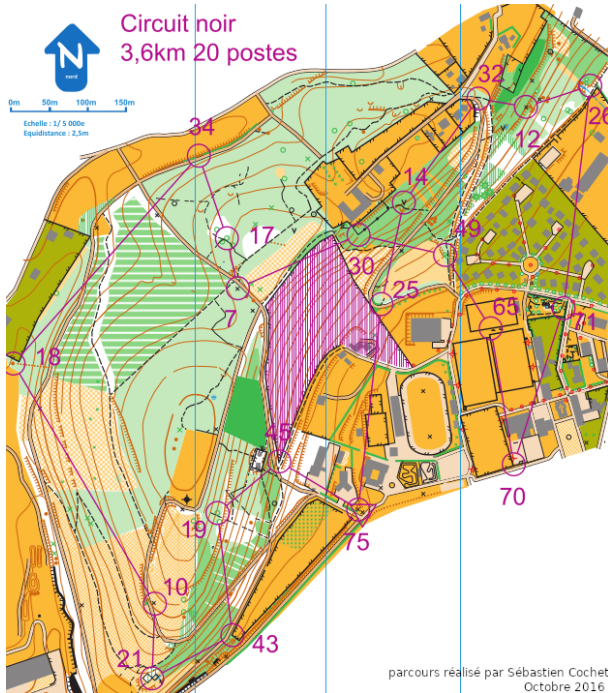


Course d'orientation Connexité

Christophe Viroulaud

Terminale - NSI

Algo 17



Parcours en
profondeur (Depth
First Search)

Principe
Complexité
Implémentation

Connexité

Définition
Vérification de la connexité

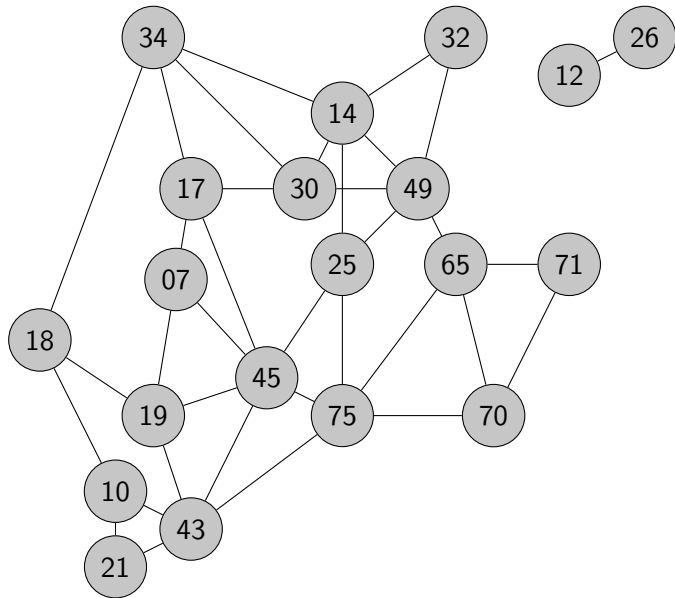


FIGURE 1 – Des chemins impraticables à cause de la météo.

Comment vérifier si tous les sommets sont atteignables ?

1. Parcours en profondeur (Depth First Search)

1.1 Principe

1.2 Complexité

1.3 Implémentation

2. Connexité

Parcours en
profondeur (Depth
First Search)

Principe

Complexité

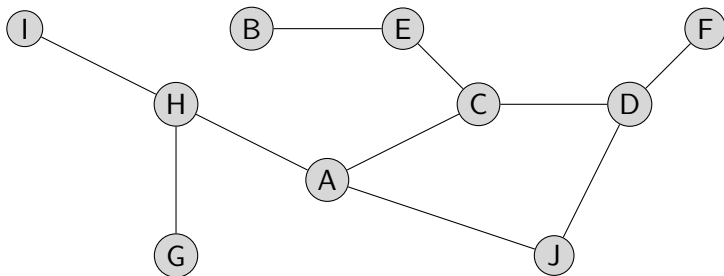
Implémentation

Connexité

Définition

Vérification de la connexité

Parcours en profondeur (Depth First Search) - Principe



À retenir

Un parcours en profondeur *avance* dans le graphe jusqu'à une extrémité ou un nœud déjà visité. Il revient alors à un sommet précédent qui propose un autre chemin.

Parcours en
profondeur (Depth
First Search)

Principe
Complexité
Implémentation

Connexité

Définition
Vérification de la connexité

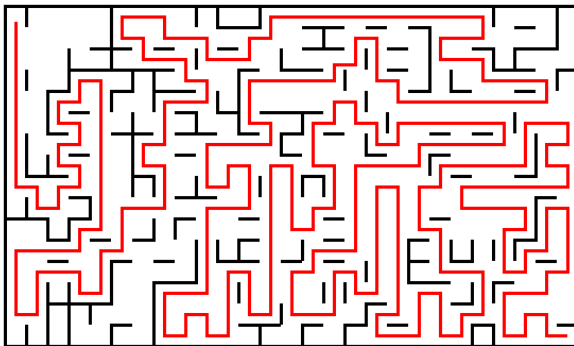


FIGURE 2 – Parcours en profondeur dans un labyrinthe

Parcours en
profondeur (Depth
First Search)

Principe

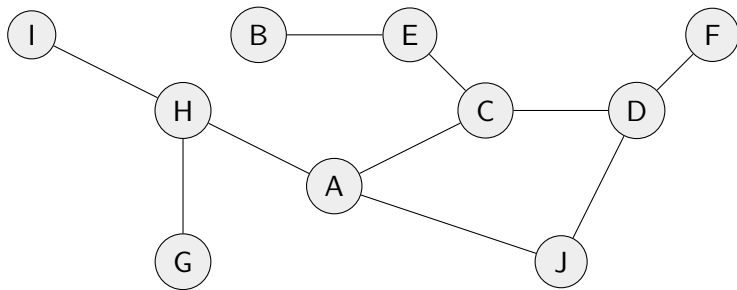
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



Parcours en
profondeur (Depth
First Search)

Principe

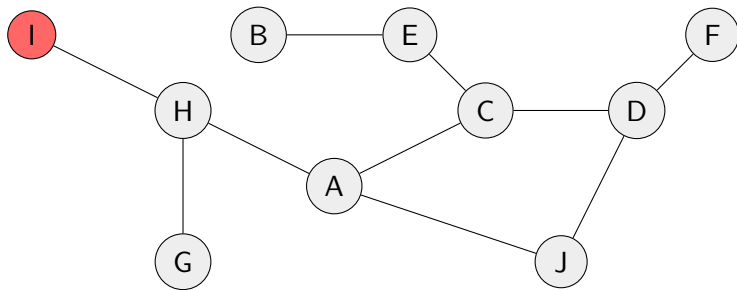
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



Parcours en
profondeur (Depth
First Search)

Principe

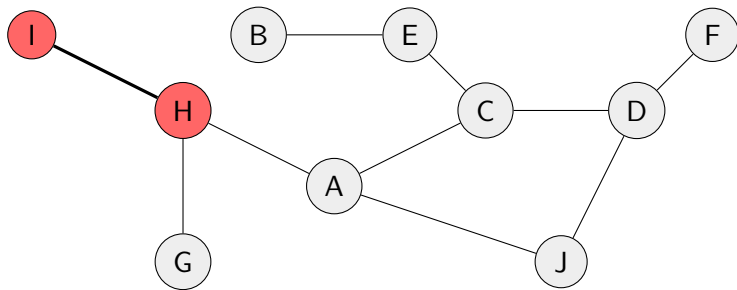
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



Parcours en
profondeur (Depth
First Search)

Principe

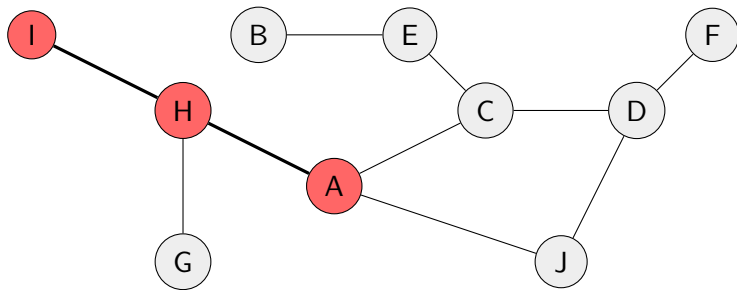
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



Parcours en
profondeur (Depth
First Search)

Principe

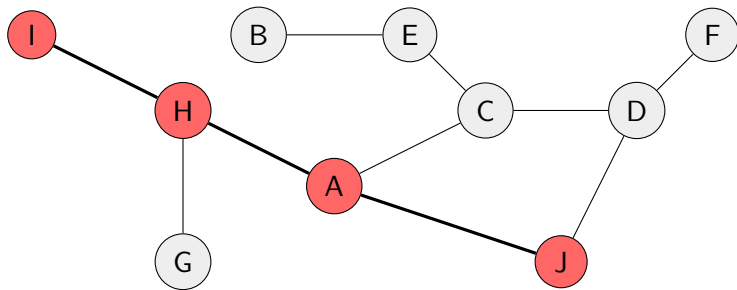
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



Parcours en
profondeur (Depth
First Search)

Principe

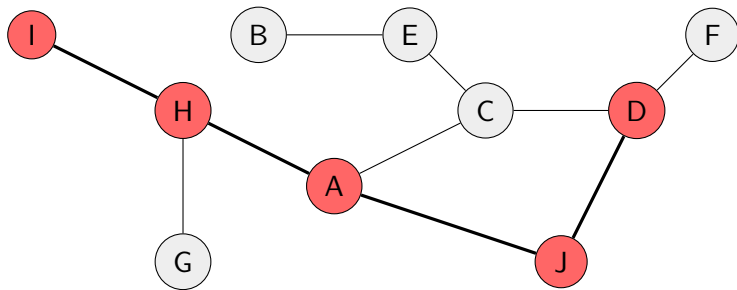
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



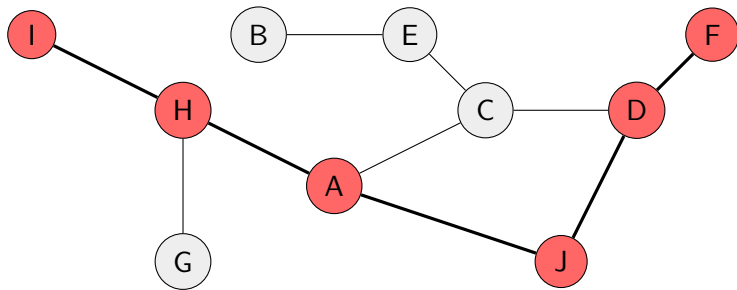


FIGURE 3 – Retour au nœud précédent

Parcours en
profondeur (Depth
First Search)

Principe

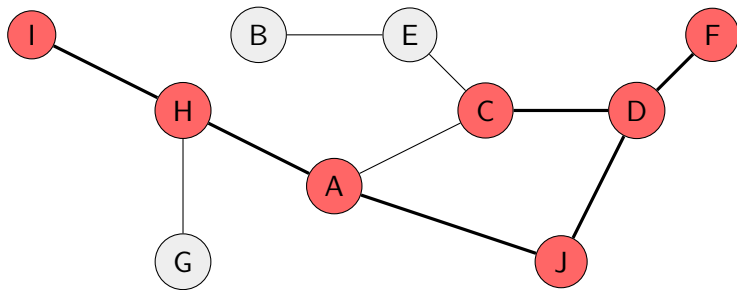
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



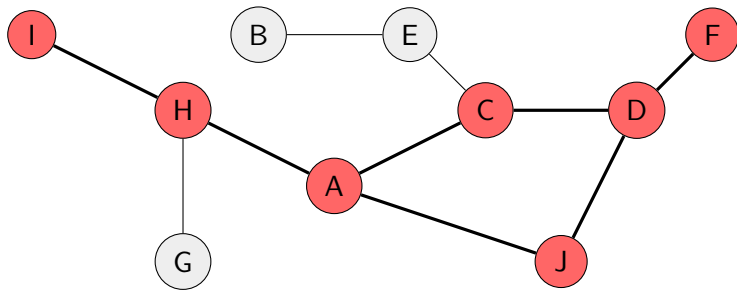


FIGURE 4 – A est déjà visité.

Parcours en
profondeur (Depth
First Search)

Principe

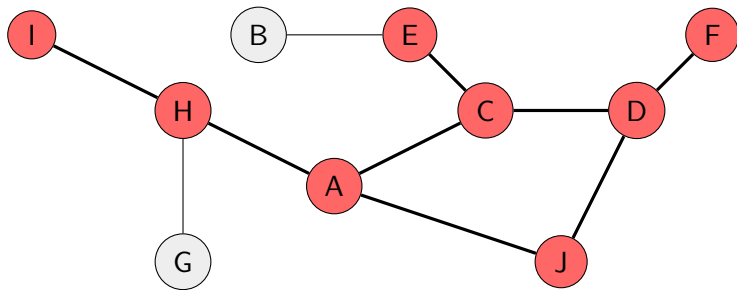
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



Parcours en
profondeur (Depth
First Search)

Principe

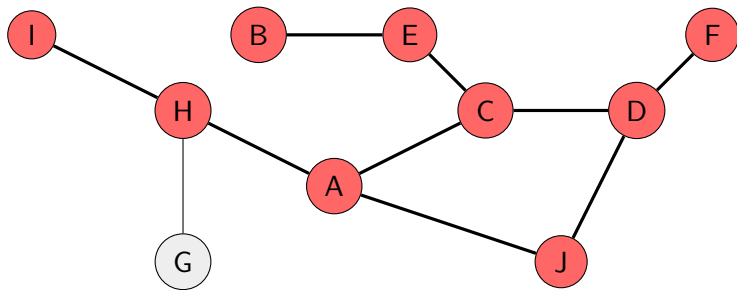
Complexité

Implémentation

Connexité

Définition

Vérification de la connexité



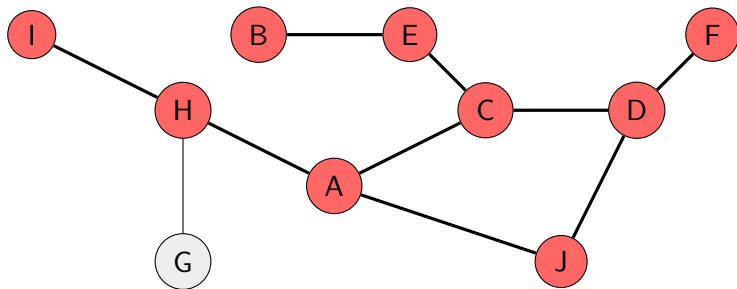


FIGURE 5 – Retour au nœud précédent

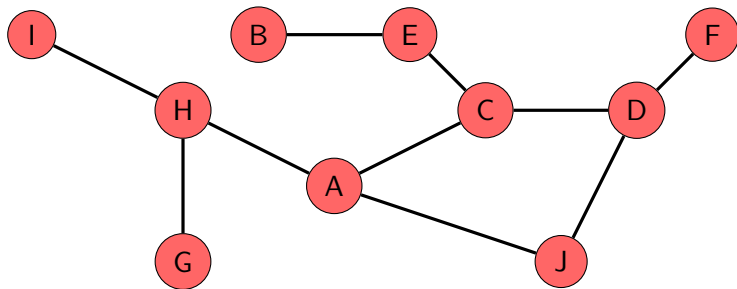


FIGURE 6 – Fin du parcours

1. Parcours en profondeur (Depth First Search)

1.1 Principe

1.2 Complexité

1.3 Implémentation

2. Connexité

Parcours en
profondeur (Depth
First Search)

Principe

Complexité

Implémentation

Connexité

Définition

Vérification de la connexité

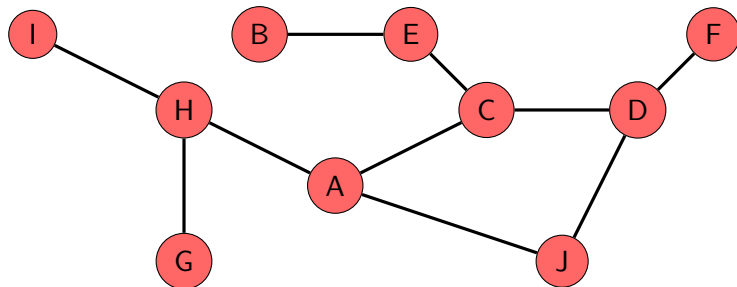


FIGURE 7 – Fin du parcours

À retenir

- On ne parcourt chaque arête qu'une seule fois.

Parcours en
profondeur (Depth
First Search)

Principe
Complexité
Implémentation

Connexité

Définition
Vérification de la connexité

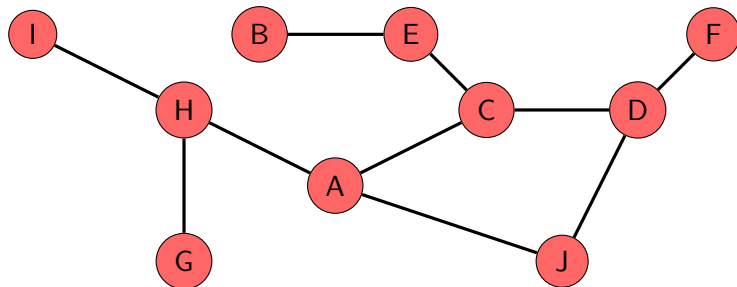


FIGURE 7 – Fin du parcours

À retenir

- ▶ On ne parcourt chaque arête qu'une seule fois.
- ▶ On ne marque chaque nœud qu'une fois.

Parcours en
profondeur (Depth
First Search)

Principe

Complexité

Implémentation

Connexité

Définition

Vérification de la connexité

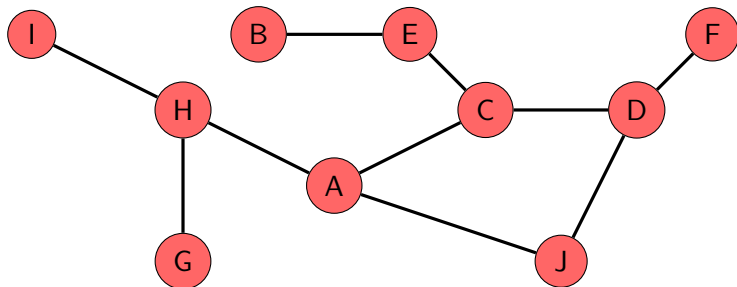


FIGURE 7 – Fin du parcours

À retenir

- ▶ On ne parcourt chaque arête qu'une seule fois.
- ▶ On ne marque chaque nœud qu'une fois.
- ▶ La complexité du parcours en profondeur dépend du nombre de nœuds et du nombre d'arêtes du graphe.

Parcours en
profondeur (Depth
First Search)

Principe

Complexité

Implémentation

Connexité

Définition

Vérification de la connexité

1. Parcours en profondeur (Depth First Search)

1.1 Principe

1.2 Complexité

1.3 Implémentation

2. Connexité

Parcours en
profondeur (Depth
First Search)

Principe

Complexité

Implémentation

Connexité

Définition

Vérification de la connexité

À retenir

Le parcours en profondeur est naturellement récursif. On choisit un nœud de départ pour commencer le parcours, puis :

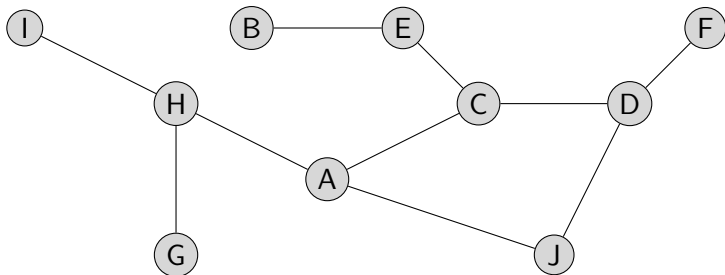
- ▶ Si le nœud n'est pas déjà visité :
 - ▶ faire quelque chose avec le nœud (afficher...)
 - ▶ le marquer *visité*,
 - ▶ pour tous les nœuds voisins :
 - ▶ effectuer le parcours récursivement depuis le voisin sélectionné.

Parcours en
profondeur (Depth
First Search)

Principe
Complexité
Implémentation

Connexité

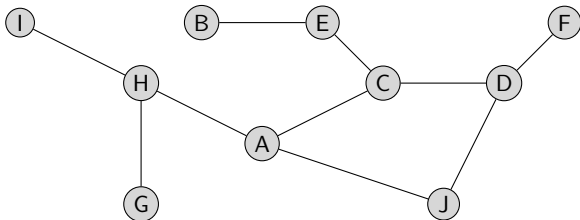
Définition
Vérification de la connexité



Activité 1 :

1. Construire le dictionnaire d'adjacence du graphe.
2. Écrire la fonction récursive `profondeur(graphe: dict, noeud: str, visites: list) → None` qui effectue un parcours en profondeur du `graphe`. À chaque appel, le nœud traversé sera affiché dans la console.

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



```

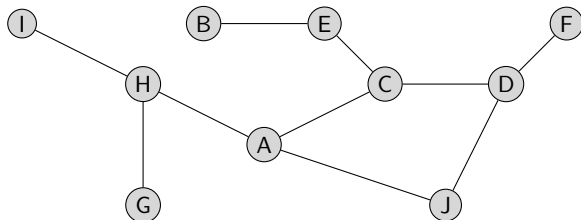
1 def profondeur(graphe: dict, noeud: str, visites: list) -> None:
2     if noeud not in visites:
3         # on fait quelque chose avec le noeud
4         print(noeud)
5         visites.append(noeud)
6         for voisin in graphe[noeud]:
7             profondeur(graphe, voisin, visites)

```

```

1 >>> profondeur(graphe, "I", [])
2 I H A C D F J E B G

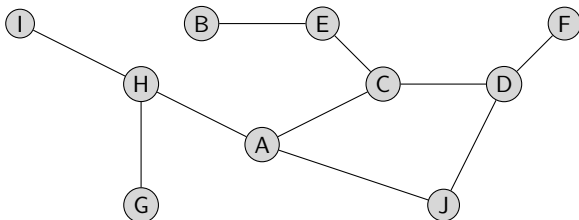
```



```
1  if noeud not in visites:
```

Observations

- Cette vérification garantit que chaque nœud et chaque arête ne sont visités qu'une fois.



```
1  if noeud not in visites:
```

Observations

- ▶ Cette vérification garantit que chaque nœud et chaque arête ne sont visités qu'une fois.
- ▶ Cependant l'implémentation masque un coût : le parcours du tableau `visites`.

Parcours en
profondeur (Depth
First Search)

Principe

Complexité

Implémentation

Connexité

Définition

Vérification de la connexité

1. Parcours en profondeur (Depth First Search)

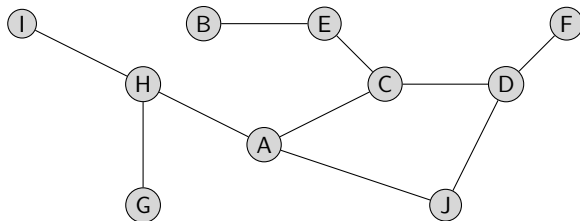
2. Connexité

2.1 Définition

2.2 Vérification de la connexité

À retenir

Dans un graphe, une **chaîne** reliant deux sommets x et y est une suite d'arêtes consécutives reliant x à y .



Parcours en
profondeur (Depth
First Search)

Principe
Complexité
Implémentation

Connexité

Définition
Vérification de la connexité

À retenir

Dans un graphe, une **chaîne** reliant deux sommets x et y est une suite d'arêtes consécutives reliant x à y .

À retenir

Un graphe est **connexe** quand tout sommet peut être relié à tout autre par une chaîne.

Parcours en
profondeur (Depth
First Search)

Principe

Complexité

Implémentation

Connexité

Définition

Vérification de la connexité

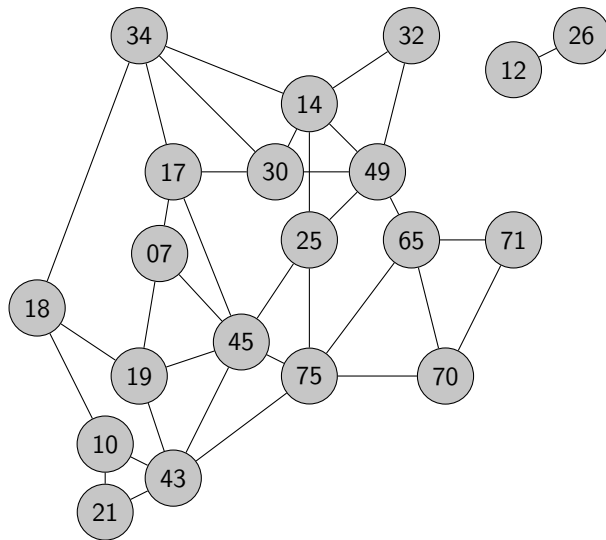
1. Parcours en profondeur (Depth First Search)

2. Connexité

2.1 Définition

2.2 Vérification de la connexité

Vérification de la connexité



Parcours en
profondeur (Depth
First Search)

Principe
Complexité
Implémentation

Connexité

Définition
Vérification de la connexité

FIGURE 8 – Un graphe non orienté est connexe si le parcours en profondeur peut atteindre tous les sommets.

Activité 2 :

1. Reprendre le fichier `parcours_noir.json` et le modifier pour supprimer les arêtes :
 - ▶ 32-12,
 - ▶ 26-71.
2. Dans le fichier Python `parcours_noir.py` construire alors le dictionnaire `graphe`.
3. Écrire la fonction `est_connexe(graphe: dict, depart: int) → bool` qui vérifie si le graphe est connexe. La fonction appellera la fonction `profondeur` est vérifiera si le nombre de sommets visités en partant du sommet `depart`, est égal à la taille de `graphe`.

```
1 def est_connexe(graphe: dict, depart: int) -> bool:
2     visites = []
3     profondeur(graphe, depart, visites)
4     return len(graphe) == len(visites)
```

```
1 >>> est_connexe(graphe, 70)
2 False
```

Code 1 – Appel de la fonction