

# Exercices parcours graphe

## Correction

Christophe Viroulaud

Terminale - NSI

**Algo 20**

# Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

Exercice 1

Exercice 2

Exercice 3

# Exercice 1

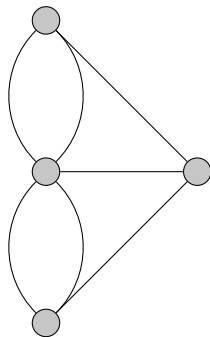


FIGURE 1 – Les sept ponts de Königsberg

Tous les sommets sont de degré impair. Il n'est pas possible de réaliser un cycle eulérien.

Exercice 1

Exercice 2

Exercice 3

# Sommaire

1. Exercice 1

2. Exercice 2

3. Exercice 3

Exercice 1

Exercice 2

Exercice 3

## Exercice 2

Exercice 1

Exercice 2

Exercice 3

- ▶ L'ordre est 8.
- ▶ Le degré de D est 4.

```
1 graphe = {"A": ["B", "C", "D"],
2         "B": ["A", "D", "E"],
3         "C": ["A", "D", "F", "H"],
4         "D": ["A", "B", "C", "G"],
5         "E": ["B", "F"],
6         "F": ["C", "E"],
7         "G": ["D"],
8         "H": ["C"]}
```

```
1 def profondeur_dict(graphe: dict, noeud: str,  
    visites: dict) -> None:  
2     if not visites[noeud]:  
3         print(noeud, end=" ")  
4         visites[noeud] = True  
5         for voisin in graphe[noeud]:  
6             profondeur_dict(graphe, voisin, visites)
```

Code 1 – Création de la fonction

```
1 visites_dico = {chr(65+i): False for i in range(8)}  
2 profondeur_dict(graphe, "A", visites_dico)
```

Code 2 – Appel de la fonction

Exercice 1

Exercice 2

Exercice 3

```
1 def get_indice(sommet: str) -> int:  
2     return ord(sommet)-65
```



```
1 def profondeur_tab(graphe: dict, noeud: str, visites
  : list) -> None:
2     ind = get_indice(noeud)
3     if not visites[ind]:
4         print(noeud, end=" ")
5         visites[ind] = True
6         for voisin in graphe[noeud]:
7             profondeur_tab(graphe, voisin, visites)
```

Code 3 – Création de la fonction

```
1 visites_tab = [False for i in range(8)]
2 profondeur_tab(graphe, "A", visites_tab)
```

Code 4 – Appel de la fonction

## Observation

Les deux fonctions construites permettent de s'affranchir du coût de parcours du tableau `visites` à chaque appel récursif. En effet, la vérification de la valeur associée à une clé dans un dictionnaire ou celle de la lecture du booléen dans le tableau s'effectuent en temps constant.

# Sommaire

1. Exercice 1
2. Exercice 2
3. Exercice 3

Exercice 1

Exercice 2

Exercice 3

# Exercice 3

Exercice 1

Exercice 2

Exercice 3

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

```
1 def dfs(graphe: list, sommet: int, visites: list) -> None:
2     # en cours de parcours
3     visites[sommet]["coul"] = GRIS
4     for voisin in graphe[sommet]:
5         # pour chaque voisin non encore atteint
6         if visites[voisin]["coul"] == BLANC:
7             visites[voisin]["pred"] = sommet
8             dfs(graphe, voisin, visites)
9     # parcours terminé pour ce sommet
10    visites[sommet]["coul"] = NOIR
```

ce 1

ce 2

ce 3

## Remarque

La vérification de la couleur s'effectue avant l'appel récursif (ligne 6) au lieu de se faire en début de fonction ; ceci afin de ne modifier le prédécesseur qu'à la première visite.