

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe
Implémentation

Parcours et structures linéaires

Christophe Viroulaud

Terminale - NSI

Algo 21

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

L'algorithme de parcours en profondeur est récursif. Il est alors naturel de pouvoir l'écrire à nouveau avec une structure linéaire : la **pile**.

Comment implémenter les algorithmes de parcours avec des structures linéaires ?

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

1. Rappels

1.1 Les structures linéaires

1.2 Représentation d'un graphe en mémoire

2. Parcours en profondeur

3. Parcours en largeur



FIGURE 1 – Un nœud

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

Activité 1 :

1. Créer un fichier `structures.py`
2. Écrire la classe `Noeud` et son constructeur qui, lors de l'instanciation, initialisera deux attributs :
 - ▶ `nom`: `int` le nom du nœud,
 - ▶ `suivant`: `Noeud` le successeur dans la structure.
 - ▶ Écrire les accesseurs et les mutateurs des attributs.

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

```
1 class Noeud:
2     def __init__(self, nom: str, suivant: object):
3         self.nom = nom
4         self.suivant = suivant
5
6     def get_nom(self) -> int:
7         return self.nom
8
9     def set_nom(self, n: int) -> None:
10        self.nom = n
11
12    def get_suivant(self) -> object:
13        return self.suivant
14
15    def set_suivant(self, s: object) -> None:
16        self.suivant = s
```

Rappels : les structures linéaires

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

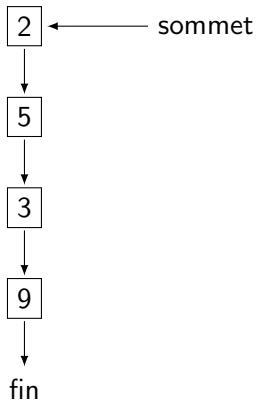


FIGURE 2 – Pile

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

Activité 2 :

1. Écrire la classe `Pile` et son constructeur qui initialisera l'attribut `sommet` à `None`.
2. Écrire la méthode `est_vide` qui renverra `True` si la pile est vide.
3. Écrire la méthode `empiler` qui empilera le nœud nommé `n`: `int` passé en paramètre.
4. Écrire la méthode `depiler` qui dépilera le nœud au sommet de la pile et renverra son nom ou `-1` si la pile est vide.

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

```
1 class Pile:
2     def __init__(self):
3         self.sommet = None
4
5     def est_vide(self) -> bool:
6         return self.sommet is None
```


Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

```
1 def empiler(self, n: int) -> None:
2     self.sommet = Noeud(n, self.sommet)
3
4 def depiler(self) -> int:
5     res = -1
6     if not self.est_vide():
7         res = self.sommet.get_nom()
8         self.sommet = self.sommet.get_suivant()
9     return res
```

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

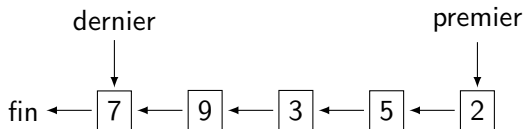


FIGURE 3 – File

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe

Implémentation

Activité 3 :

1. Écrire la classe **File** et son constructeur qui initialisera à **None** deux attributs :
 - ▶ **premier**,
 - ▶ **dernier**.
2. Écrire la méthode **est_vide** qui renverra **True** si la file est vide.
3. Écrire la méthode **enfiler** qui enfilera le nœud nommé **n**: **int** passé en paramètre.
4. Écrire la méthode **defiler** qui défilera le nœud en première place de la file et renverra son nom ou -1 si la file est vide.

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

```
1 class File:
2     def __init__(self):
3         self.premier = None
4         self.dernier = None
5
6     def est_vide(self) -> bool:
7         return self.premier is None
```

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

```
1 def enfiler(self, n: int) -> None:
2     nouveau = Noeud(n, None)
3     if self.est_vide():
4         self.premier = nouveau
5     else:
6         self.dernier.set_suivant(nouveau)
7     self.dernier = nouveau
8
9 def defiler(self) -> int:
10     res = -1
11     if not self.est_vide():
12         res = self.premier.get_nom()
13         self.premier = self.premier.get_suivant()
14     return res
```

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe

Implémentation

1. Rappels

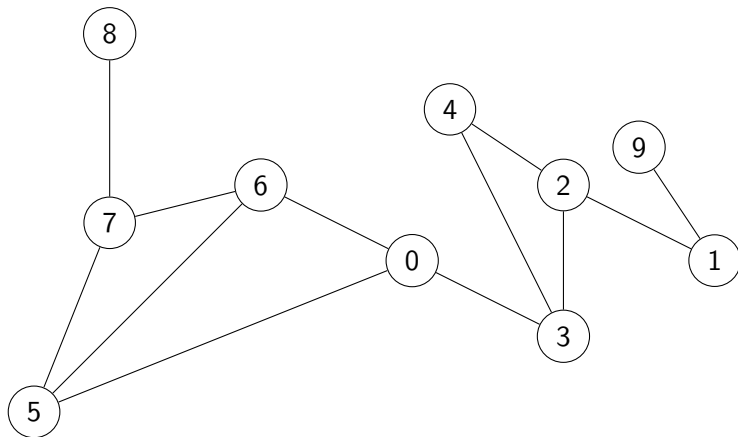
1.1 Les structures linéaires

1.2 Représentation d'un graphe en mémoire

2. Parcours en profondeur

3. Parcours en largeur

Représentation d'un graphe en mémoire



Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe

Implémentation

Activité 4 :

1. Créer le fichier `parcours.py`
2. Écrire la liste d'adjacence du graphe.

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe

Implémentation

```
1 graphe = [  
2     [3, 5, 6],  
3     [2, 9],  
4     [1, 3, 4],  
5     [0, 2, 4],  
6     [2, 3],  
7     [0, 6, 7],  
8     [0, 5, 7],  
9     [5, 6, 8],  
10    [7],  
11    [1]  
12 ]
```


Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

1. Rappels

2. Parcours en profondeur

3. Parcours en largeur

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

À retenir

Lors d'un parcours en profondeur on repart du nœud en cours de visite pour continuer le parcours. On utilise alors une **pile** pour l'implémenter.

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe
Implémentation

Activité 5 :

1. Écrire la fonction `dfs(graphe: list) → list` qui effectue un parcours en profondeur en utilisant une pile. La fonction renverra la liste ordonnée des sommets parcourus.
2. Écrire la fonction `dfs_aretes(graphe: list) → list` qui renvoie les liste ordonnée des arêtes parcourues. La fonction colorera les sommets en :
 - ▶ BLANC au départ,
 - ▶ GRIS lors de l'empilement,
 - ▶ NOIR quand son parcours est terminé.

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe
Implémentation

```
1 def dfs(graphe: list) -> list:
2     parcours = []
3     p = Pile()
4     p.empiler(0)
5     while not p.est_vide():
6         en_cours = p.depiler()
7         if en_cours not in parcours:
8             parcours.append(en_cours)
9
10            for voisin in graphe[en_cours]:
11                p.empiler(voisin)
12    return parcours
```

```
1 >>> dfs(graphe)
2 [0, 6, 7, 8, 5, 3, 4, 2, 1, 9]
```

Code 1 – Appel de la fonction

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe
Implémentation

```
1 def dfs_aretes(graphe: list) -> list:
2     parcours = []
3     etats = [BLANC for _ in range(len(graphe))]
4     p = Pile()
5     p.empiler(0)
6     while not p.est_vide():
7         en_cours = p.depiler()
8         for voisin in graphe[en_cours]:
9             if etats[voisin] == BLANC:
10                 etats[voisin] = GRIS
11                 p.empiler(voisin)
12                 # stockage arête parcourue
13                 parcours.append((en_cours, voisin))
14
15         etats[en_cours] = NOIR
16     return parcours
```

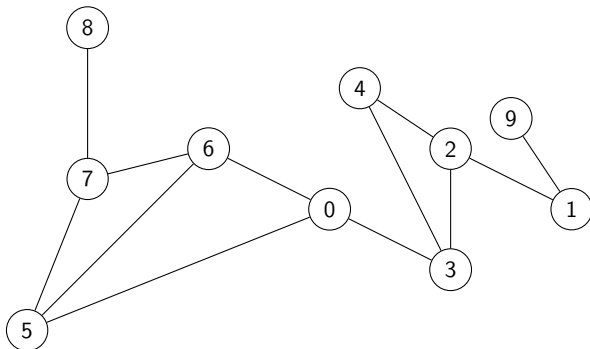
Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation



```
1 >>> dfs_aretes(graphe)
2 [(0, 3), (0, 5), (0, 6), (6, 0), (6, 5), (6, 7),
  (7, 5), (7, 6), (7, 8), (8, 7), (5, 0), (5, 6),
  (5, 7), (3, 0), (3, 2), (3, 4), (4, 2), (4, 3),
  (2, 1), (2, 3), (2, 4), (1, 2), (1, 9), (9, 1)]
```

Code 2 – Chaque sommet n'est empilé qu'une seule fois. Chaque arête est parcourue deux fois (graphe non orienté).

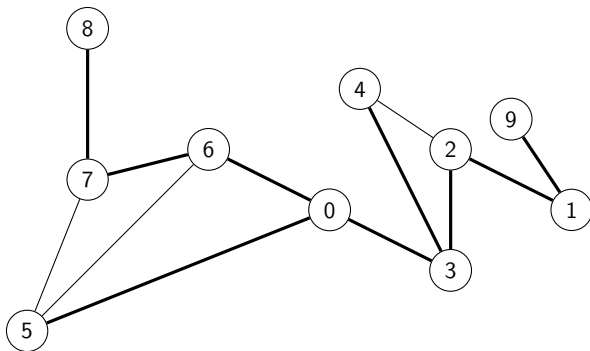
Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation



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

Observation

Si on stocke les arêtes seulement quand on empile les voisins, elles ne sont pas toutes visitées.

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

1. Rappels

2. Parcours en profondeur

3. Parcours en largeur

3.1 Principe

3.2 Implémentation

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

À retenir

Le parcours en largeur visite tous les sommets au rang n avant ceux au rang $n+1$. On utilise une **file** pour l'implémenter.

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe

Implémentation

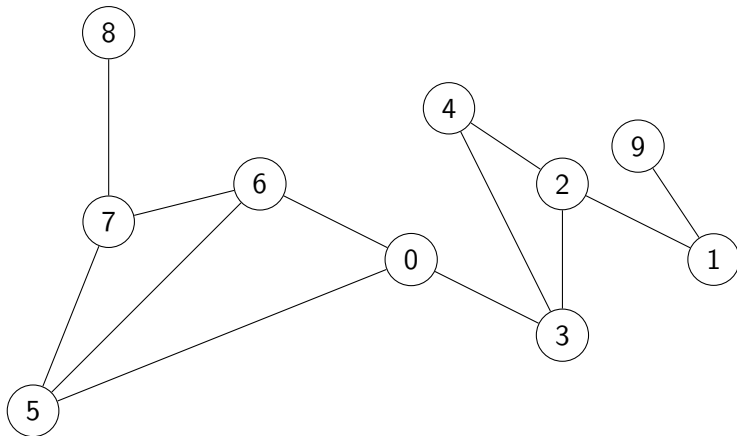


FIGURE 4 – Départ

File : 0

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

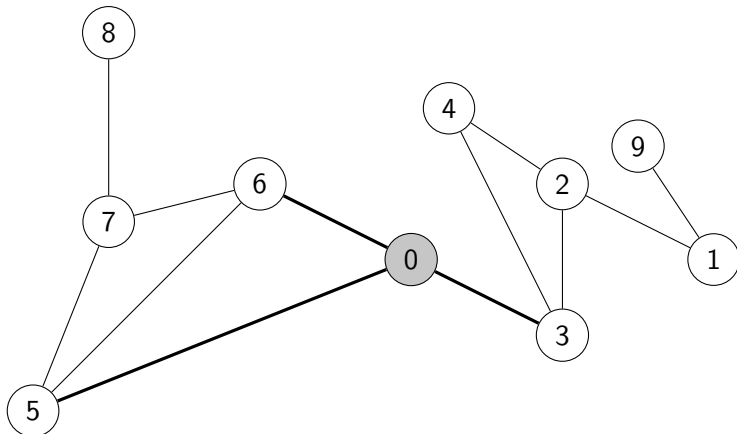


FIGURE 5 – On enfile les voisins à distance 1.

File : $6 \rightarrow 5 \rightarrow 3$

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

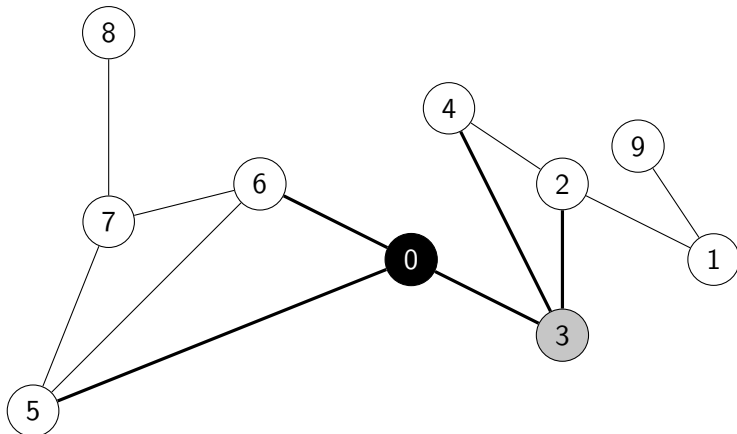


FIGURE 6 – On enfile les voisins à distance 2.

File : $4 \rightarrow 2 \rightarrow 6 \rightarrow 5$

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

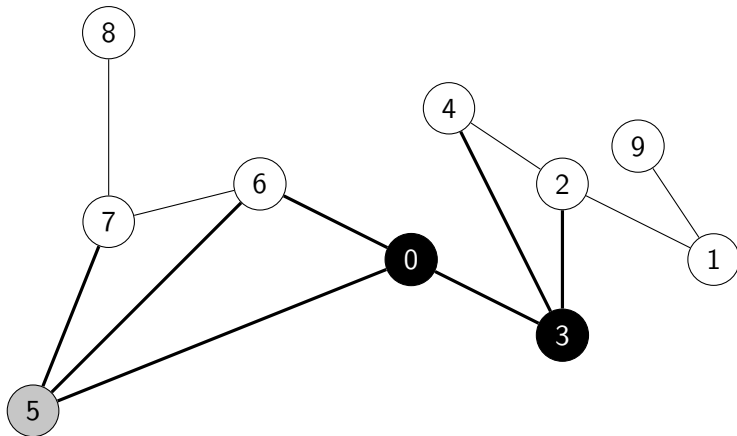


FIGURE 7 – On enfile les voisins à distance 2.

File : $7 \rightarrow 4 \rightarrow 2 \rightarrow 6$

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

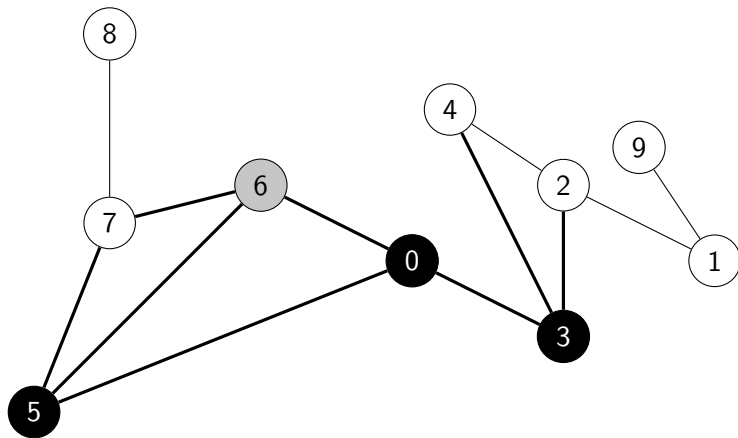


FIGURE 8 – On enfile les voisins à distance 2 : pas de nouveau sommet pour 6.

File : $7 \rightarrow 4 \rightarrow 2$

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

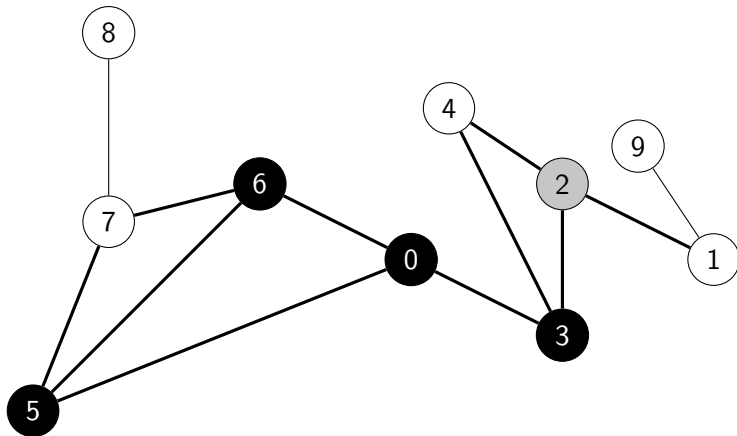


FIGURE 9 – On enfile les voisins à distance 3.

File : $1 \rightarrow 7 \rightarrow 4$

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe
Implémentation

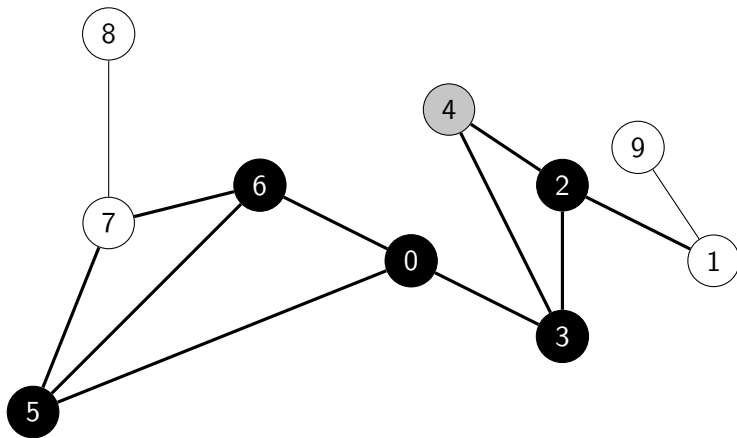


FIGURE 10 – On enfila les voisins à distance 3 : pas de nouveau sommet pour 4.

File : 1 \rightarrow 7

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

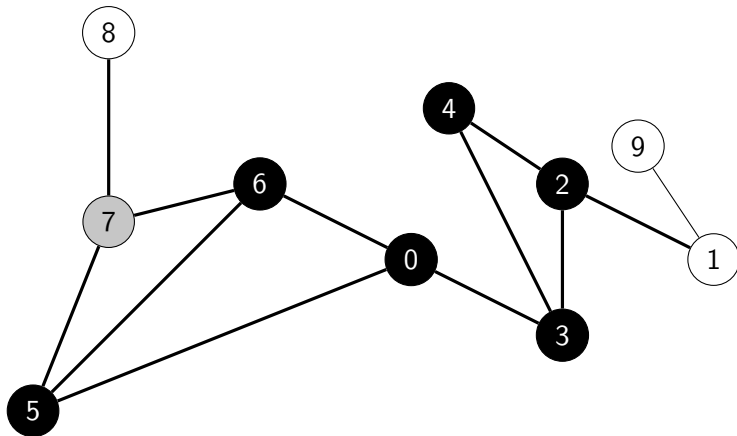


FIGURE 11 – On enfile les voisins à distance 3.

File : 8 \rightarrow 1

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

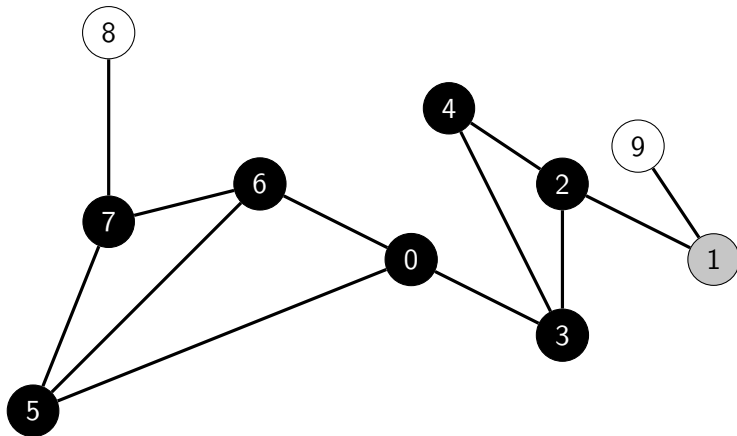


FIGURE 12 – On enfile les voisins à distance 4.

File : 9 \rightarrow 8

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe
Implémentation

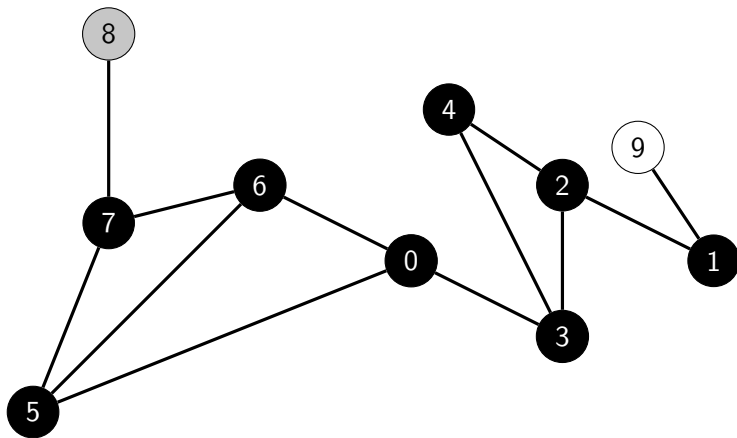


FIGURE 13 – On enfila les voisins à distance 4 : pas de nouveau sommet pour 8.

File : 9

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

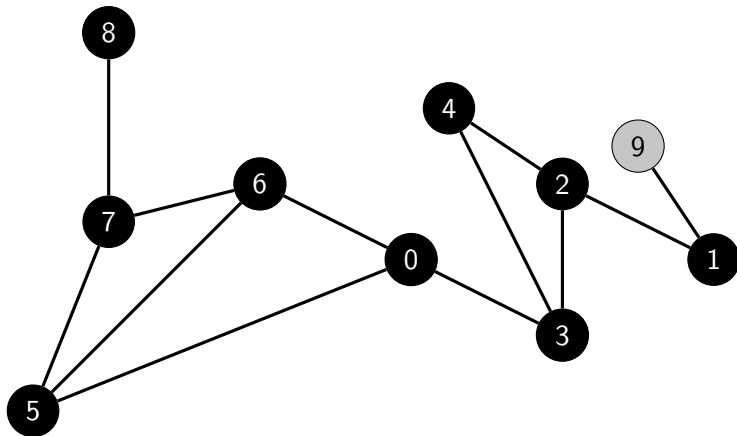


FIGURE 14 – Pas de sommet à distance 5.

File :

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

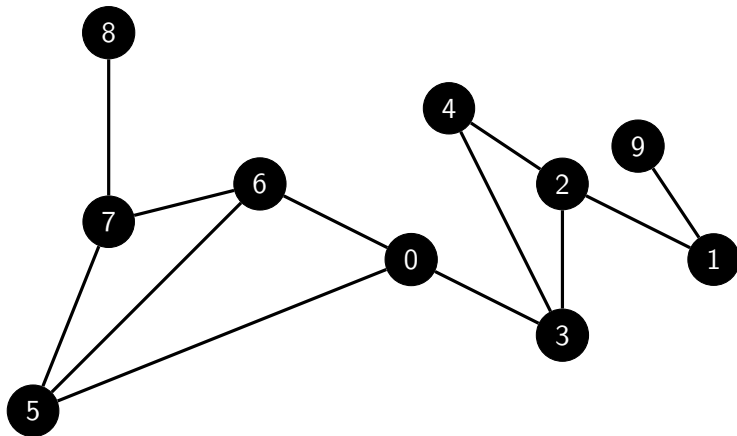


FIGURE 15 – Fin du parcours.

File :

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe

Implémentation

1. Rappels

2. Parcours en profondeur

3. Parcours en largeur

3.1 Principe

3.2 Implémentation

Rappels

Les structures linéaires

Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe

Implémentation

Activité 6 :

1. Écrire la fonction `bfs(graphe: list) → list` qui effectue un parcours en profondeur en utilisant une file. La fonction renverra la liste ordonnée des sommets parcourus.
2. Écrire la fonction `bfs_aretes(graphe: list) → list` qui renvoie les liste ordonnée des arêtes parcourues. La fonction colorera les sommets en :
 - ▶ BLANC au départ,
 - ▶ GRIS lors de l'empilement,
 - ▶ NOIR quand son parcours est terminé.

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

```
1 def bfs(graphe: list) -> list:
2     parcours = []
3     f = File()
4     f.enfiler(0)
5     while not f.est_vide():
6         en_cours = f.defiler()
7         if en_cours not in parcours:
8             parcours.append(en_cours)
9
10            for voisin in graphe[en_cours]:
11                f.enfiler(voisin)
12    return parcours
```

```
1 >>> bfs(graphe)
2 [0, 3, 5, 6, 2, 4, 7, 1, 8, 9]
```

Code 3 – Appel de la fonction

Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation

```
1 def bfs_aretes(graphe: list) -> list:
2     parcours = []
3     etats = [BLANC for _ in range(len(graphe))]
4     f = File()
5     f.enfiler(0)
6     etats[0]=GRIS
7     while not f.est_vide():
8         en_cours = f.defiler()
9         for voisin in graphe[en_cours]:
10             if etats[voisin] == BLANC:
11                 etats[voisin] = GRIS
12                 f.enfiler(voisin)
13
14             # stockage arête parcourue
15             parcours.append((en_cours, voisin))
16             etats[en_cours] = NOIR
17     return parcours
```

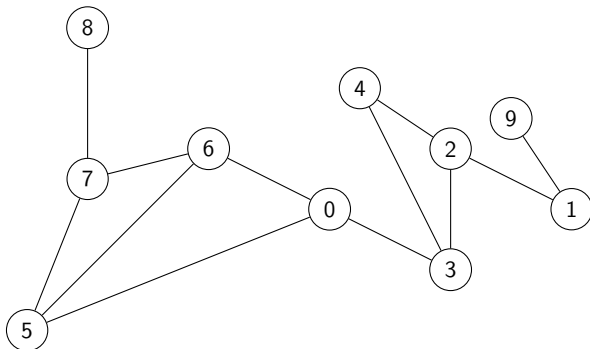
Rappels

Les structures linéaires
Représentation d'un graphe
en mémoire

Parcours en profondeur

Parcours en largeur

Principe
Implémentation



```
1 >>> bfs_aretes(graphe)
2 [(0, 3), (0, 5), (0, 6), (3, 0), (3, 2), (3, 4),
  (5, 0), (5, 6), (5, 7), (6, 0), (6, 5), (6, 7),
  (2, 1), (2, 3), (2, 4), (4, 2), (4, 3), (7, 5),
  (7, 6), (7, 8), (1, 2), (1, 9), (8, 7), (9, 1)]
```

Code 4 – Chaque sommet n'est enfilé qu'une seule fois. Chaque arête est parcourue deux fois (graphe non orienté).

Rappels

Les structures linéaires

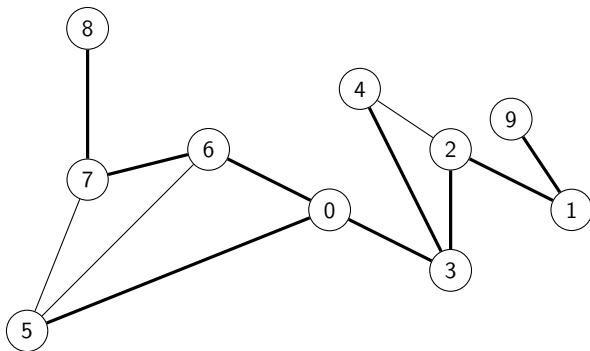
Représentation d'un graphe
en mémoire

Parcours en
profondeur

Parcours en largeur

Principe

Implémentation



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

Observation

Si on stocke les arêtes seulement quand on enfila les voisins, elles ne sont pas toutes visitées.