parcours en largeur et d'une liste d'adjacence.

```
def est_connexe(L):
    n=len(L)
    Marques=[False]*n
    File_explores=[0] # s=0
    Marques[0]=True # s=0
    while File_explores!=[]:
        t=File_explores.pop(0)
        for v in L[t]:
            if not(Marques[v]):
                File_explores.append(v)
                Marques[v]=True
    for i in range(n): # début du test
        if not(Marques[i]):
            return False # G n'est pas connexe
    return True # G est connexe
```

Algorithme de parcours en largeur

Application : Calculer la composante connexe de s.

La composante connexe de s est l'ensemble des sommets marqués après un parcours en largeur partant de s.

Algorithme de parcours en largeur

Exemple d'implémentation d'un calcul de composante connexe à l'aide d'un parcours en largeur et d'une liste d'adjacence.

```python
def composante_connexe(L,s):
    n=len(L)
    Marques=[False]*n
    File_explores=[s]
    Marques[s]=True
    while File_explores!=[]:
        t=File_explores.pop(0)
        for v in L[t]:
            if not(Marques[v]):
                File_explores.append(v)
                Marques[v]=True
    Composante_connexe_s=[] # début du calcul
    for i in range(n):
        if Marques[i]:
            Composante_connexe_s.append(i)
    return Composante_connexe_s
```

Algorithme de parcours en largeur

Appl. : Calculer la dist. de s à chaque som. de sa comp. connexe.

Lors d'un parcours en largeur partant de s, les sommets sont explorés par distances croissantes à s :

Algorithme de parcours en largeur

Appl. : Calculer la dist. de s à chaque som. de sa comp. connexe.

Lors d'un parcours en largeur partant de s, les sommets sont explorés par distances croissantes à s :

- s qui est à distance 0 de s,

Algorithme de parcours en largeur

Appl. : Calculer la dist. de s à chaque som. de sa comp. connexe.

Lors d'un parcours en largeur partant de s, les sommets sont explorés par distances croissantes à s :

- s qui est à distance 0 de s,
- les voisins de s qui sont à distance 1 de s,

Algorithme de parcours en largeur

Appl. : Calculer la dist. de s à chaque som. de sa comp. connexe.

Lors d'un parcours en largeur partant de s, les sommets sont explorés par distances croissantes à s :

- s qui est à distance 0 de s,
- les voisins de s qui sont à distance 1 de s,
- les voisins des voisins de s (sauf s) qui sont à distance 2 de s,

Algorithme de parcours en largeur

Appl. : Calculer la dist. de s à chaque som. de sa comp. connexe.

Lors d'un parcours en largeur partant de s, les sommets sont explorés par distances croissantes à s :

- s qui est à distance 0 de s,
- les voisins de s qui sont à distance 1 de s,
- les voisins des voisins de s (sauf s) qui sont à distance 2 de s,
- les voisins non-explorés des som. déjà explorés qui sont à dist. 3 de s,

Algorithme de parcours en largeur

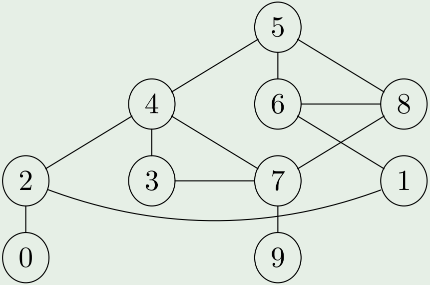
Appl. : Calculer la dist. de s à chaque som. de sa comp. connexe.

Lors d'un parcours en largeur partant de s, les sommets sont explorés par distances croissantes à s :

- s qui est à distance 0 de s,
- les voisins de s qui sont à distance 1 de s,
- les voisins des voisins de s (sauf s) qui sont à distance 2 de s,
- les voisins non-explorés des som. déjà explorés qui sont à dist. 3 de s,
- puis etc.

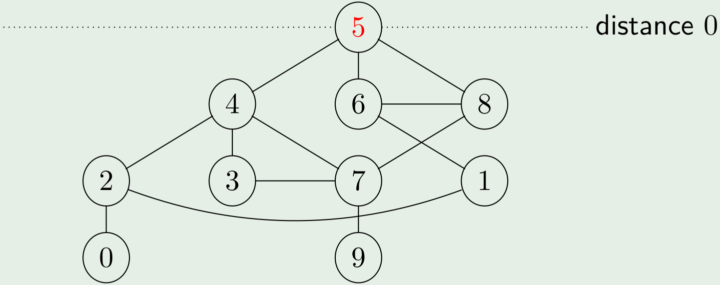
Algorithme de parcours en largeur

Exemple : Calcul des distances



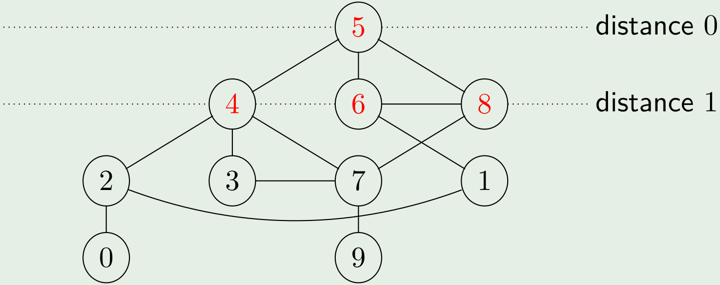
Algorithme de parcours en largeur

Exemple : Calcul des distances



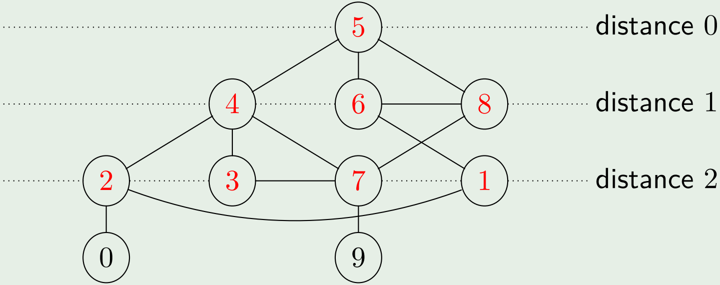
Algorithme de parcours en largeur

Exemple : Calcul des distances



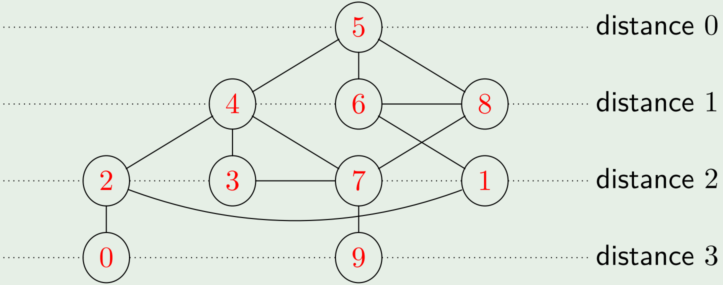
Algorithme de parcours en largeur

Exemple : Calcul des distances



Algorithme de parcours en largeur

Exemple : Calcul des distances



Algorithme de parcours en largeur

Exemple d'implémentation du calcul des distances à l'aide d'un parcours en largeur et d'une liste d'adjacence.

```python
def distances(L,s):
    n=len(L)
    Marques=[False]*n
    File_explores=[s]
    Marques[s]=True
    Dist=[None]*n # liste des distances
    Dist[s]=0 # distance de s à s
    while File_explores!=[]:
        t=File_explores.pop(0)
        for v in L[t]:
            if not(Marques[v]):
                File_explores.append(v)
                Marques[v]=True
                Dist[v]=Dist[t]+1 # dist. suivante
    return Dist
```

Plan

Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS pour *Depth-First Search* en anglais) consiste à :

Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS pour *Depth-First Search* en anglais) consiste à :

- partir d'un sommet,

Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS pour *Depth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer des sommets en suivant un chemin élémentaire le plus loin possible,

Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS pour *Depth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer des sommets en suivant un chemin élémentaire le plus loin possible,
- revenir en arrière jusqu'au dernier sommet du chemin ayant des voisins non-explorés,

Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS pour *Depth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer des sommets en suivant un chemin élémentaire le plus loin possible,
- revenir en arrière jusqu'au dernier sommet du chemin ayant des voisins non-explorés,
- explorer des sommets non-explorés en suivant un chemin élémentaire le plus loin possible,

Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS pour *Depth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer des sommets en suivant un chemin élémentaire le plus loin possible,
- revenir en arrière jusqu'au dernier sommet du chemin ayant des voisins non-explorés,
- explorer des sommets non-explorés en suivant un chemin élémentaire le plus loin possible,
- puis etc.

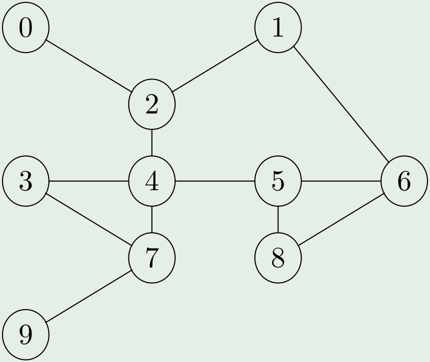
Algorithme de parcours en profondeur

L'algorithme de parcours en profondeur (ou DFS pour *Depth-First Search* en anglais) consiste à :

- partir d'un sommet,
- explorer des sommets en suivant un chemin élémentaire le plus loin possible,
- revenir en arrière jusqu'au dernier sommet du chemin ayant des voisins non-explorés,
- explorer des sommets non-explorés en suivant un chemin élémentaire le plus loin possible,
- puis etc.
- jusqu'à ce qu'il n'existe plus de voisins non-explorés.

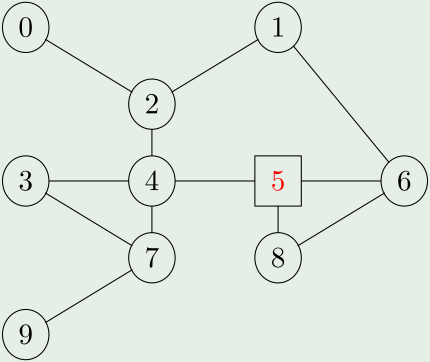
Algorithme de parcours en profondeur

Exemple



Algorithme de parcours en profondeur

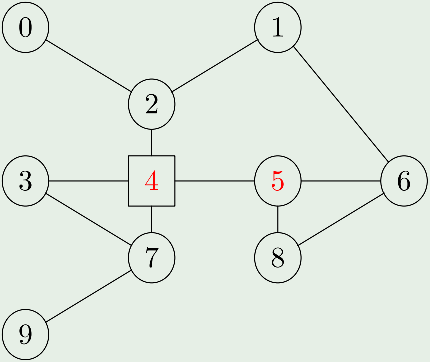
Exemple



$\boxed{\text{Choix}}$ et marquage d'un sommet de départ

Algorithme de parcours en profondeur

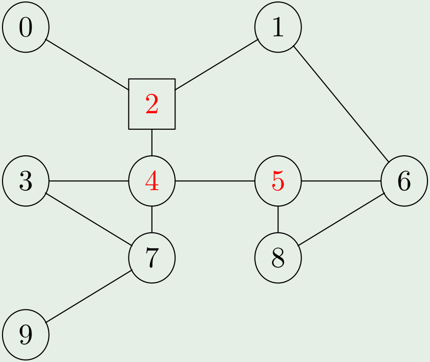
Exemple



$\boxed{\text{Exploration}}$ et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

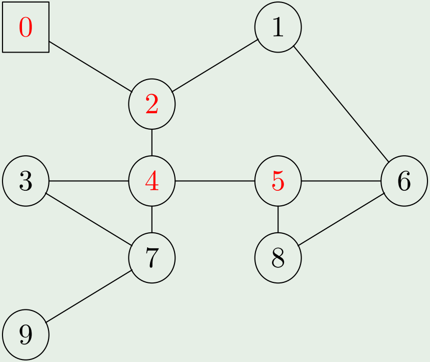
Exemple



Exploration et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

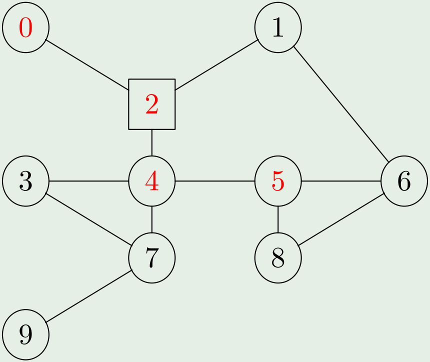
Exemple



$\boxed{\text{Exploration}}$ et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

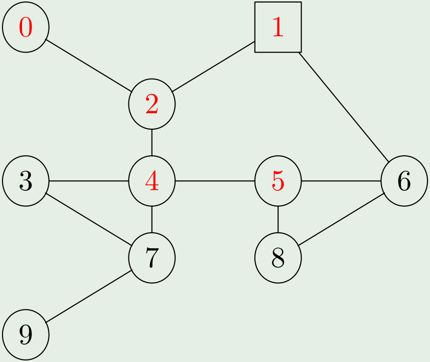
Exemple



Retour en arrière

Algorithme de parcours en profondeur

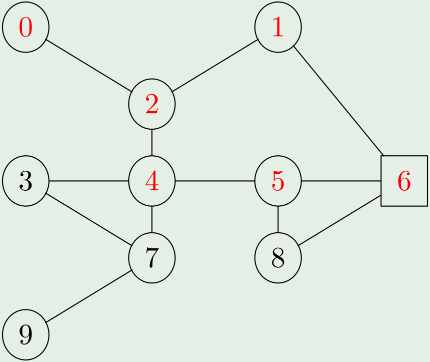
Exemple



Exploration et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

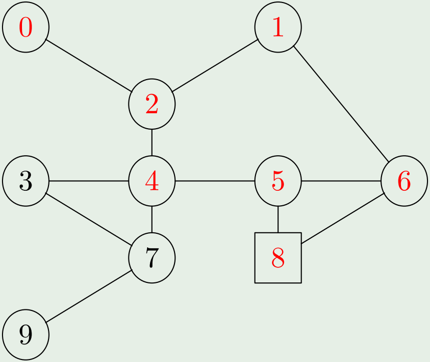
Exemple



Exploration et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

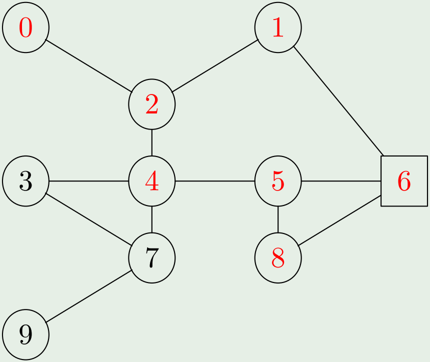
Exemple



Exploration et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

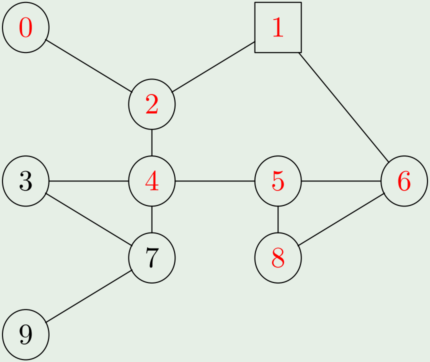
Exemple



Retour en arrière

Algorithme de parcours en profondeur

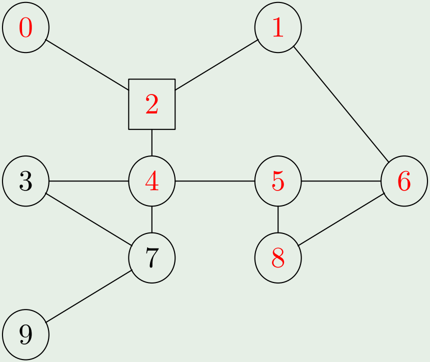
Exemple



Retour en arrière

Algorithme de parcours en profondeur

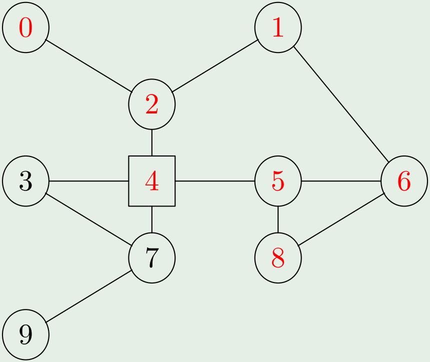
Exemple



Retour en arrière

Algorithme de parcours en profondeur

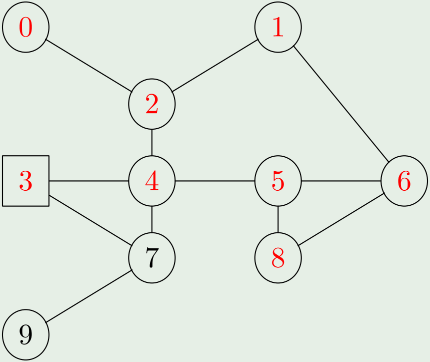
Exemple



Retour en arrière

Algorithme de parcours en profondeur

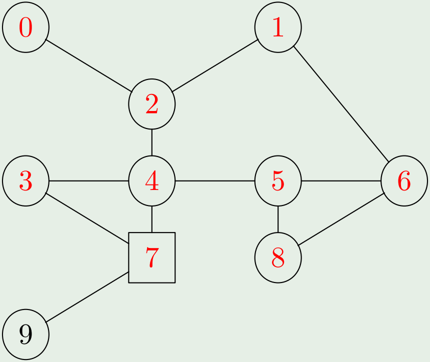
Exemple



Exploration et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

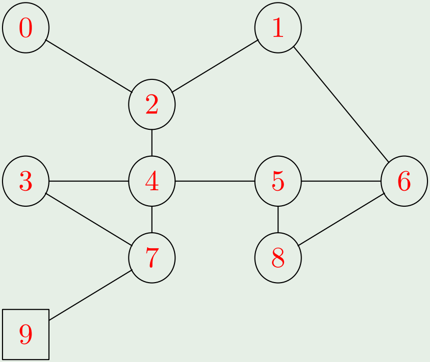
Exemple



$\boxed{\text{Exploration}}$ et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

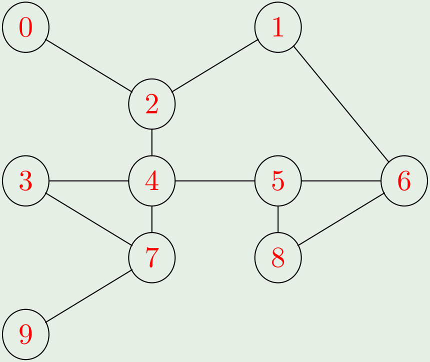
Exemple



Exploration et marquage d'un sommet suivant sur un chemin élémentaire

Algorithme de parcours en profondeur

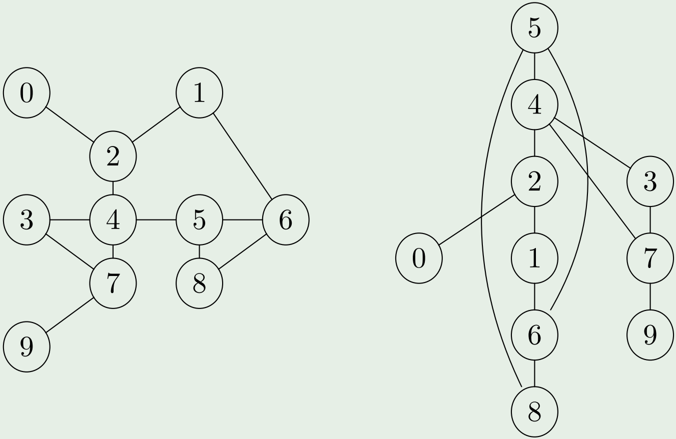
Exemple



Fin

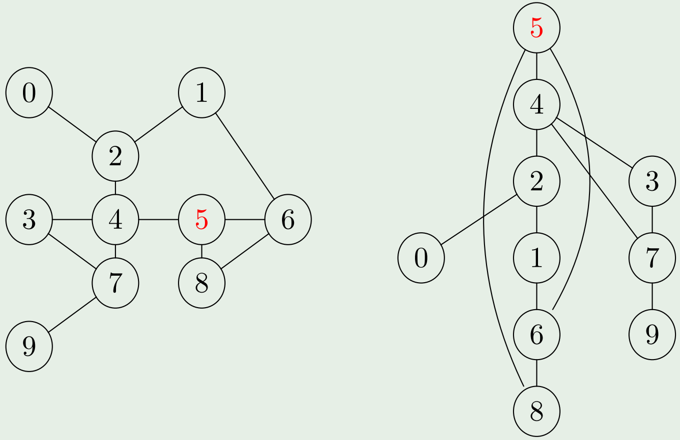
Algorithme de parcours en profondeur

Exemple : Représentation équivalente



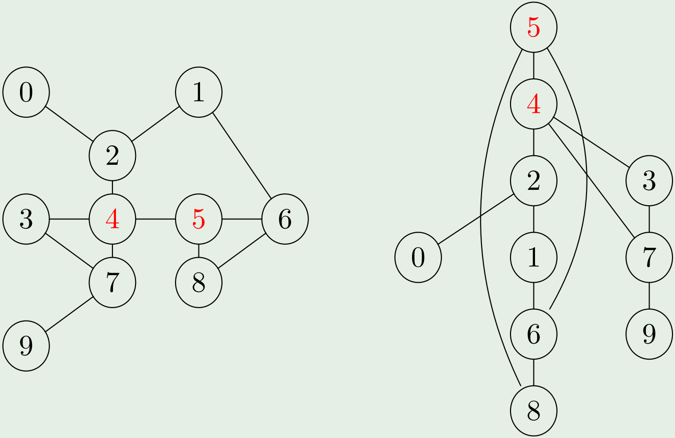
Algorithme de parcours en profondeur

Exemple : Représentation équivalente



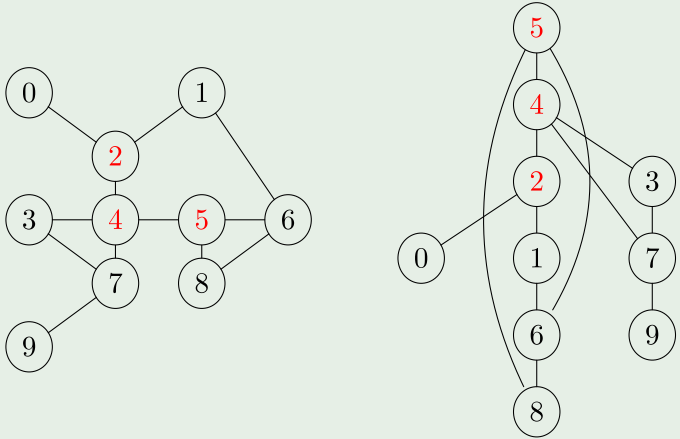
Algorithme de parcours en profondeur

Exemple : Représentation équivalente



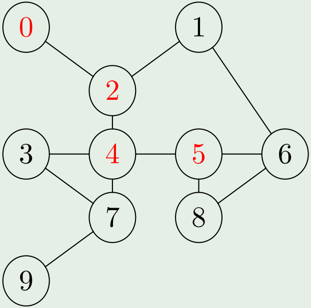
Algorithme de parcours en profondeur

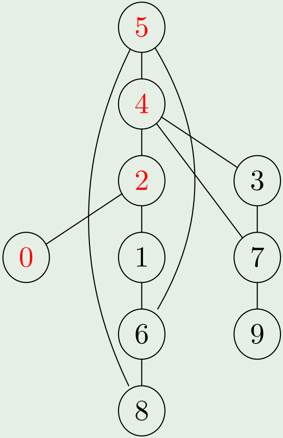
Exemple : Représentation équivalente



Algorithme de parcours en profondeur

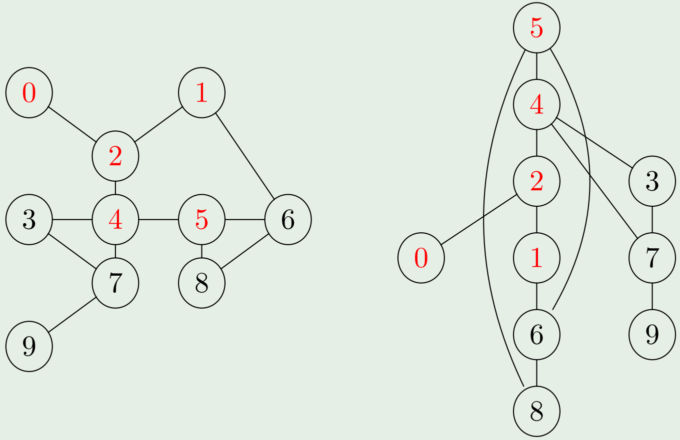
Exemple : Représentation équivalente





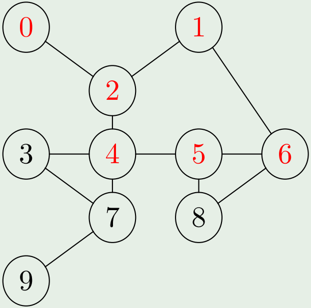
Algorithme de parcours en profondeur

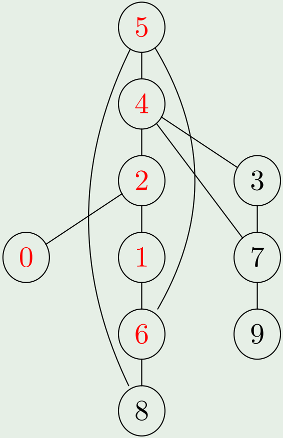
Exemple : Représentation équivalente



Algorithme de parcours en profondeur

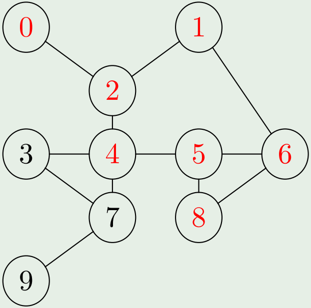
Exemple : Représentation équivalente

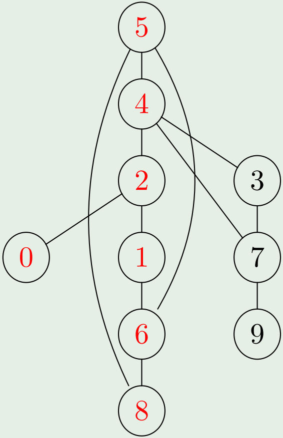




Algorithme de parcours en profondeur

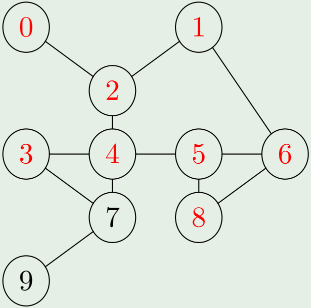
Exemple : Représentation équivalente

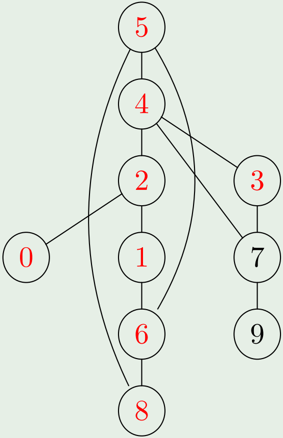




Algorithme de parcours en profondeur

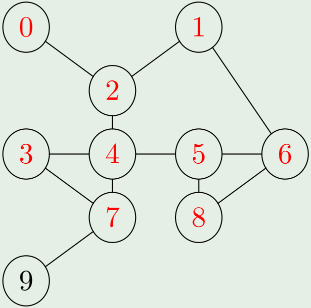
Exemple : Représentation équivalente

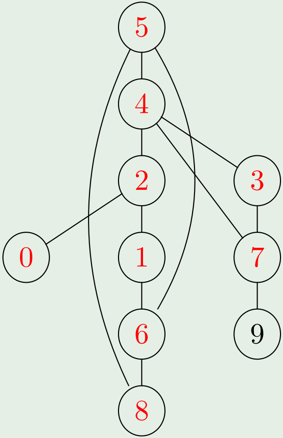




Algorithme de parcours en profondeur

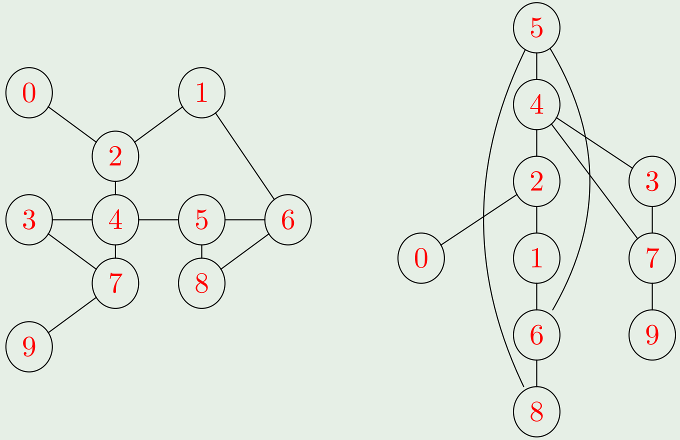
Exemple : Représentation équivalente





Algorithme de parcours en profondeur

Exemple : Représentation équivalente



Algorithme de parcours en profondeur

Pour distinguer les sommets marqués des sommets non-marqués, on peut utiliser une liste de booléens comme pour le parcours en largeur :

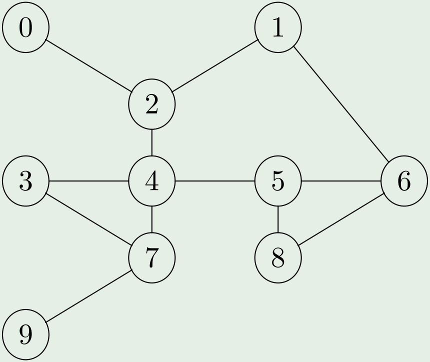
$$\texttt{Marques[s]} = \begin{cases} \texttt{True} \text{ si le sommet } \texttt{s} \text{ est marqué,} \\ \texttt{False} \text{ si le sommet } \texttt{s} \text{ n'est pas marqué.} \end{cases}$$

Algorithme de parcours en profondeur

Pour modéliser l'exploration des sommets, on peut aussi utiliser une liste.

Algorithme de parcours en profondeur

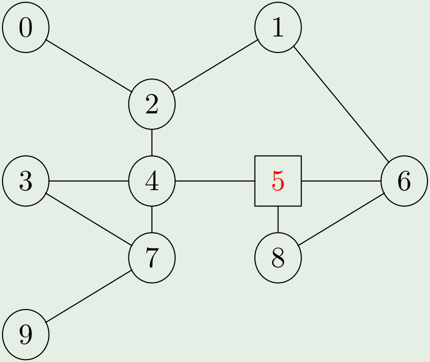
Exemple : Description de l'exploration



`Explores=[]`

Algorithme de parcours en profondeur

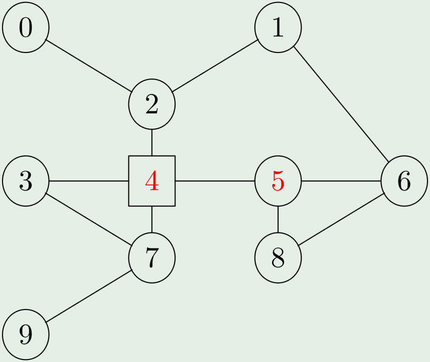
Exemple : Description de l'exploration



`Explores=[5]`

Algorithme de parcours en profondeur

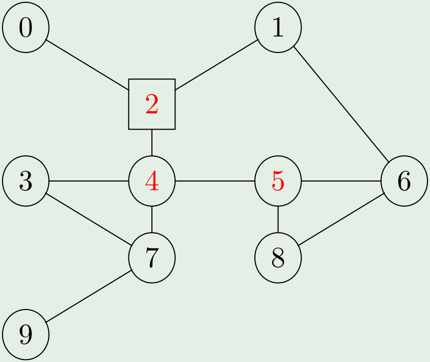
Exemple : Description de l'exploration



`Explores=[5,4]`

Algorithme de parcours en profondeur

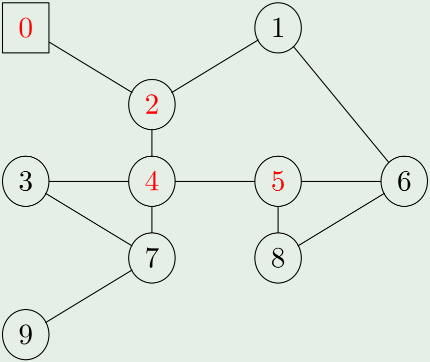
Exemple : Description de l'exploration



`Explores=[5,4,2]`

Algorithme de parcours en profondeur

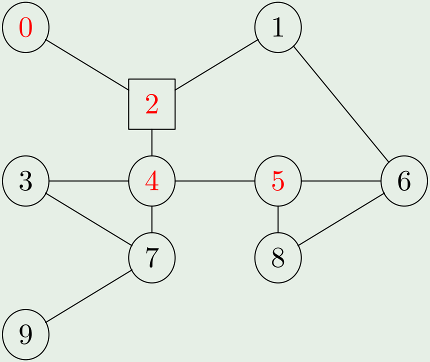
Exemple : Description de l'exploration



`Explores=[5,4,2,0]`

Algorithme de parcours en profondeur

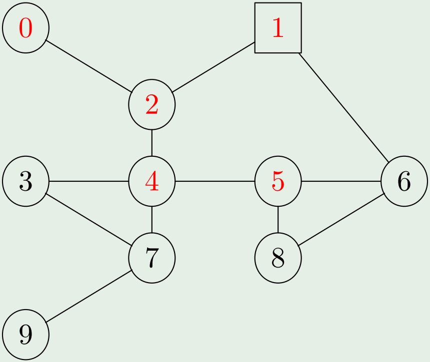
Exemple : Description de l'exploration



`Explores=[5,4,2,0]`

Algorithme de parcours en profondeur

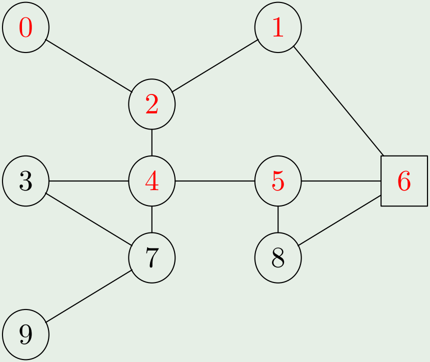
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1]`

Algorithme de parcours en profondeur

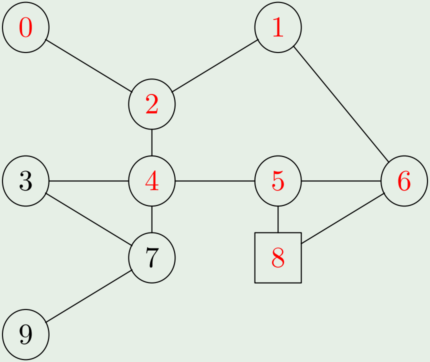
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6]`

Algorithme de parcours en profondeur

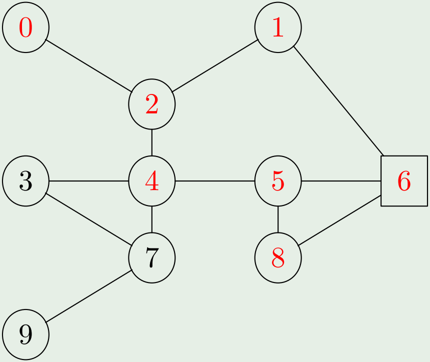
Exemple : Description de l'exploration



Explores=[5,4,2,0,1,6,8]

Algorithme de parcours en profondeur

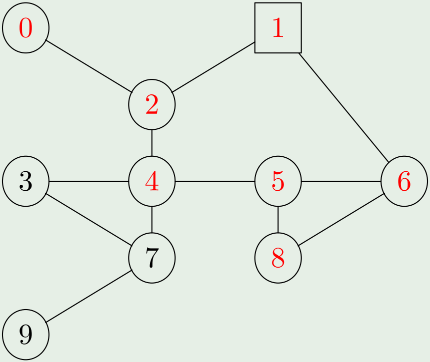
Exemple : Description de l'exploration



Explores=[5,4,2,0,1,6,8]

Algorithme de parcours en profondeur

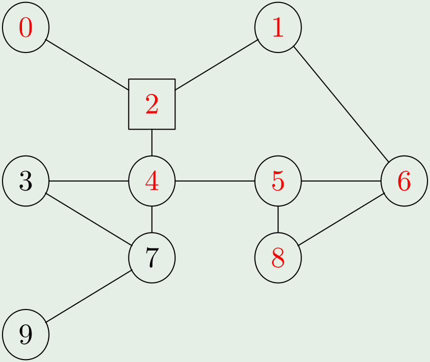
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8]`

Algorithme de parcours en profondeur

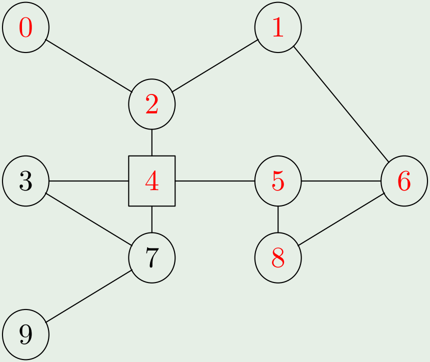
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8]`

Algorithme de parcours en profondeur

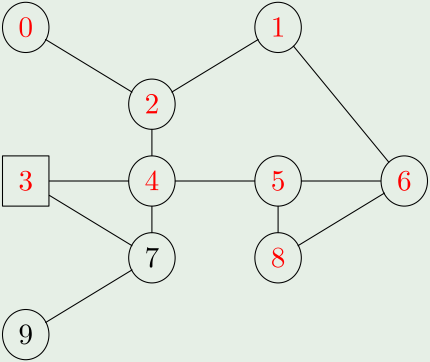
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8]`

Algorithme de parcours en profondeur

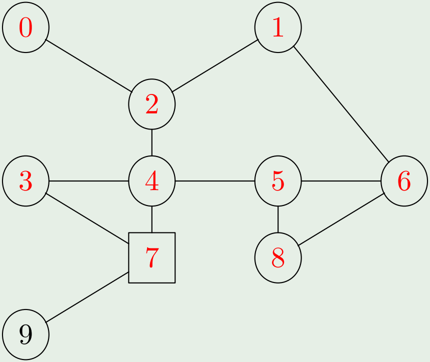
Exemple : Description de l'exploration



Explores=[5,4,2,0,1,6,8,3]

Algorithme de parcours en profondeur

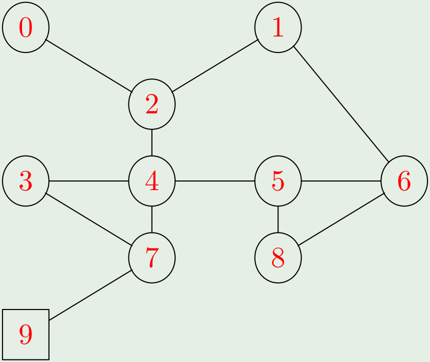
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8,3,7]`

Algorithme de parcours en profondeur

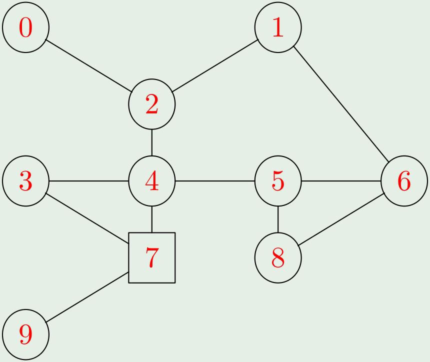
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8,3,7,9]`

Algorithme de parcours en profondeur

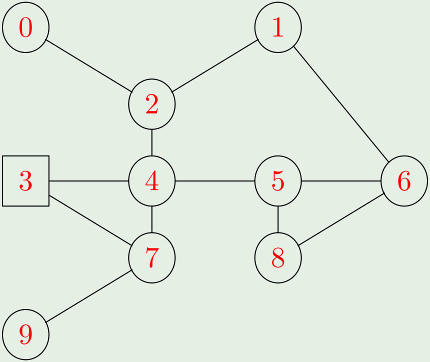
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8,3,7,9]`

Algorithme de parcours en profondeur

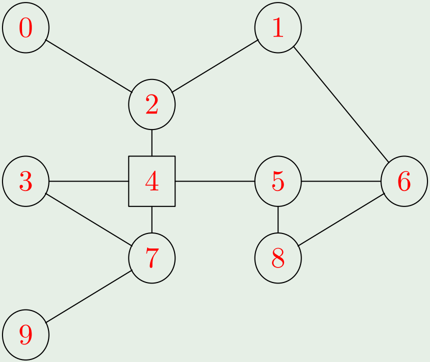
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8,3,9,7]`

Algorithme de parcours en profondeur

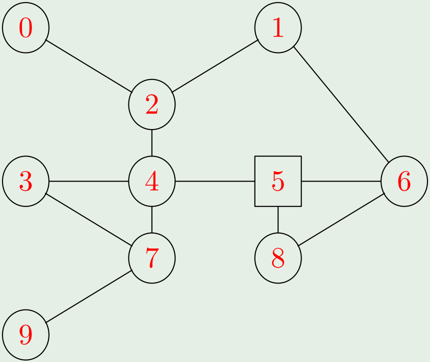
Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8,3,9,7]`

Algorithme de parcours en profondeur

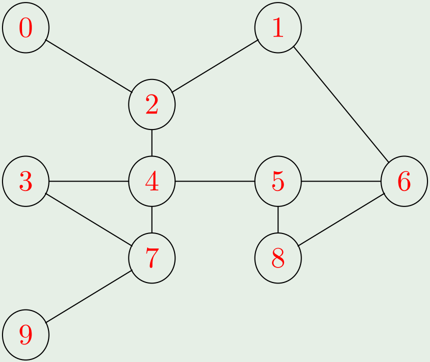
Exemple : Description de l'exploration



Explores=[5,4,2,0,1,6,8,3,9,7]

Algorithme de parcours en profondeur

Exemple : Description de l'exploration



`Explores=[5,4,2,0,1,6,8,3,9,7]`

Algorithme de parcours en profondeur

Exemple : Évolution de la liste d'exploration

`Explores=`

[]	[5,4,2,0,1,6]	[5,4,2,0,1,6,8,3,7]
[5]	[5,4,2,0,1,6,8]	[5,4,2,0,1,6,8,3,7,9]
[5,4]	[5,4,2,0,1,6,8]	[5,4,2,0,1,6,8,3,7,9]
[5,4,2]	[5,4,2,0,1,6,8]	[5,4,2,0,1,6,8,3,9,7]
[5,4,2,0]	[5,4,2,0,1,6,8]	[5,4,2,0,1,6,8,3,9,7]
[5,4,2,0]	[5,4,2,0,1,6,8]	[5,4,2,0,1,6,8,3,9,7]
[5,4,2,0,1]	[5,4,2,0,1,6,8,3]	[5,4,2,0,1,6,8,3,9,7]

Algorithme de parcours en profondeur

Exemple : Évolution de la liste d'exploration

Ou plus simplement :

[]	[5,4,2,1,6]	[5,4,7]
[5]	[5,4,2,1,6,8]	[5,4,3,7,9]
[5,4]	[5,4,2,1,6]	[5,4,3,7]
[5,4,2]	[5,4,2,1]	[5,4,3]
[5,4,2,0]	[5,4,2]	[5,4]
[5,4,2]	[5,4]	[5]
[5,4,2,1]	[5,4,3]	[]

Algorithme de parcours en profondeur

Définition

Une **pile** est une structure de données (par exemple une liste) construite sur le principe «dernier entré, premier sorti» (ou **LIFO** pour *Last In First Out* en anglais).

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.
2. **Mettre** s dans la pile et **marquer** s.

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.
2. **Mettre** s dans la pile et **marquer** s.
3. **Tant que** la pile n'est pas vide :

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.
2. **Mettre** s dans la pile et **marquer** s.
3. **Tant que** la pile n'est pas vide :
    4. **Si** le dernier sommet de la pile a au moins un voisin non-marqué v :

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.
2. **Mettre** s dans la pile et **marquer** s.
3. **Tant que** la pile n'est pas vide :
    4. **Si** le dernier sommet de la pile a au moins un voisin non-marqué v :
        5. **Mettre** v à la fin de la pile et **marquer** v.

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.
2. **Mettre** s dans la pile et **marquer** s.
3. **Tant que** la pile n'est pas vide :
    4. **Si** le dernier sommet de la pile a au moins un voisin non-marqué v :
        5. **Mettre** v à la fin de la pile et **marquer** v.
    6. **Sinon** :

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.
2. **Mettre** s dans la pile et **marquer** s.
3. **Tant que** la pile n'est pas vide :
   4. **Si** le dernier sommet de la pile a au moins un voisin non-marqué v :
      5. **Mettre** v à la fin de la pile et **marquer** v.
   6. **Sinon** :
      7. **Retirer** le dernier sommet de la pile.

Algorithme de parcours en profondeur

Soit $G = (S, A)$ un graphe. On fixe un sommet de départ $s \in S$.

Description de l'algorithme de parcours en profondeur

1. **Créer** une pile vide.
2. **Mettre** s dans la pile et **marquer** s.
3. **Tant que** la pile n'est pas vide :
   4. **Si** le dernier sommet de la pile a au moins un voisin non-marqué v :
      5. **Mettre** v à la fin de la pile et **marquer** v.
   6. **Sinon** :
      7. **Retirer** le dernier sommet de la pile.
   8. **Retourner** à 3.

Algorithme de parcours en profondeur

Convergence

Puisque tout sommet marqué n'est plus exploré, chaque sommet est donc exploré au plus une fois. Par conséquent, l'algorithme de parcours en profondeur s'arrête après un nombre fini d'instructions.

Algorithme de parcours en profondeur

Exemple d'implémentation à l'aide d'une liste d'adjacence :

```python
def dfs_liste(L,s):
    n=len(L) # ordre du graphe
    Marques=[False]*n # initialisation du marquage
    Pile_explores=[s] # étapes 1 et 2
    Marques[s]=True # étape 2
    while Pile_explores!=[]: # étapes 3 et 8
        t=Pile_explores[-1] # dern. som. de la pile
        (v,i)=(None,0) # i=indice des voisins de t
        while v==None and i<len(L[t]): # chercher v
            if not(Marques[L[t][i]]): # étape 4
                v=L[t][i] # trouver v
                Pile_explores.append(v) # étape 5
                Marques[v]=True # étape 5
            i=i+1 # indice suivant
        if v==None: # étape 6
            Pile_explores.remove(t) # étape 7
    # à compléter
```

Algorithme de parcours en profondeur

Exemple d'implémentation à l'aide d'une matrice d'adjacence :

```
def dfs_matrice(M,s):
    n=len(M)
    Marques=[False]*n
    Pile_explores=[s]
    Marques[s]=True
    while Pile_explores!=[]:
        t=Pile_explores[-1]
        (v,i)=(None,0)
        while v==None and i<n: # !!
            if M[t][i]!=0 and not(Marques[i]): # !!
                v=i # !!
                Pile_explores.append(v)
                Marques[v]=True
            i=i+1
        if v==None:
            Pile_explores.remove(t)
    # à compléter
```

Algorithme de parcours en profondeur

On peut aussi utiliser une fonction récursive sans pile :

```
def dfs_recursif_liste(L,s):
    n=len(L)
    Marques=[False]*n
    def explore(t):
        if not(Marques[t]):
            Marques[t]=True
            for v in L[t]:
                explore(v)
    explore(s)
   # à compléter
```

Algorithme de parcours en profondeur

Application : Tester la connexité de G

G est connexe si et seulement si tous ses sommets sont marqués après un parcours en profondeur partant de n'importe quel sommet.

Algorithme de parcours en profondeur

Application : Tester la connexité de G

G est connexe si et seulement si tous ses sommets sont marqués après un parcours en profondeur partant de n'importe quel sommet.

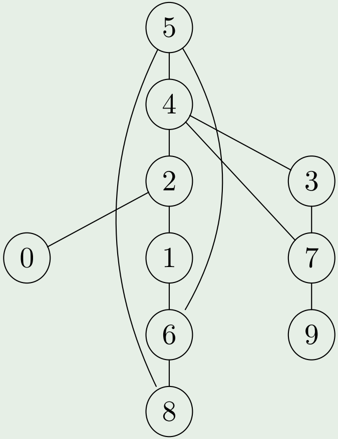
Application : Calculer la composante connexe de s.

La composante connexe de s est l'ensemble des sommets marqués après un parcours en profondeur partant de s.

Algorithme de parcours en profondeur

Par contre, il est plus compliqué de calculer des distances avec un parcours en profondeur qu'avec un parcours en largeur.

Exemple : Représentation équivalente



Algorithme de parcours en profondeur

Application : Tester l'acyclicité de la composante connexe de s

Algorithme de parcours en profondeur

Application : Tester l'acyclicité de la composante connexe de s

Lors d'un parcours en profondeur partant de s, si le dernier sommet de la pile a (au moins) un voisin non-marqué v, il y a deux cas :

Algorithme de parcours en profondeur

Application : Tester l'acyclicité de la composante connexe de s

Lors d'un parcours en profondeur partant de s, si le dernier sommet de la pile a (au moins) un voisin non-marqué v, il y a deux cas :

- v a un seul voisin déjà exploré : le dernier sommet de la pile,

Algorithme de parcours en profondeur

Application : Tester l'acyclicité de la composante connexe de s

Lors d'un parcours en profondeur partant de s, si le dernier sommet de la pile a (au moins) un voisin non-marqué v, il y a deux cas :

- v a un seul voisin déjà exploré : le dernier sommet de la pile,
- v a au moins deux voisins déjà explorés : alors v est dans un cycle.

Algorithme de parcours en profondeur

Application : Tester l'acyclicité de la composante connexe de s

Lors d'un parcours en profondeur partant de s, si le dernier sommet de la pile a (au moins) un voisin non-marqué v, il y a deux cas :

- v a un seul voisin déjà exploré : le dernier sommet de la pile,
- v a au moins deux voisins déjà explorés : alors v est dans un cycle.

Réciproquement, tout cycle de la composante connexe de s est détecté par le deuxième cas : lorsque v est le dernier sommet exploré du cycle.

Algorithme de parcours en profondeur

Application : Tester l'acyclicité de la composante connexe de s

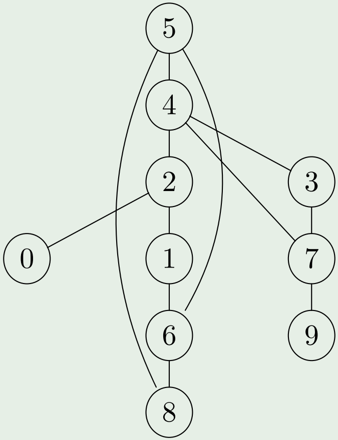
Lors d'un parcours en profondeur partant de s, si le dernier sommet de la pile a (au moins) un voisin non-marqué v, il y a deux cas :

- v a un seul voisin déjà exploré : le dernier sommet de la pile,
- v a au moins deux voisins déjà explorés : alors v est dans un cycle.

Réciproquement, tout cycle de la composante connexe de s est détecté par le deuxième cas : lorsque v est le dernier sommet exploré du cycle.
Par conséquent, la composante connexe de s est acyclique si et seulement si tout voisin non-marqué du dernier sommet de la pile a toujours un seul voisin marqué.

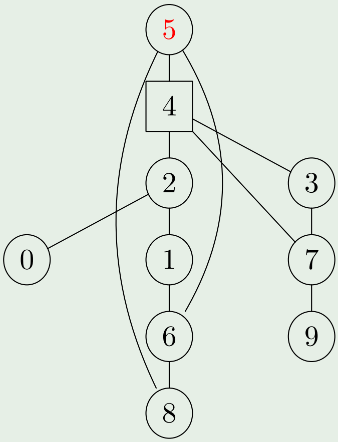
Algorithme de parcours en profondeur

Exemple : Détection de cycle



Algorithme de parcours en profondeur

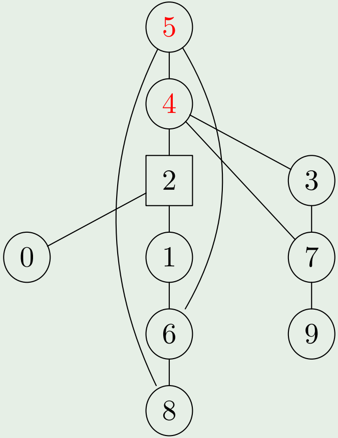
Exemple : Détection de cycle



$\boxed{v = 4}$ a un seul voisin marqué

Algorithme de parcours en profondeur

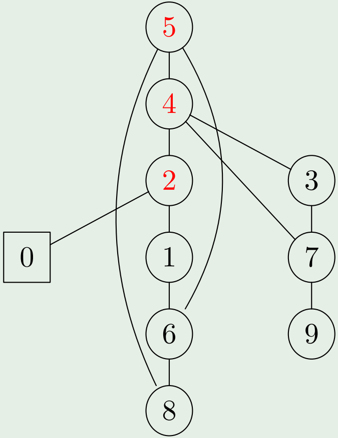
Exemple : Détection de cycle



$\boxed{v = 2}$ a un seul voisin marqué

Algorithme de parcours en profondeur

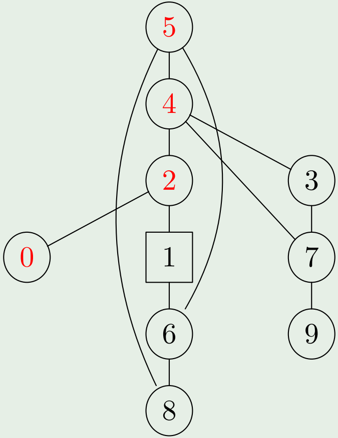
Exemple : Détection de cycle



$\boxed{v = 0}$ a un seul voisin marqué

Algorithme de parcours en profondeur

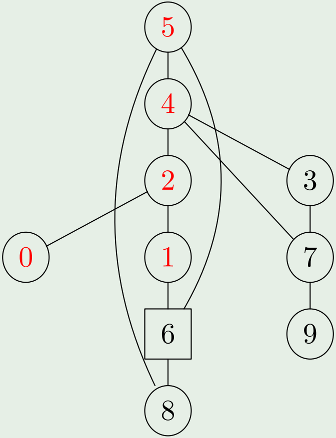
Exemple : Détection de cycle



$\boxed{v = 1}$ a un seul voisin marqué

Algorithme de parcours en profondeur

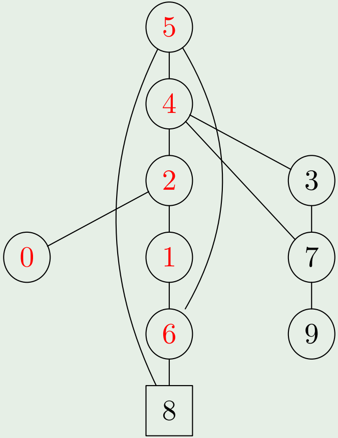
Exemple : Détection de cycle



$\boxed{v = 6}$ a deux voisins marqués : cycle $(6, 1, 2, 4, 5, 6)$

Algorithme de parcours en profondeur

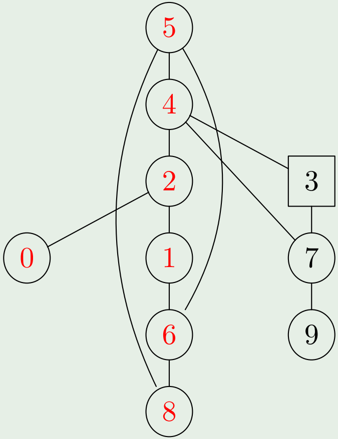
Exemple : Détection de cycle



$\boxed{v = 8}$ a deux voisins marqués : cycle $(8, 5, 6, 8)$

Algorithme de parcours en profondeur

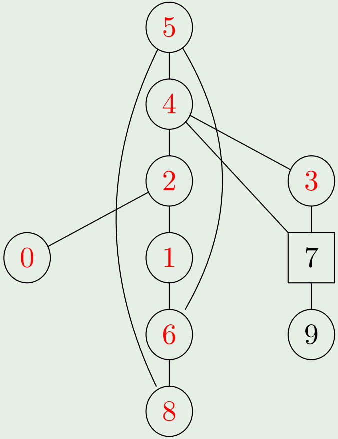
Exemple : Détection de cycle



$\boxed{v = 3}$ a un seul voisin marqué

Algorithme de parcours en profondeur

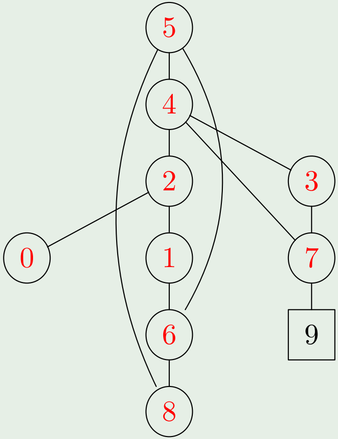
Exemple : Détection de cycle



$\boxed{v = 7}$ a deux voisins marqués : cycle $(7, 3, 4, 7)$

Algorithme de parcours en profondeur

Exemple : Détection de cycle



$\boxed{v = 9}$ a un seul voisin marqué

Algorithme de parcours en profondeur

Exemple d'implémentation d'un test d'acyclicité à l'aide d'un parcours en profondeur et d'une liste d'adjacence.

```
def est_acyclique(L,s):
    n=len(L)
    Marques=[False]*n
    Pile_explores=[s]
    Marques[s]=True
    while Pile_explores!=[]:
        t=Pile_explores[-1]
        (v,i)=(None,0)
        while v==None and i<len(L[t]):
```

Algorithme de parcours en profondeur

```
            if not(Marques[L[t][i]]):
                v=L[t][i]
                nb=0 # nb=nombr. vois. marq. de v
                for j in L[v]:
                    if Marques[j]:
                        nb=nb+1 # calcul de nb
                if nb>=2: # détection d'un cycle
                    return False # pas acyclique
                Pile_explores.append(v)
                Marques[v]=True
            i=i+1
        if v==None:
            Pile_explores.remove(t)
    return True # acyclique
```