Course d'orientation Connexité

Christophe Viroulaud

Terminale - NSI

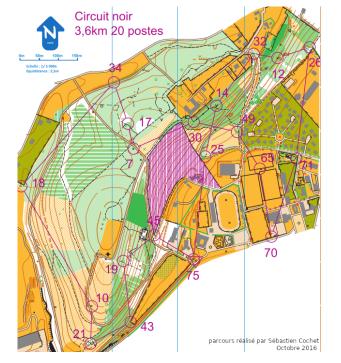
Algo 17

Parcours en profondeur (Depth First Search)

Complexité Implémentation

Connexite

Définition

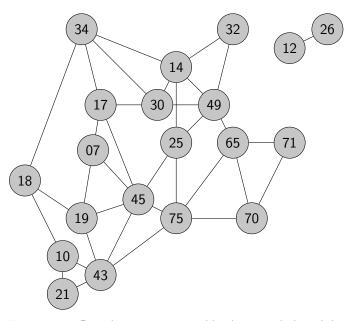


arcours en rofondeur (Depth irst Search)

Comployitá

Implémentation

Connexité



 $\ensuremath{\mathrm{Figure}}\ 1$ – Des chemins impraticables à cause de la météo.

arcours en rofondeur (Depth irst Search)

Complexité

Connexité

Définition

Comment vérifier si tous les sommets sont atteignables?

Sommaire

Course d'orientation Connexité

Parcours en profondeur (Depth First Search)

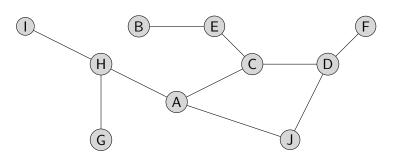
Implémentatio

Connevit

Définition

- 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



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.

Course d'orientation Connexité

Parcours en profondeur (Depth First Search)

Principe Compleyité

Complexité

Connexité

Définition



Principe Complexité

Complexite

Définition

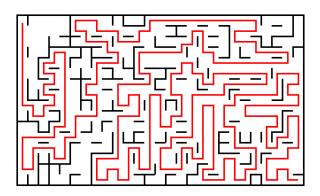


FIGURE 2 – Parcours en profondeur dans un labyrinthe

Parcours en rofondeur (Depth First Search)

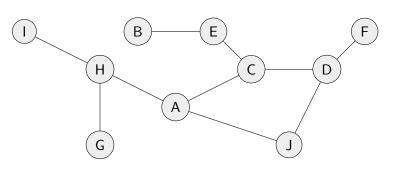
Principe

Complexit

Implément

Jonnesite

Définition



'arcours en rofondeur (Depth 'irst Search)

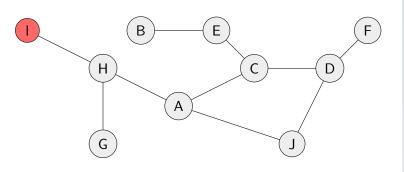
Principe

Landina

Implémenta

50111107110

Definition



Parcours en profondeur (Depth First Search)

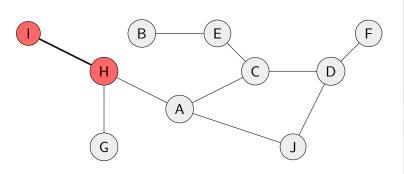
Principe

Complexite

Implementation

Connexite

Définition



Parcours en profondeur (Depth First Search)

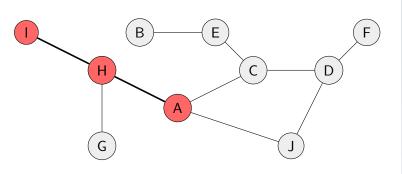
Principe

Complexit

Implémenta

Lonnexite

Définition



Parcours en profondeur (Depth First Search)

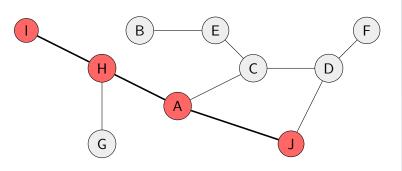
Principe

Complexit

Implémenta

.01111071101

Definition



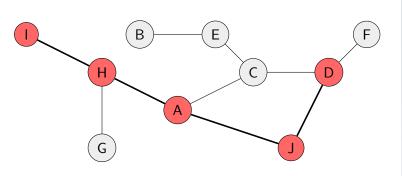
Parcours en profondeur (Depth First Search)

Principe

Complexit

Implémenta

OTHICKIE



'arcours en rofondeur (Depth 'irst Search)

Principe Complexité

Complexité

Connexité

éfinition

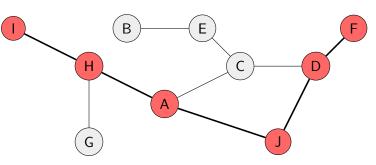


FIGURE 3 – Retour au nœud précédent

Parcours en profondeur (Depth First Search)

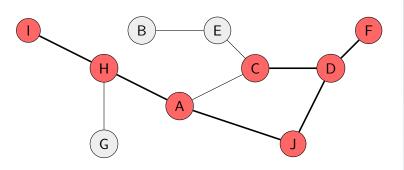
Principe

lanal farance

impiementation

...

Várification do la connevitá



arcours en rofondeur (Depth irst Search)

Principe Complexité

Lomplexite

Community

Définition

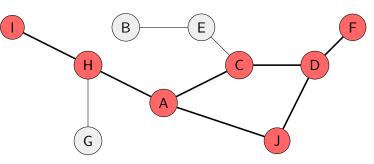


FIGURE 4 – A est déjà visité.

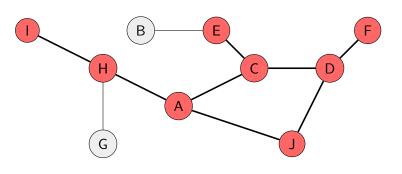
arcours en rofondeur (Depth irst Search)

Principe

Complexit

Implémenta

éfinition



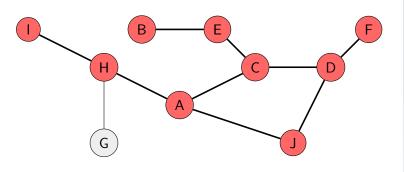
Parcours en profondeur (Depth First Search)

Principe

Complexit

Implémenta

éfinition



arcours en rofondeur (Depth irst Search)

Principe Comployitá

Complexité

Affinition

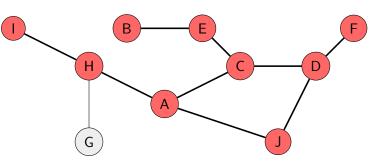


FIGURE 5 – Retour au nœud précédent

Parcours en profondeur (Depth First Search)

Principe

Complexite

Implementa

......

etinition

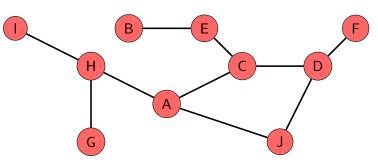


FIGURE 6 – Fin du parcours

Sommaire

- Course d'orientation Connexité
- Parcours en profondeur (Depth First Search)
 - Complexité

Implémenta

OTTICALL

- 1. Parcours en profondeur (Depth First Search)
- 1.1 Principe
- 1.2 Complexité
- 1.3 Implémentation
- 2. Connexité

Complexité

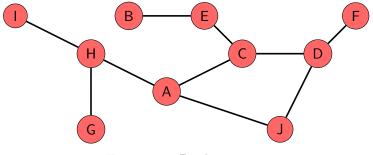


FIGURE 7 – Fin du parcours

À retenir

On ne parcourt chaque arête qu'une seule fois.

Course d'orientation Connexité

Complexité

Complexité

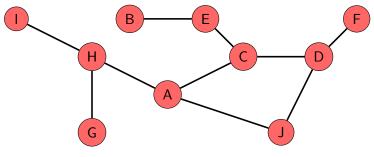


FIGURE 7 – Fin du parcours

À retenir

- On ne parcourt chaque arête qu'une seule fois.
- La complexité du parcours en profondeur dépend du nombre d'arêtes du graphe.

Course d'orientation Connexité

Complexité

Sommaire

- Course d'orientation Connexité
- Parcours en profondeur (Depth First Search)
 - Complexité
- Implémentation

Lonnexite

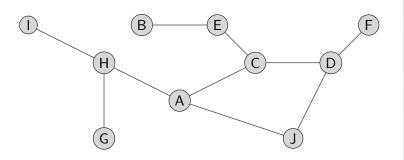
Définition

- 1. Parcours en profondeur (Depth First Search)
- 1.1 Principe
- 1.2 Complexité
- 1.3 Implémentation
- 2. 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.



Activité 1 :

- 1. Construire le dictionnaire d'adjacence du graphe.
- 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.

Course d'orientation Connexité

'arcours en rofondeur (Depth 'irst Search)

Complexité

Affinition

```
Parcours en
profondeur (Depth
First Search)
```

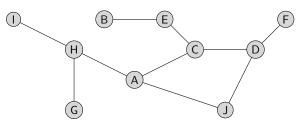
```
Complexité
Implémentation
```

C-----

Définition

lárification do la connevi

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



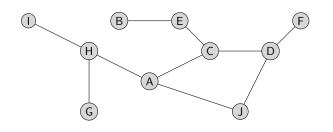
```
Course
d'orientation
Connexité
```

Parcours en profondeur (Depth First Search)

Complexité Implémentation

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

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



if noeud not in visites:

Observations

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

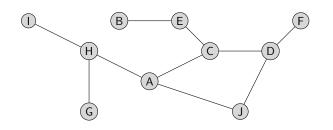
Course d'orientation Connexité

'arcours en rofondeur (Depth 'irst Search)

Complexité

Implémentation

Définition



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.

Course d'orientation Connexité

'arcours en rofondeur (Depth 'irst Search)

Complexité

Connevité

Définition

Sommaire

Course d'orientation Connexité

Parcours en profondeur (Depth First Search)

Implémentation

Connexité

Définitio

Vérification de la connevite

1. Parcours en profondeur (Depth First Search)

- 2. Connexité
- 2.1 Définition
- 2.2 Vérification de la connexité

Connexité - Définition

Course d'orientation Connexité

Parcours en profondeur (Depth First Search)

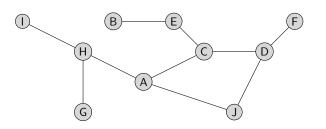
Complexité

Connexité

Définition

À retenir

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



Connexité - Définition

Course d'orientation Connexité

Parcours en profondeur (Depth First Search)

Implémentation

Définition

Várification de la connevit

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.

Sommaire

- Course d'orientation Connexité

- Vérification de la connexité

- 2. Connexité
- 2.2 Vérification de la connexité

Vérification de la connexité

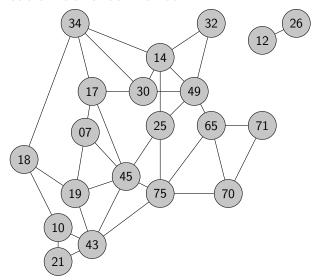


FIGURE 8 – Un graphe non orienté est connexe si le parcours en profondeur peut atteindre tous les sommets.

Course d'orientation Connexité

arcours en rofondeur (Depth irst Search)

Complexité

Connexi

- 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.

profondeur (Depth First Search)

Complexité

Implémentation

- ...

Définition

4

```
profondeur (Depth
First Search)
```

Connexité

Définition Vérification de la connexité

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

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

Code 1 – Appel de la fonction