

Partie III – Parcours de graphes

Graphes et Algorithmes – GRAAL

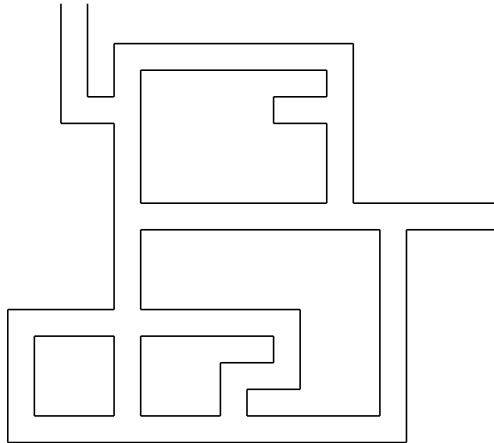
Gaëtan Staquet
gaetan.staquet@ec-nantes.fr

École Centrale de Nantes – LS2N
S508

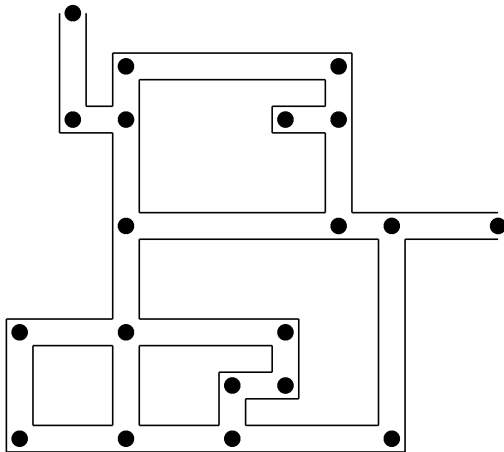
Janvier à mars 2026

1. Parcours en profondeur d'abord
2. Composantes fortement connexes
3. Parcours en largeur d'abord et chemin minimal

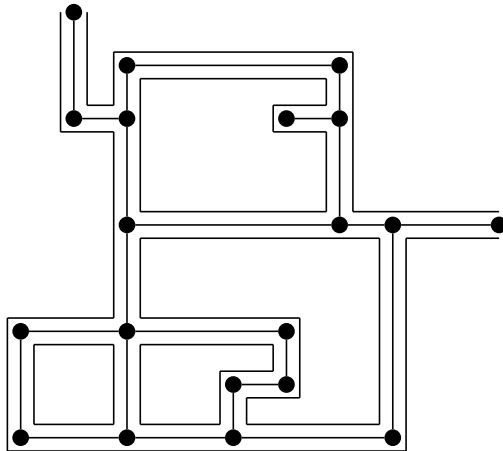
Trouver son chemin



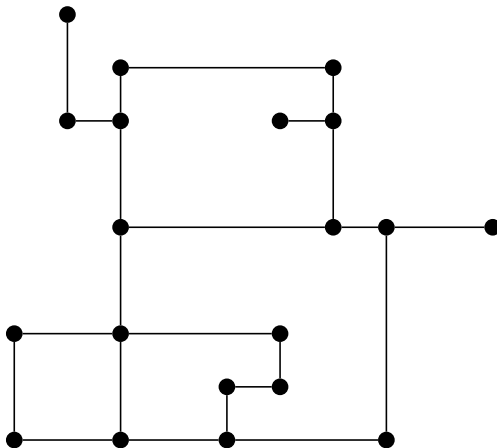
Trouver son chemin



Trouver son chemin

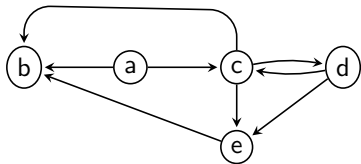


Trouver son chemin

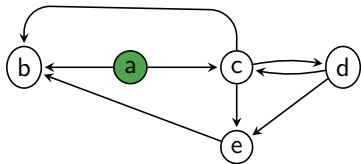


1. Parcours en profondeur d'abord
2. Composantes fortement connexes
3. Parcours en largeur d'abord et chemin minimal

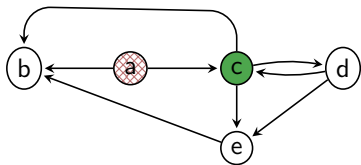
Parcours en profondeur d'abord – Ordre d'exploration



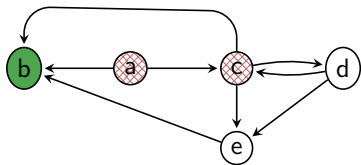
Parcours en profondeur d'abord – Ordre d'exploration



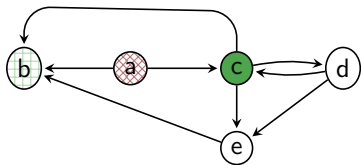
Parcours en profondeur d'abord – Ordre d'exploration



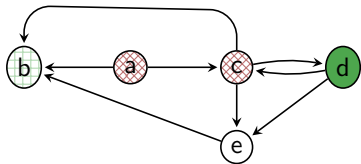
Parcours en profondeur d'abord – Ordre d'exploration



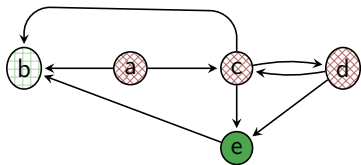
Parcours en profondeur d'abord – Ordre d'exploration



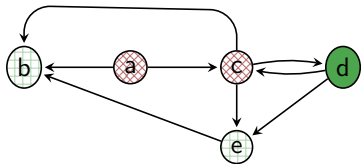
Parcours en profondeur d'abord – Ordre d'exploration



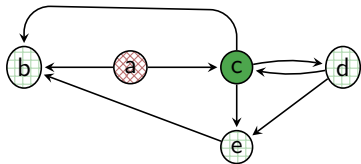
Parcours en profondeur d'abord – Ordre d'exploration



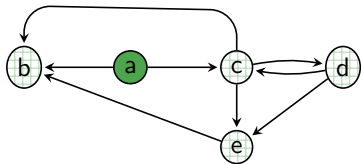
Parcours en profondeur d'abord – Ordre d'exploration



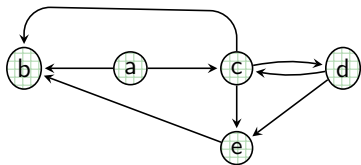
Parcours en profondeur d'abord – Ordre d'exploration



Parcours en profondeur d'abord – Ordre d'exploration



Parcours en profondeur d'abord – Ordre d'exploration



Tous les sommets sont atteignables depuis a.



Pile

Définition III.1 (Pile). Une **pile** est une structure de données **dernier arrivé, premier servi**¹. Typiquement, une pile est implémentée comme une liste avec les opérations :

- ▶ **ajouter** pour insérer un élément à la fin ;
- ▶ **retirer** pour retirer et retourner l'élément à la fin.

Pensez à une pile de livres, de vêtements, *etc.* Pour arriver à prendre l'élément tout en bas, il faut d'abord bouger tout ce qui est au-dessus.

Remarque III.2. Les appels de fonction sont gérés via une pile. En particulier, si la fonction est réursive, on a une pile « gratuitement ».

1. En anglais : LIFO pour last in, first out

Parcours en profondeur d'abord – Algorithme

Nécessite : $\mathcal{G} = (V, E), v \in V$

```
1 : procédure PARCOURSPROFONDEUR( $\mathcal{G}, v$ )  
2 :   retourner PARCOURSPROFONDEURRÉCURSIF( $\mathcal{G}, v, \emptyset$ )
```

Nécessite : $V_{\text{us}} \subseteq V$

```
3 : procédure PARCOURSPROFONDEURRÉCURSIF( $\mathcal{G}, v, V_{\text{us}}$ )  
4 :    $V_{\text{us}} \leftarrow V_{\text{us}} \cup \{v\}$   
5 :   pour chaque successeur  $s$  de  $v$  dans  $\mathcal{G}$  faire  
6 :     si  $s \notin V_{\text{us}}$  alors  
7 :        $V_{\text{us}} \leftarrow \text{PARCOURSPROFONDEURRÉCURSIF}(\mathcal{G}, s, V_{\text{us}})$   
8 :   retourner  $V_{\text{us}}$ 
```

En anglais, le **parcours en profondeur d'abord** est appelé **depth-first search**, abrégé en **DFS**.

Parcours en profondeur d'abord – Algorithme souple

```
1 : procédure PARCOURSPROFONDEUR( $\mathcal{G}, v$ )
2 :   prévisite( $v$ )
3 :    $V_{us} \leftarrow \text{PARCOURSPROFONDEURRÉCURSIF}(\mathcal{G}, v, \emptyset)$ 
4 :   retourner  $V_{us}$ 
5 : postvisite( $v$ )
6 : procédure PARCOURSPROFONDEURRÉCURSIF( $\mathcal{G}, v, V_{us}$ )
7 :   visite( $v$ )
8 :    $V_{us} \leftarrow V_{us} \cup \{v\}$ 
9 :   pour chaque successeur  $s$  de  $v$  dans  $\mathcal{G}$  faire
10 :     si  $s \notin V_{us}$  alors
11 :       prévisite( $s$ )
12 :        $V_{us} \leftarrow \text{PARCOURSPROFONDEURRÉCURSIF}(\mathcal{G}, s, V_{us})$ 
13 :       postvisite( $s$ )
14 :     sinon
15 :       revisite( $s$ )
16 :   retourner  $V_{us}$ 
```

DFS est souple : on peut faire ce qu'on veut grâce à **prévisite**, **visite**, **revisite** et **postvisite**.

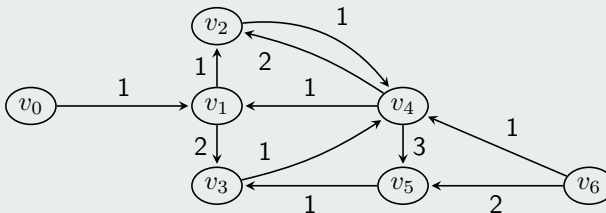
Parcours en profondeur d'abord – Quelques résultats

Lemme III.3. Soit v le sommet de départ. Un sommet u est visité lors du parcours en profondeur d'abord si et seulement s'il existe un chemin de v à u .

Lemme III.4. Si un sommet est postvisité, alors tous ses successeurs ont été vus (prévisités ou visités).

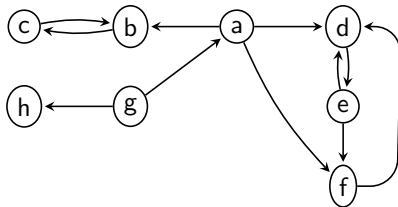
Parcours en profondeur – Exercices

Exercice III.5. Prenons le graphe suivant

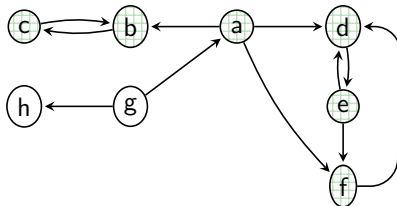


- ▶ Quel est l'ensemble des sommets qui sont atteignables depuis v_0 ?
- ▶ Appliquez DFS à partir de v_0 . Le chiffre à côté de chaque arc donne l'ordre d'itérations sur les successeurs. Par exemple, à partir de v_1 , v_2 est vu avant v_3 .

Parcours en profondeur – Sommets manquants



Parcours en profondeur – Sommets manquants



Un parcours ne suffit pas toujours. Il faut relancer sur les sommets qui n'ont pas été visités.

↪ Parcours en profondeur **étendu**.

```

1 : procédure PARCOURS PROFONDEUR ÉTENDU( $\mathcal{G} = (V, E)$ )
2 :   Visités  $\leftarrow \emptyset$ 
3 :   tant que il existe un sommet  $v \in V \setminus \text{Visités}$  faire
4 :      $\text{Visités} \leftarrow \text{Visités} \cup \text{PARCOURS PROFONDEUR}(\mathcal{G}, v, \text{Visités})$ 
  
```

Parcours en profondeur – Complexité

Chaque **sommet est visité une et une seule fois**.

Pour chaque sommet visité, on parcourt tous les arcs sortants. Donc, dans un graphe orienté, chaque **arc est emprunté une et une seule fois**.

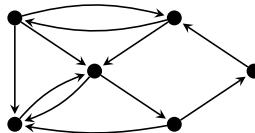
En conclusion, la complexité est $\mathcal{O}(|V| + |E|)$, *i.e.*, linéaire.²

2. Pour rappel, $|E| \leq |V|^2$. On peut donc aussi dire que la complexité du DFS est en $\mathcal{O}(|V|^2)$.

1. Parcours en profondeur d'abord
2. Composantes fortement connexes
3. Parcours en largeur d'abord et chemin minimal

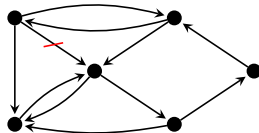
Des rues à sens unique

On a une ville dont les quartiers sont reliés par des rues à **sens unique**. On désire fermer certaines rues pour faire des travaux, tout en s'assurant de ne **pas isoler** des quartiers.



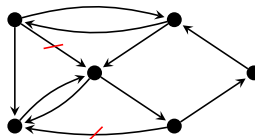
Des rues à sens unique

On a une ville dont les quartiers sont reliés par des rues à **sens unique**. On désire fermer certaines rues pour faire des travaux, tout en s'assurant de ne **pas isoler** des quartiers.



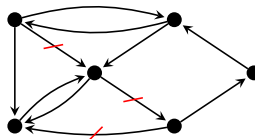
Des rues à sens unique

On a une ville dont les quartiers sont reliés par des rues à **sens unique**. On désire fermer certaines rues pour faire des travaux, tout en s'assurant de ne **pas isoler** des quartiers.



Des rues à sens unique

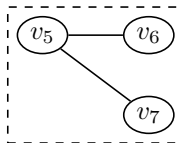
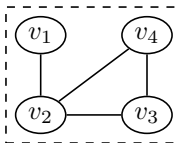
On a une ville dont les quartiers sont reliés par des rues à **sens unique**. On désire fermer certaines rues pour faire des travaux, tout en s'assurant de ne **pas isoler** des quartiers.



Graphes non orientés et composantes connexes

Définition III.6 (Graphe non orienté connexe). Un graphe $\mathcal{G} = (V, E)$ **non orienté** est dit **connexe** si, pour toute paire de sommets u, v , il existe une chaîne qui relie u et v .

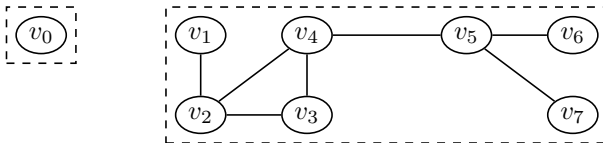
Définition III.7 (Composante connexe). Une **composante connexe** d'un graphe est un sous-graphe **induit**, **connexe** et **maximal** (c'est-à-dire qu'on ne peut pas rajouter de sommet sans rendre le sous-graphe non connexe).



Graphes non orientés et composantes connexes

Définition III.6 (Graphe non orienté connexe). Un graphe $\mathcal{G} = (V, E)$ **non orienté** est dit **connexe** si, pour toute paire de sommets u, v , il existe une chaîne qui relie u et v .

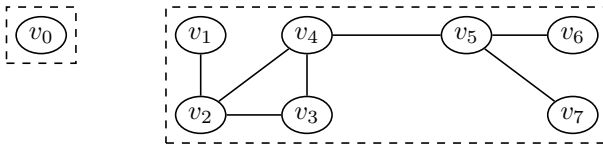
Définition III.7 (Composante connexe). Une **composante connexe** d'un graphe est un sous-graphe **induit**, **connexe** et **maximal** (c'est-à-dire qu'on ne peut pas rajouter de sommet sans rendre le sous-graphe non connexe).



Graphes non orientés et composantes connexes

Définition III.6 (Graphe non orienté connexe). Un graphe $\mathcal{G} = (V, E)$ **non orienté** est dit **connexe** si, pour toute paire de sommets u, v , il existe une chaîne qui relie u et v .

Définition III.7 (Composante connexe). Une **composante connexe** d'un graphe est un sous-graphe **induit**, **connexe** et **maximal** (c'est-à-dire qu'on ne peut pas rajouter de sommet sans rendre le sous-graphe non connexe).

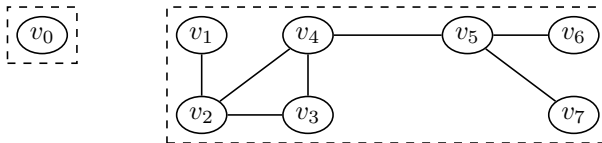


Exercice III.8. Soit un graphe non orienté $\mathcal{G} = (V, E)$. Prouvez que, si \mathcal{G} a un sous-graphe **couvrant** $\mathcal{G}' = (V, E')$ **connexe**, alors \mathcal{G} est **connexe**.

Graphes non orientés et composantes connexes

Définition III.6 (Graphe non orienté connexe). Un graphe $\mathcal{G} = (V, E)$ **non orienté** est dit **connexe** si, pour toute paire de sommets u, v , il existe une chaîne qui relie u et v .

Définition III.7 (Composante connexe). Une **composante connexe** d'un graphe est un sous-graphe **induit**, **connexe** et **maximal** (c'est-à-dire qu'on ne peut pas rajouter de sommet sans rendre le sous-graphe non connexe).



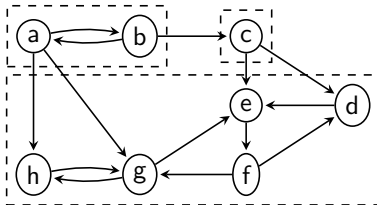
Exercice III.8. Soit un graphe non orienté $\mathcal{G} = (V, E)$. Prouvez que, si \mathcal{G} a un sous-graphe **couvrant** $\mathcal{G}' = (V, E')$ **connexe**, alors \mathcal{G} est **connexe**.

Trouver les composantes connexes se fait facilement avec DFS.

Graphe orienté et composantes **fortement** connexe

Définition III.9 (Graphe fortement connexe). Un graphe $\mathcal{G} = (V, E)$ **orienté** est dit **fortement connexe** si, pour toute paire de sommets u, v , il existe un chemin qui relie u et v .

Définition III.10 (Composante fortement connexe). Une **composante fortement connexe** d'un graphe est un sous-graphe **induit**, **fortement connexe** et **maximal** (c'est-à-dire qu'on ne peut pas rajouter de sommet sans rendre le sous-graphe non connexe).



Exercice III.11. Soit un graphe orienté $\mathcal{G} = (V, E)$. Est-il vrai que, si \mathcal{G} a un sous-graphe **couvrant** $\mathcal{G}' = (V, E')$ **fortement connexe**, alors \mathcal{G} est **fortement connexe** ?

Trouver les composantes fortement connexes – Algorithme naïf

On veut identifier les composantes fortement connexes d'un graphe.

Trouver les composantes fortement connexes – Algorithme naïf

On veut identifier les composantes fortement connexes d'un graphe.

Lemme III.12. Soient u et v deux sommets d'un graphe orienté. S'il existe un chemin de u vers v et un chemin de v vers u , alors u et v sont dans la même composante fortement connexe.

Trouver les composantes fortement connexes – Algorithme naïf

On veut identifier les composantes fortement connexes d'un graphe.

Lemme III.12. Soient u et v deux sommets d'un graphe orienté. S'il existe un chemin de u vers v et un chemin de v vers u , alors u et v sont dans la même composante fortement connexe.

On peut donc déterminer si u et v sont dans la même composante grâce à deux DFS.

Trouver les composantes fortement connexes – Algorithme naïf

On va associer à chaque sommet un nombre. À la fin, deux sommets sont dans la **même composante** si et seulement si ils ont le **même nombre**.

Trouver les composantes fortement connexes – Algorithme naïf

On va associer à chaque sommet un nombre. À la fin, deux sommets sont dans la **même composante** si et seulement si ils ont le **même nombre**.

Nécessite : $V = \{v_1, \dots, v_n\}$

```
1 : procédure TROUVERCOMPOSANTESFORTEMENTCONNEXES( $\mathcal{G} = (V, E)$ )
2 :   Composantes  $\leftarrow$  tableau de taille  $n$  initialisé avec des 0
3 :   pour chaque  $i \in \{1, \dots, n\}$  faire
4 :     Composantes[ $v_i$ ]  $\leftarrow i$ 
5 :     Atteignable $_i \leftarrow$  DFS( $\mathcal{G}, v_i$ )            $\triangleright$  L'ensemble des sommets atteignables depuis  $v_i$ 
6 :     pour chaque  $j \in \{1, \dots, n\}$  faire
7 :       pour chaque  $j \in \{i + 1, \dots, n\}$  faire
8 :         si  $v_i \in \text{Atteignable}_j \wedge v_j \in \text{Atteignable}_i$  alors
9 :           Composantes[ $v_j$ ]  $\leftarrow$  Composantes[ $v_i$ ]
10 :   retourner Composantes
```

Trouver les composantes fortement connexes – Algorithme naïf

On va associer à chaque sommet un nombre. À la fin, deux sommets sont dans la **même composante** si et seulement si ils ont le **même nombre**.

Nécessite : $V = \{v_1, \dots, v_n\}$

```
1 : procédure TROUVERCOMPOSANTESFORTEMENTCONNEXES( $\mathcal{G} = (V, E)$ )
2 :   Composantes  $\leftarrow$  tableau de taille  $n$  initialisé avec des 0
3 :   pour chaque  $i \in \{1, \dots, n\}$  faire
4 :     Composantes[ $v_i$ ]  $\leftarrow i$ 
5 :     Atteignable $_i \leftarrow$  DFS( $\mathcal{G}, v_i$ )       $\triangleright$  L'ensemble des sommets atteignables depuis  $v_i$ 
6 :     pour chaque  $j \in \{1, \dots, n\}$  faire
7 :       pour chaque  $j \in \{i + 1, \dots, n\}$  faire
8 :         si  $v_i \in \text{Atteignable}_j \wedge v_j \in \text{Atteignable}_i$  alors
9 :           Composantes[ $v_j$ ]  $\leftarrow$  Composantes[ $v_i$ ]
10 :   retourner Composantes
```

On fait $|V|$ appels à DFS, dont la complexité est $|V| + |E| \cdot \sim \mathcal{O}(|V|^2 + |V| \cdot |E|) \in \mathcal{O}(|V|^3)$.

Algorithme de Kosaraju-Sharir

```

1 : procédure KOSARAJUSHARIR( $\mathcal{G} = (V, E)$ )
2 :   Composantes  $\leftarrow$  tableau de taille  $n$ 
   initialisé avec des 0
3 :   L  $\leftarrow$  liste initialement vide
4 :    $V_{us} \leftarrow \emptyset$ 
5 :   pour chaque  $v \in V$  faire
6 :      $V_{us} \leftarrow \text{VISITE}(\mathcal{G}, v, V_{us}, L)$ 
7 :   pour chaque  $v \in L$  faire
8 :      $i \leftarrow \max \text{Composantes} + 1$ 
9 :     ASSIGNER( $\mathcal{G}, v, i, \text{Composantes}$ )
10 : retourner Composantes
  
```

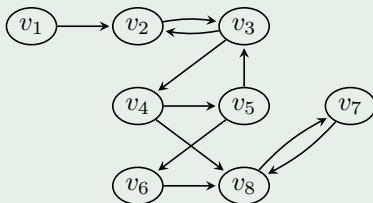
```

1 : procédure VISITE( $\mathcal{G}, v, V_{us}, L$ )
2 :   si  $v \notin V_{us}$  alors
3 :      $V_{us} \leftarrow V_{us} \cup \{v\}$ 
4 :     pour chaque successeur  $u$  de  $v$  faire
5 :        $V_{us} \leftarrow \text{VISITE}(\mathcal{G}, u, V_{us})$ 
6 :      $L.\text{PREPEND}(v)$ 
   retourner  $V_{us}$ 

7 : procédure ASSIGNER( $\mathcal{G}, v, i, \text{Composantes}$ )
8 :   si  $\text{Composantes}[v] = 0$  alors
9 :      $\text{Composantes}[v] \leftarrow i$ 
10 :   pour chaque prédecesseur  $u$  de  $v$  faire
11 :     ASSIGNER( $\mathcal{G}, u, i, \text{Composantes}$ )
  
```

Kosaraju-Sharir – Exercices

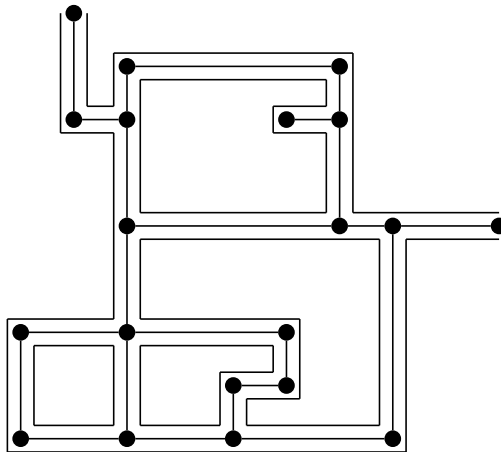
Exercice III.13. Appliquez l'algorithme de Kosaraju-Sharir sur le graphe suivant pour identifier ses composantes fortement connexes.



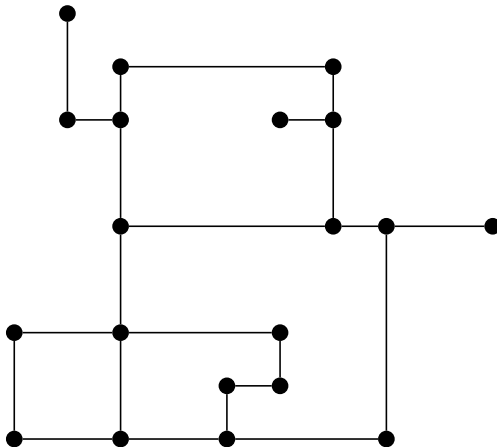
Exercice III.14. Quelle est la complexité de l'algorithme de Kosaraju-Sharir, en fonction de $|V|$ et $|E|$?

1. Parcours en profondeur d'abord
2. Composantes fortement connexes
3. Parcours en largeur d'abord et chemin minimal

Trouver un chemin avec un nombre minimal de pas

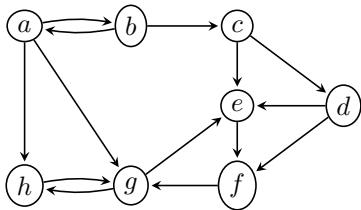


Trouver un chemin avec un nombre minimal de pas



Chemin minimal

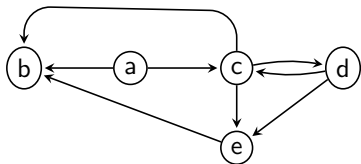
Définition III.15 (Chemin minimal). Soit un graphe $\mathcal{G} = (V, E)$. Un chemin (v_1, \dots, v_n) est dit **minimal** si n'importe quel chemin (v'_1, \dots, v'_m) avec $v_1 = v'_1$ et $v_n = v'_m$ est tel que $n \leq m$.



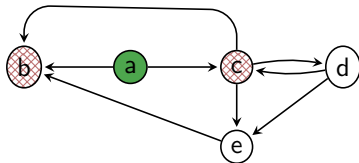
Le chemin (a, b, c, e) n'est pas minimal.
 Le chemin (a, g, e) est minimal.
 Les chemins (c, e, f) et (c, d, f) sont tous deux minimaux.

Exercice III.16. Est-ce que DFS garantit de trouver un chemin minimal entre deux sommets ?

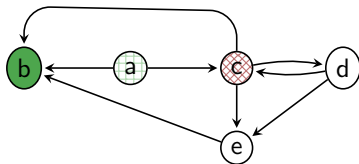
Parcours en largeur d'abord – Ordre d'exploration



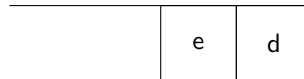
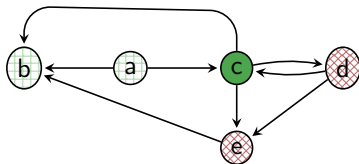
Parcours en largeur d'abord – Ordre d'exploration



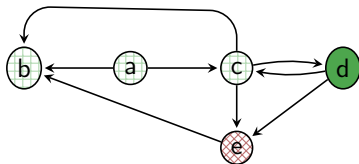
Parcours en largeur d'abord – Ordre d'exploration



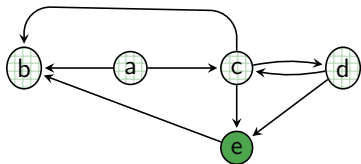
Parcours en largeur d'abord – Ordre d'exploration



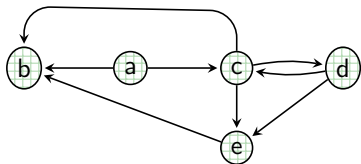
Parcours en largeur d'abord – Ordre d'exploration



Parcours en largeur d'abord – Ordre d'exploration



Parcours en largeur d'abord – Ordre d'exploration



Tous les sommets sont atteignables depuis a.

File

Définition III.17 (File). Une **file** est une structure de données **premier arrivé, premier servi**³.

Typiquement, une file est implémentée comme une liste avec les opérations :

- ▶ **enfiler** pour insérer un élément à la fin ;
- ▶ **dépiler** pour retirer et retourner l'élément au début.

Pensez à une file d'attente.

3. En anglais : FIFO pour first in, first out

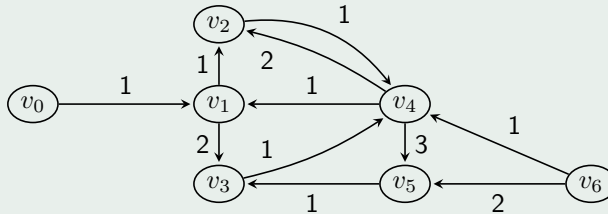
Parcours en largeur d'abord

```
1 : procédure PARCOURS LARGEUR( $\mathcal{G}, v$ )
2 :   prévisite( $v$ )
3 :    $V_{us} \leftarrow \{v\}$ 
4 :   File  $\leftarrow$  nouvelle file
5 :   File.ENFILER( $v$ )
6 :   tant que File n'est pas vide faire
7 :      $u \leftarrow$  File.DÉFILER()
8 :     visite( $u$ )
9 :     pour chaque successeur  $s$  de  $u$  dans  $\mathcal{G}$  faire
10 :       si  $s \notin V_{us}$  alors
11 :         prévisite( $v$ )
12 :          $V_{us} \leftarrow V_{us} \cup \{s\}$ 
13 :         File.ENFILER( $s$ )
14 :       sinon
15 :         revisite( $v$ )
16 :     postvisite( $v$ )
17 :   retourner  $V_{us}$ 
```

En anglais, le **parcours en largeur d'abord** est appelé **breadth-first search**, abrégé en **BFS**.

Exercices

Exercice III.18. Appliquez BFS sur le graphe suivant à partir de v_0 . Le chiffre à côté de chaque arc donne l'ordre d'itérations sur les successeurs. Par exemple, à partir de v_1 , v_2 est vu avant v_3 .



Exercice III.19. Quelle est la complexité du parcours en largeur d'abord ?

Exercice III.20. Modifiez le pseudo-code de BFS pour retourner le chemin minimal entre deux sommets u et v .