Introduction aux graphes

Parcours de graphes

Mathilde Vernet
mathilde.vernet@univ-avignon.fr

Licence Informatique, CERI
Licence Mathématiques, Institut AgES
Avignon Université

Automne 2025

Plan

Objectif du chapitre

- Faire les premiers pas sur l'algorithmique dans les graphes
- Savoir appliquer des algorithmes de parcours de graphes
- Savoir utiliser des parcours de graphes à bon escient

Qu'est-ce qu'un parcours de graphe ?

- Explorer tous les sommets du graphe

Questions

- Quelle est l'utilité de parcourir un graphe ?
- Dans quel ordre est-ce qu'on visite les sommets ?

Quelle est l'utilité de parcourir un graphe ?

- Répondre à certaines questions sur le graphe (notamment sur sa structure)
- Résoudre certains problèmes de graphes

Exemples de questions

- Quelles sont les composantes connexes d'un graphe ?
- Quelles sont les composantes fortement connexes d'un graphe orienté ?
- Un graphe est-il acyclique ?
- Un graphe est-il biparti ?
- ...

Exemples de problèmes

- Plus courts chemins
- Flots
- ...

Dans quel ordre est-ce qu'on visite les sommets ?

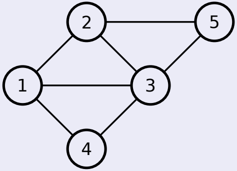
- En choisissant les sommets au hasard
  - NON : inefficace, n'apporte pas d'informations
- En visitant les sommets de proche en proche
  - OUI : à partir d'un sommet initial puis en visitant ses voisins, etc

De proche en proche, c'est-à-dire ?

Marquer v // *v un sommet quelconque*
Tant que $\exists$ une arête (i, j) avec i marqué et j non marqué
    Marquer j

Idée

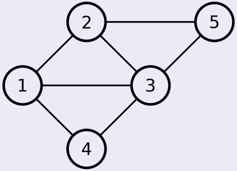
On parcourt les sommets de proche en proche :



Idée

On parcourt les sommets de proche en proche :

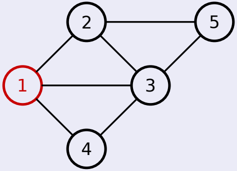
- On part d'un sommet s



Idée

On parcourt les sommets de proche en proche :

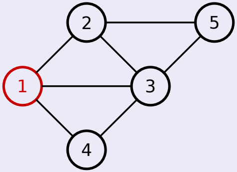
- On part d'un sommet s
- On le marque



Idée

On parcourt les sommets de proche en proche :

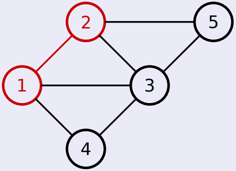
- On part d'un sommet s
- On le marque
- On passe à un voisin



Idée

On parcourt les sommets de proche en proche :

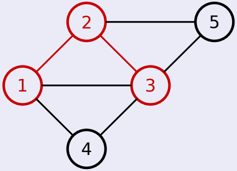
- On part d'un sommet s
- On le marque
- On passe à un voisin
- On le marque



Idée

On parcourt les sommets de proche en proche :

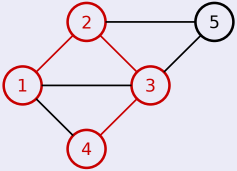
- On part d'un sommet s
- On le marque
- On passe à un voisin
- On le marque
- ...



Idée

On parcourt les sommets de proche en proche :

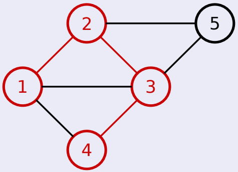
- On part d'un sommet s
- On le marque
- On passe à un voisin
- On le marque
- ...



Idée

On parcourt les sommets de proche en proche :

- On part d'un sommet s
- On le marque
- On passe à un voisin
- On le marque
- ...

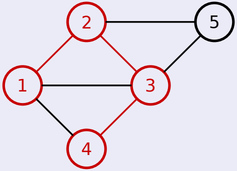


Comment faire lorsqu'il n'y a plus de voisin non marqué ?

Idée

On parcourt les sommets de proche en proche :

- On part d'un sommet s
- On le marque
- On passe à un voisin
- On le marque
- ...



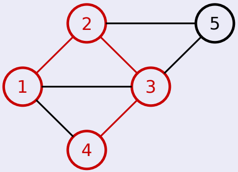
Comment faire lorsqu'il n'y a plus de voisin non marqué ?

- On ne passe pas à un voisin déjà marqué

Idée

On parcourt les sommets de proche en proche :

- On part d'un sommet s
- On le marque
- On passe à un voisin
- On le marque
- ...



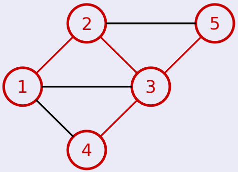
Comment faire lorsqu'il n'y a plus de voisin non marqué ?

- On ne passe pas à un voisin déjà marqué
- On doit se souvenir des sommets que l'on a visité

Idée

On parcourt les sommets de proche en proche :

- On part d'un sommet s
- On le marque
- On passe à un voisin
- On le marque
- ...



Comment faire lorsqu'il n'y a plus de voisin non marqué ?

- On ne passe pas à un voisin déjà marqué
- On doit se souvenir des sommets que l'on a visité

Que connaît-on du graphe ?

- Ses sommets
- Ses arêtes

Donc à partir d'un sommet, on connaît ses voisins

De quoi a-t-on besoin ?

- Marquer les sommets visités
- Se souvenir des voisins que l'on doit visiter
  - Comment ? Utilisation d'une structure de données pour stocker cette information

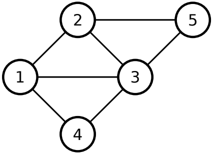
Grandes lignes de l'algorithme de parcours

Choisir un sommet v quelconque
Marquer v
Se souvenir qu'on l'a visité afin de visiter ses voisins plus tard
Continuer tant qu'il reste des voisins à visiter

Algorithme générique de parcours de graphe

Procedure `explore`(G,s) *//Pour parcourir le graphe G à partir du sommet s*
  v.visited $\leftarrow$ false $\forall v \in V$ *//Tous les sommets commencent non marqués*
  s.visited $\leftarrow$ true *//s est marqué*
  open $\leftarrow \{s\}$ *//Pour se souvenir de visiter les voisins de s*
  TantQue open $\neq \emptyset$ **Faire**
    //Tant qu'il reste des sommets dont on n'a pas visité tous les voisins
    $v \leftarrow$ un élément de open *//En choisir un*
    //Puis choisir un de ses voisins non visités
    $u \leftarrow$ un voisin de v avec u.visited $=$ false
    Si u existe **Alors**
      //S'il existe un voisin de v non visité
      u.visited $\leftarrow$ true *//On le marque*
      open $\leftarrow$ open $\cup \{u\}$ *//Pour se souvenir de visiter ses voisins*
    Sinon
      //Tous les voisins de v sont visités
      //On n'a plus besoin de v dans la structure de données
      open $\leftarrow$ open $\setminus \{v\}$
    FinSi
  FinTantQue
FinProcedure

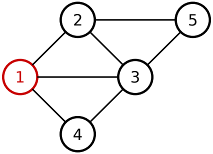
Exemple



Étapes de `explore(`G`,1)`

- Un sommet visité apparaît en rouge
- Tous les sommets sont non visités

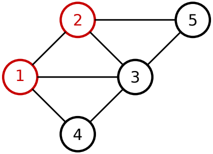
Exemple



Étapes de `explore(`G`,1)`

- Un sommet visité apparaît en rouge
- Tous les sommets sont non visités
- On visite 1
- open $\leftarrow \{1\}$

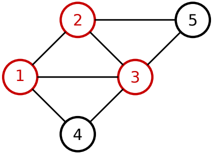
Exemple



Étapes de `explore(`G`,1)`

- Un sommet visité apparaît en rouge
- Tous les sommets sont non visités
- On visite 1
- open $\leftarrow \{1\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (unique élément de open)
- $u \leftarrow 2$ (un voisin non visité de 1)
- On visite 2
- open $\leftarrow \{1, 2\}$

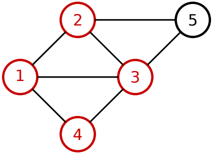
Exemple



Étapes de `explore(`G`,1)`

- Un sommet visité apparaît en rouge
- Tous les sommets sont non visités
- On visite 1
- open $\leftarrow \{1\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (unique élément de open)
- $u \leftarrow 2$ (un voisin non visité de 1)
- On visite 2
- open $\leftarrow \{1, 2\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (un élément de open)
- $u \leftarrow 3$ (un voisin non visité de 1)
- On visite 3
- open $\leftarrow \{1, 2, 3\}$

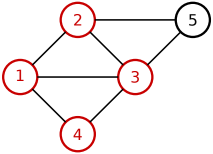
Exemple



Étapes de `explore(`G`,1)`

- Un sommet visité apparaît en rouge
- Tous les sommets sont non visités
- On visite 1
- open $\leftarrow \{1\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (unique élément de open)
- $u \leftarrow 2$ (un voisin non visité de 1)
- On visite 2
- open $\leftarrow \{1, 2\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (un élément de open)
- $u \leftarrow 3$ (un voisin non visité de 1)
- On visite 3
- open $\leftarrow \{1, 2, 3\}$
- open $\neq \emptyset$
- $v \leftarrow 3$ (un élément de open)
- $u \leftarrow 4$ (un voisin non visité de 3)
- On visite 4
- open $\leftarrow \{1, 2, 3, 4\}$

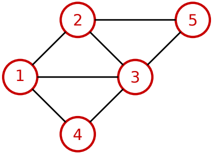
Exemple



Étapes de `explore(`G`,1)`

- Un sommet visité apparaît en rouge
- Tous les sommets sont non visités
- On visite 1
- open $\leftarrow \{1\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (unique élément de open)
- $u \leftarrow 2$ (un voisin non visité de 1)
- On visite 2
- open $\leftarrow \{1, 2\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (un élément de open)
- $u \leftarrow 3$ (un voisin non visité de 1)
- On visite 3
- open $\leftarrow \{1, 2, 3\}$
- open $\neq \emptyset$
- $v \leftarrow 3$ (un élément de open)
- $u \leftarrow 4$ (un voisin non visité de 3)
- On visite 4
- open $\leftarrow \{1, 2, 3, 4\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (un élément de open)
- 1 n'a pas de voisin non visité
- open $\leftarrow \{2, 3, 4\}$

Exemple



Étapes de `explore(`G`,1)`

- Un sommet visité apparaît en rouge
- Tous les sommets sont non visités
- On visite 1
- open $\leftarrow \{1\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (unique élément de open)
- $u \leftarrow 2$ (un voisin non visité de 1)
- On visite 2
- open $\leftarrow \{1, 2\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (un élément de open)
- $u \leftarrow 3$ (un voisin non visité de 1)
- On visite 3
- open $\leftarrow \{1, 2, 3\}$
- open $\neq \emptyset$
- $v \leftarrow 3$ (un élément de open)
- $u \leftarrow 4$ (un voisin non visité de 3)
- On visite 4
- open $\leftarrow \{1, 2, 3, 4\}$
- open $\neq \emptyset$
- $v \leftarrow 1$ (un élément de open)
- 1 n'a pas de voisin non visité
- open $\leftarrow \{2, 3, 4\}$
- open $\neq \emptyset$
- $v \leftarrow 2$ (un élément de open)
- $u \leftarrow 5$ (un voisin non visité de 2)
- On visite 5
- open $\leftarrow \{2, 3, 4, 5\}$
- open $\neq \emptyset$
- $v \leftarrow 5$ (un élément de open)
- 5 n'a pas de voisin non visité
- open $\leftarrow \{2, 3, 4\}$
- open $\neq \emptyset$
- $v \leftarrow 3$ (un élément de open)
- 3 n'a pas de voisin non visité
- open $\leftarrow \{2, 4\}$
- open $\neq \emptyset$
- $v \leftarrow 4$ (un élément de open)
- 4 n'a pas de voisin non visité
- open $\leftarrow \{2\}$
- $v \leftarrow 2$ (un élément de open)
- 2 n'a pas de voisin non visité
- open $= \emptyset$
- Fin de la procédure

Les éléments de l'algorithme

- `explore`(G, s) parcourt les sommets de G à partir du sommet s
- v.visited marque le sommet v comme ayant été visité
- `open` est la structure de données permettant de se souvenir des sommets dont on doit visiter les voisins
- `open`$\leftarrow$`open`$\cup\{u\}$ ajoute le sommet u à la structure `open`
- `open`$\leftarrow$`open`$\setminus\{v\}$ supprime le sommet v de la structure `open`

Remarques sur la procédure `explore`

- Procédure applicable aux graphes orientés : on s'intéresse aux <u>voisins sortants</u> du sommets considéré dans la boucle
- À la fin de la procédure `explore(`G`,`s`)`, on a v.visited $=$ true si et seulement si v est atteignable à partir de s
- Si l'on veut parcourir tous les sommets du graphe, il faut recommencer la procédure à partir d'un sommet non visité tant qu'il en reste
- Selon l'ordre dans lequel on choisit les éléments à traiter de `open`, les sommets ne sont pas parcourus dans le même ordre

La structure de liste

- Un ensemble d'éléments dans un certain ordre
- On peut ajouter des éléments à la liste
- On peut retirer des éléments de la liste

La structure de file

- *Comme une file d'attente chez un commerçant*
- Structure aussi appelée FIFO (*first in first out* en anglais)
- C'est une liste particulière :
  - Un ensemble d'éléments dans un certain ordre
  - Les éléments sont ajoutés à la fin
  - Les éléments sont retirés au début

Opérations sur une file f

- f.`empty()` : la file est-elle vide ?
- f.`add(`v`)` : ajoute le sommet v à la fin de la file
- f.`peek()` : retourne le premier élément de la file
- f.`remove()` : retire et retourne le premier élément de la file

Exemple

Opération sur la file	**État de la file après l'opération**
initialisation	$f = []$
f.`add(`A`)`	$f = [A]$
f.`add(`B`)`	$f = [A, B]$
f.`remove()`	$f = [B]$
f.`add(`C`)`	$f = [B, C]$
f.`add(`D`)`	$f = [B, C, D]$
f.`remove()`	$f = [C, D]$
f.`remove()`	$f = [D]$
f.`remove()`	$f = []$
f.`add(`E`)`	$f = [E]$

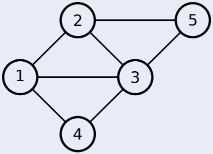
Procédure `explore` pour le BFS

Procedure `explore`(G,s)
  v.visited $\leftarrow$ false $\forall v \in V$
  s.visited $\leftarrow$ true
  open.add(s)
  TantQue $\neg$open.empty() **Faire**
    $v \leftarrow$ open.peek()
    $u \leftarrow$ un voisin de v avec u.visited $=$ false
    Si u existe **Alors**
      u.visited $\leftarrow$ true
      open.add(u)
    Sinon
      open.remove()
    FinSi
  FinTantQue
FinProcedure

Remarque

- C'est la procédure `explore` générique où `open` est géré comme une file

Exemple : `explore(`G`,5)`

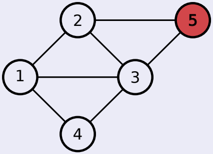


Remarque

- On visite d'abord un sommet (5)
- Puis on visite tous les voisins (2 et 3)
- Puis on visite les voisins des voisins (1, puis 4)

C'est ce qu'on appelle un parcours en largeur, ou BFS (*breadth-first search*)

Exemple : explore(G,5)



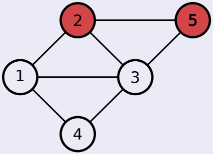
- 5.*visited* $\leftarrow$ true
- open.*add*(5) (open=[5])

Remarque

- On visite d'abord un sommet (5)
- Puis on visite tous les voisins (2 et 3)
- Puis on visite les voisins des voisins (1, puis 4)

C'est ce qu'on appelle un parcours en largeur, ou BFS (*breadth-first search*)

Exemple : explore(G,5)



- open.*add*(2) (open=[5,2])

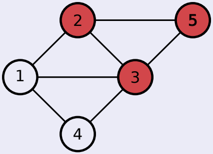
- 5.*visited* ← true
- open.*add*(5) (open=[5])
- 2.*visited* ← true

Remarque

- On visite d'abord un sommet (5)
- Puis on visite tous les voisins (2 et 3)
- Puis on visite les voisins des voisins (1, puis 4)

C'est ce qu'on appelle un parcours en largeur, ou BFS (*breadth-first search*)

Exemple : explore(G,5)



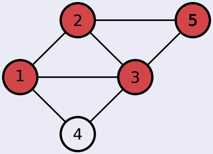
- 5.*visited* ← true
- open.*add*(5) (open=[5])
- 2.*visited* ← true
- open.*add*(2) (open=[5,2])
- 3.*visited* ← true
- open.*add*(3) (open=[5,2,3])
- open.*remove*() (open=[2,3])

Remarque

- On visite d'abord un sommet (5)
- Puis on visite tous les voisins (2 et 3)
- Puis on visite les voisins des voisins (1, puis 4)

C'est ce qu'on appelle un parcours en largeur, ou BFS (*breadth-first search*)

Exemple : explore(G,5)



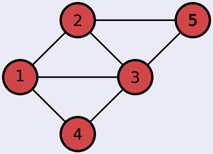
- 5.*visited* ← true
- open.*add*(5) (open=[5])
- 2.*visited* ← true
- open.*add*(2) (open=[5,2])
- 3.*visited* ← true
- open.*add*(3) (open=[5,2,3])
- open.*remove*() (open=[2,3])
- 1.*visited* ← true
- open.*add*(1) (open=[2,3,1])
- open.*remove*() (open=[3,1])

Remarque

- On visite d'abord un sommet (5)
- Puis on visite tous les voisins (2 et 3)
- Puis on visite les voisins des voisins (1, puis 4)

C'est ce qu'on appelle un parcours en largeur, ou BFS (*breadth-first search*)

Exemple : explore(G,5)



- 5.*visited* ← true
- open.*add*(5) (open=[5])
- 2.*visited* ← true
- open.*add*(2) (open=[5,2])
- 3.*visited* ← true
- open.*add*(3) (open=[5,2,3])
- open.*remove*() (open=[2,3])
- 1.*visited* ← true
- open.*add*(1) (open=[2,3,1])
- open.*remove*() (open=[3,1])
- 4.*visited* ← true
- open.*add*(4) (open=[3,1,4])
- open.*remove*() (open=[1,4])
- open.*remove*() (open=[4])
- open.*remove*() (open=[])

Remarque

- On visite d'abord un sommet (5)
- Puis on visite tous les voisins (2 et 3)
- Puis on visite les voisins des voisins (1, puis 4)

C'est ce qu'on appelle un parcours en largeur, ou BFS (*breadth-first search*)

Remarques

- La procédure précédente ne permet de visiter que les sommets accessibles depuis s, c'est-à-dire la composante connexe contenant s
- Pour visiter tout le graphe, il faut recommencer la procédure dans chaque composante connexe du graphe

Parcours d'un graphe non orienté avec BFS

Procedure `BFS`(G)
  v.visited $\leftarrow$ false $\forall v \in V$
  Pour $(v) \in V$ **Faire**
    Si $\neg v$.visited **Alors**
      `exploreBFS`(G,v)
    FinSi
  FinPour
FinProcedure

Procedure `exploreBFS`(G,s)
  s.visited $\leftarrow$ true
  open.`add`(s)
  TantQue $\neg$open.`empty()` **Faire**
    $v \leftarrow$ open.`peek()`
    $u \leftarrow$ un voisin de v avec u.visited = false
    Si u existe **Alors**
      u.visited $\leftarrow$ true
      open.`add`(u)
    Sinon
      open.`remove()`
    FinSi
  FinTantQue
FinProcedure

Adaptation aux graphes orientés

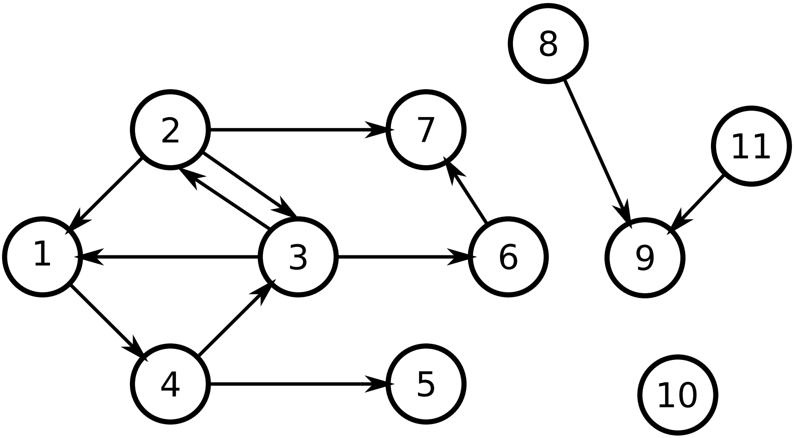
- Parcourir le graphe dans le sens des arcs
- Considérer que les arcs sortants d'un sommet (et donc que ses voisins sortants)

Parcours d'un graphe orienté avec BFS

Procedure `BFS`(G)
  v.visited $\leftarrow$ false $\forall v \in V$
  Pour $(v) \in V$ **Faire**
    Si $\neg v$.visited **Alors**
      `exploreBFS`(G,v)
    FinSi
  FinPour
FinProcedure

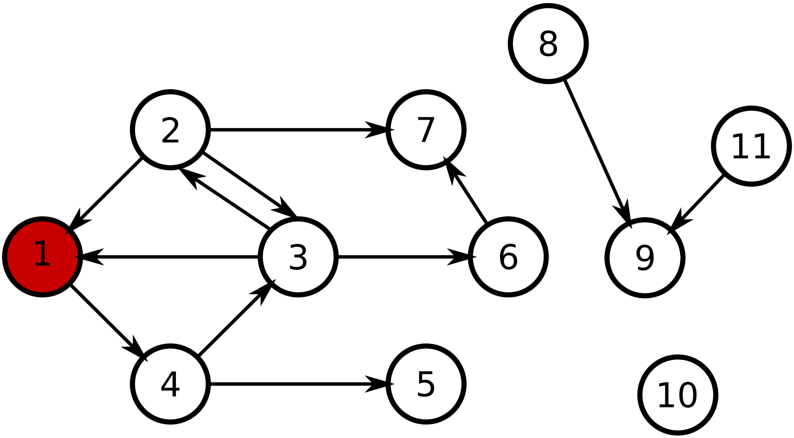
Procedure `exploreBFS`(G,s)
  s.visited $\leftarrow$ true
  open.add(s)
  TantQue $\neg$open.empty() **Faire**
    $v \leftarrow$ open.peek()
    $u \leftarrow$ sommet tel que $(v, u) \in A$
        et u.visited $=$ false
    Si u existe **Alors**
      u.visited $\leftarrow$ true
      open.add(u)
    Sinon
      open.remove()
    FinSi
  FinTantQue
FinProcedure

Exemple



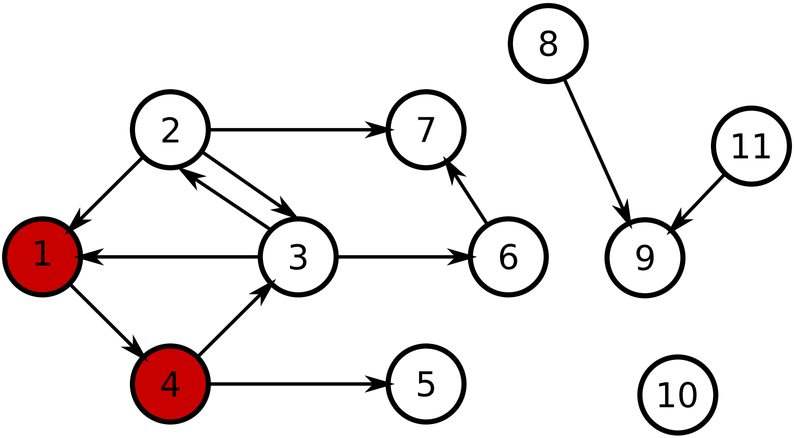
File open =

Exemple



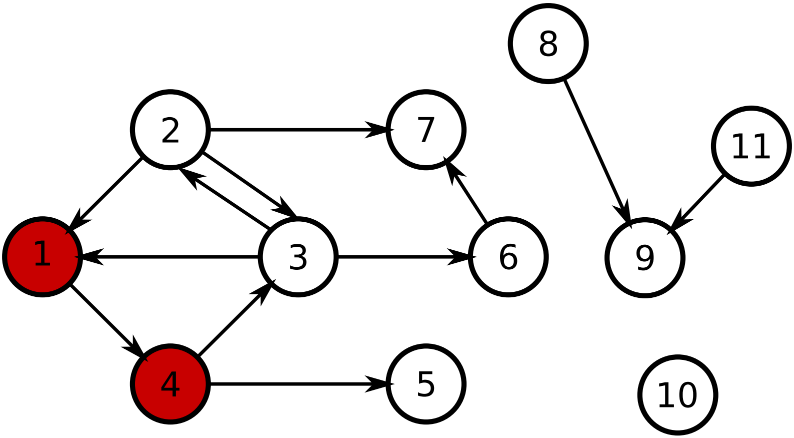
File open = 1

Exemple



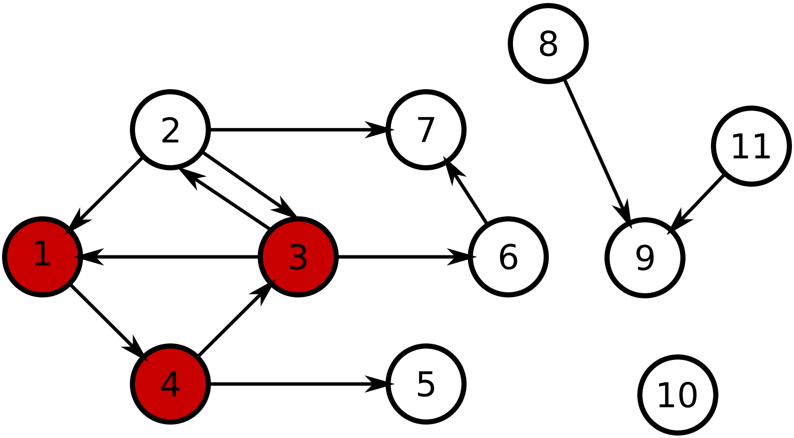
File open = 1 , 4

Exemple



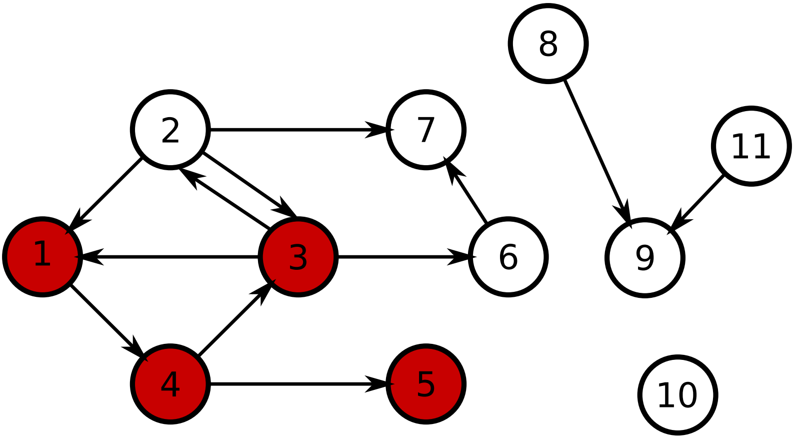
File open = $\cancel{1}$, 4

Exemple



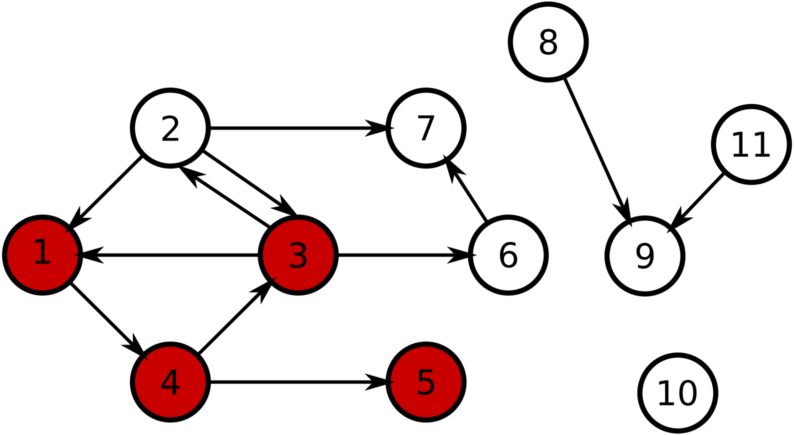
File open = $\cancel{1}$, 4 , 3

Exemple



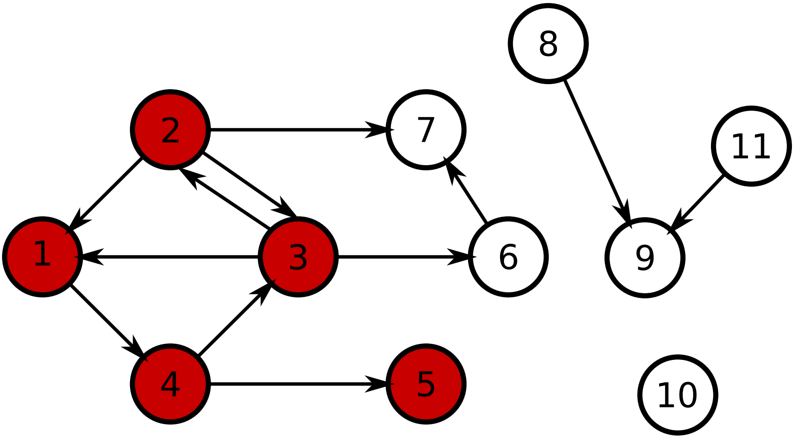
File open = $\cancel{1}$, 4 , 3 , 5

Exemple



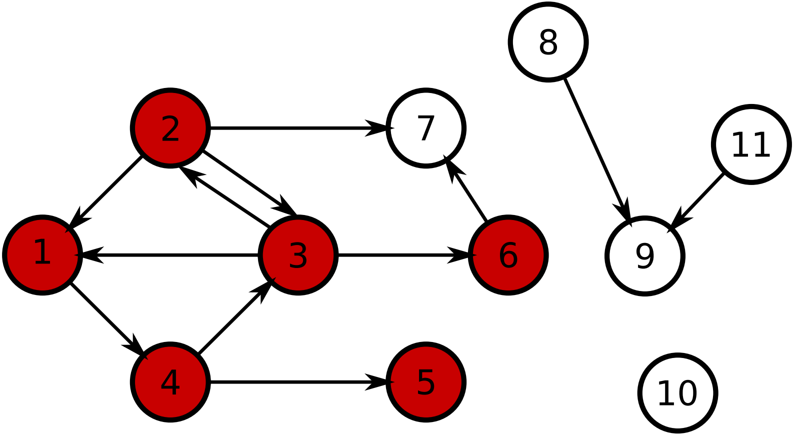
File open = $\cancel{1}$, 4 , 3 , 5

Exemple



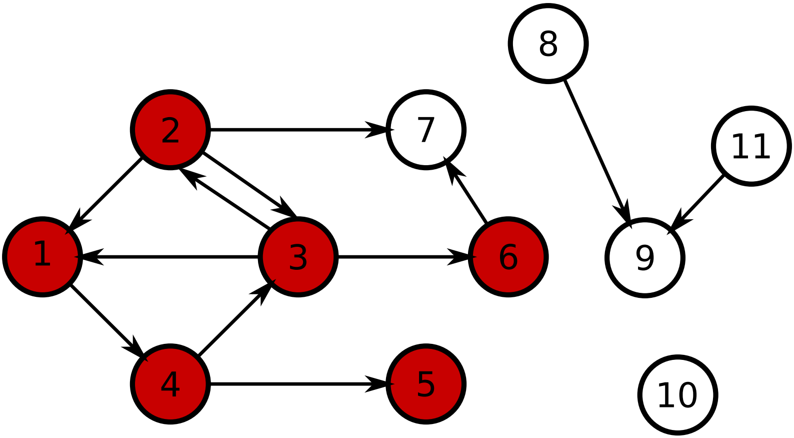
File open = ~~1~~ , ~~4~~ , 3 , 5 , 2

Exemple



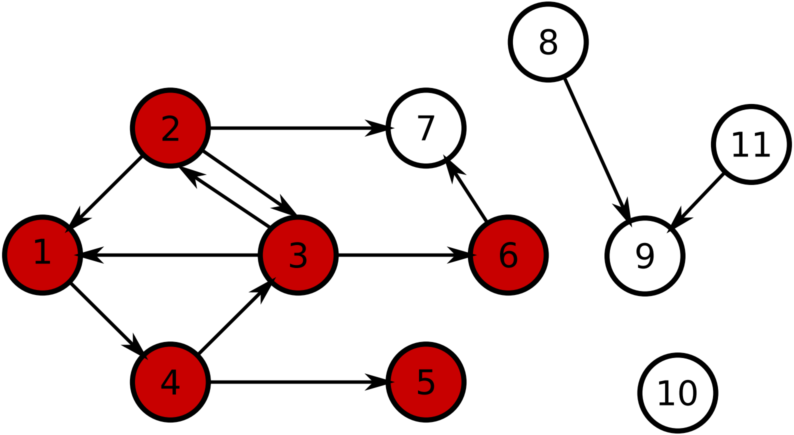
File open $= \cancel{1}$, $\cancel{4}$, 3 , 5 , 2 , 6

Exemple



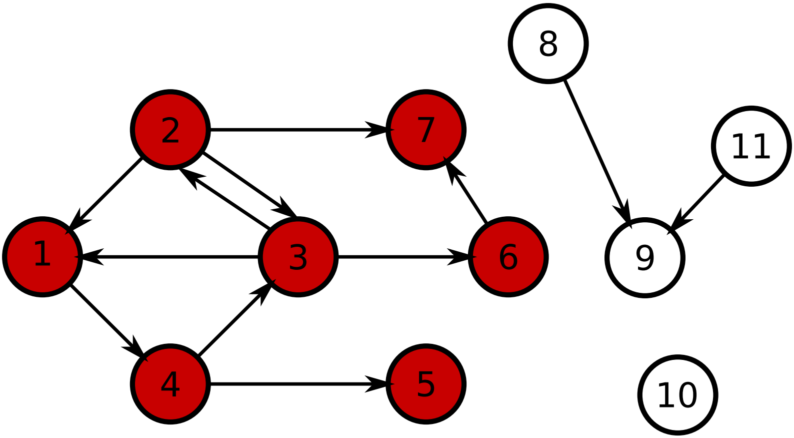
File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, 5 , 2 , 6

Exemple



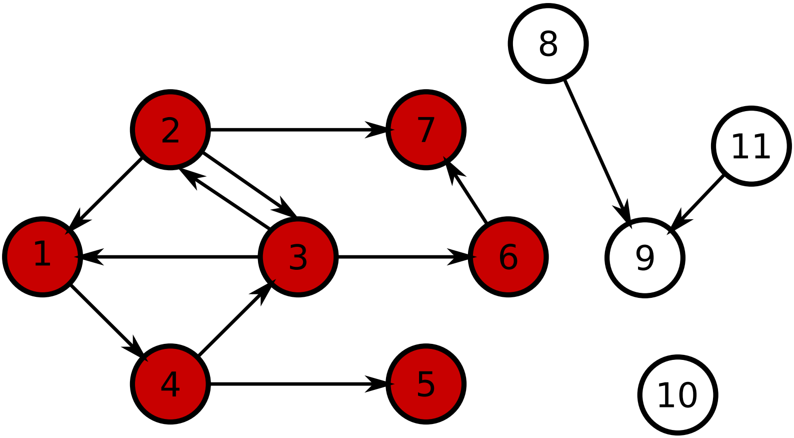
File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, $\cancel{5}$, 2 , 6

Exemple



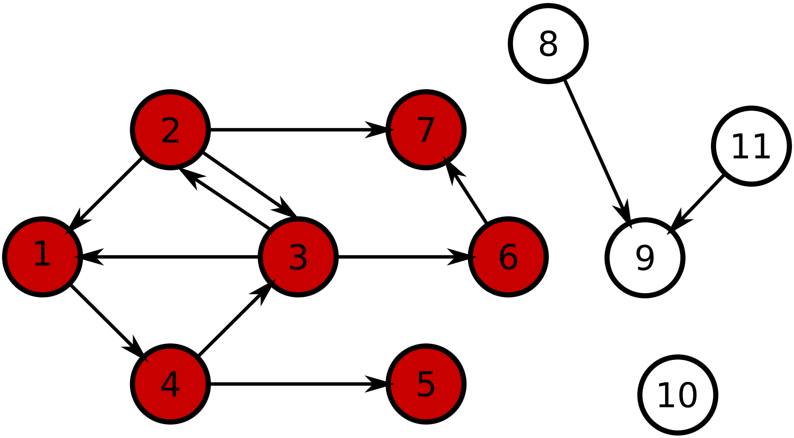
File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, $\cancel{5}$, $\cancel{2}$, 6 , 7

Exemple



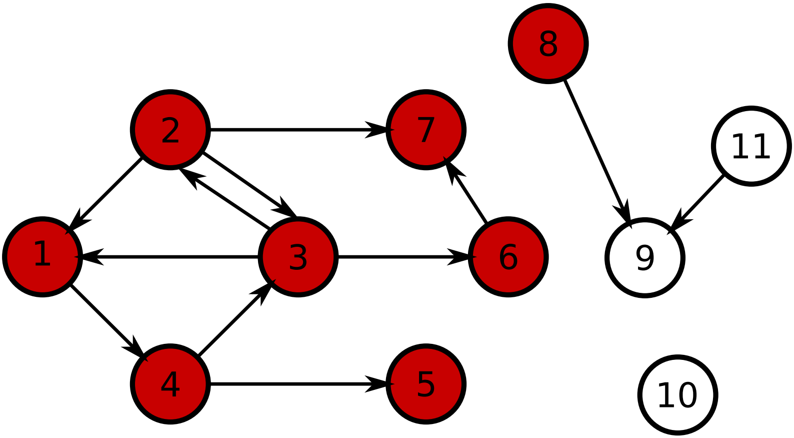
File open = ~~1~~ , ~~4~~ , ~~3~~ , ~~5~~ , ~~2~~ , ~~6~~ , 7

Exemple



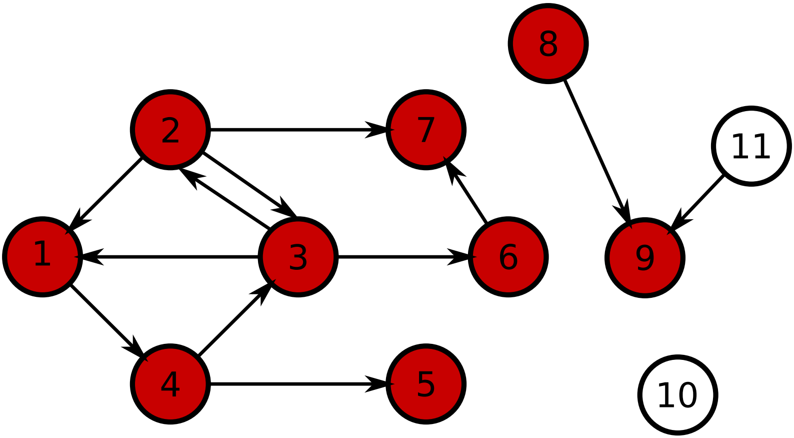
File open = $\cancel{1}$, 4 , $\cancel{3}$, $\cancel{5}$, $\cancel{2}$, $\cancel{6}$, $\cancel{7}$

Exemple



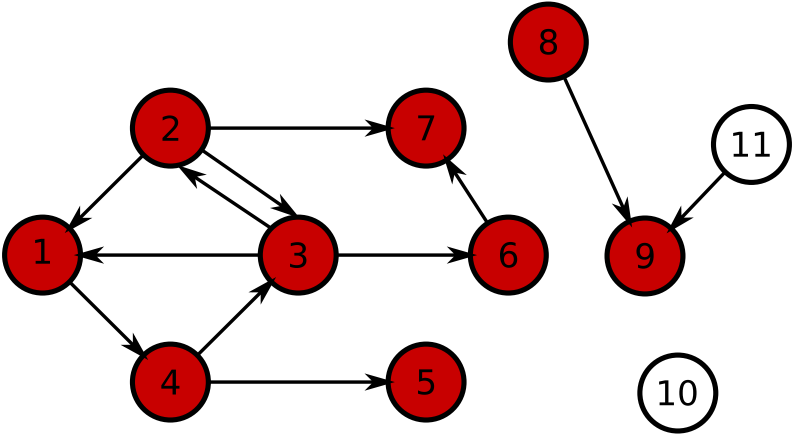
File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, $\cancel{5}$, $\cancel{2}$, $\cancel{6}$, $\cancel{7}$, 8

Exemple



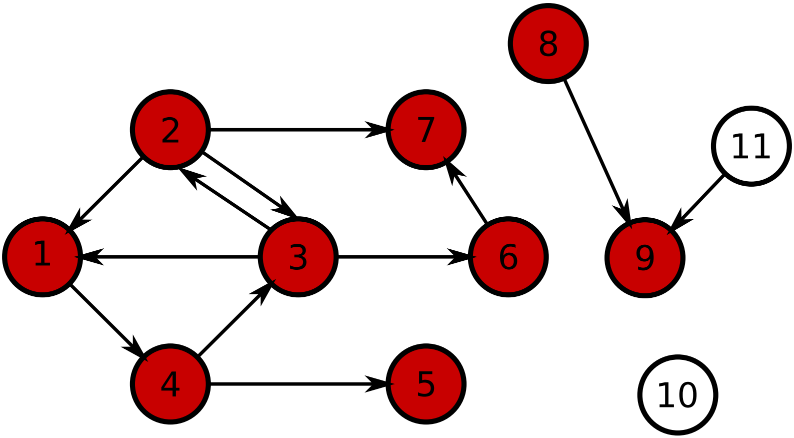
File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, $\cancel{5}$, $\cancel{2}$, $\cancel{6}$, $\cancel{7}$, 8 , 9

Exemple



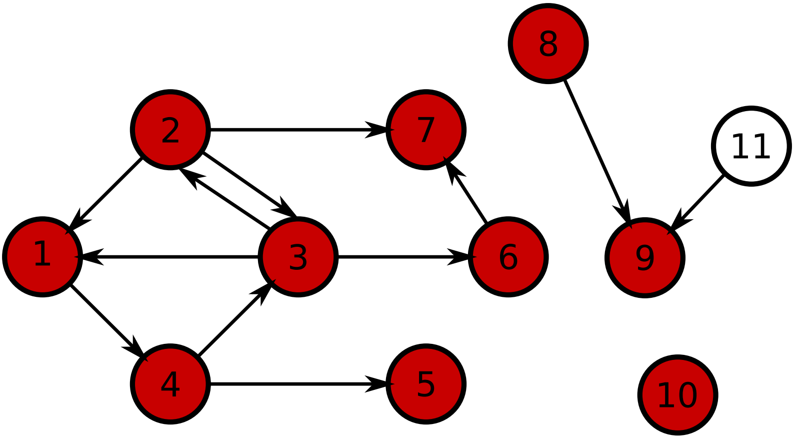
File open $= \cancel{1}\ ,\ \cancel{4}\ ,\ \cancel{3}\ ,\ \cancel{5}\ ,\ \cancel{2}\ ,\ \cancel{6}\ ,\ \cancel{7}\ ,\ \cancel{8}\ ,\ 9$

Exemple



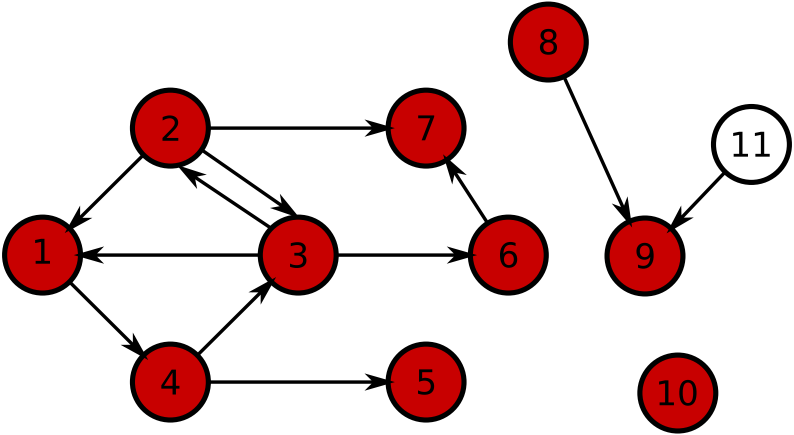
File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, $\cancel{5}$, $\cancel{2}$, $\cancel{6}$, $\cancel{7}$, $\cancel{8}$, $\cancel{9}$

Exemple



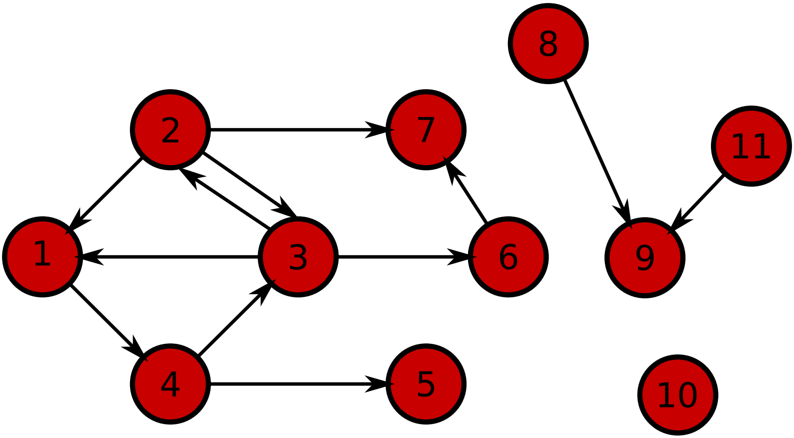
File open = ~~1~~ , 4 , ~~3~~ , ~~5~~ , ~~2~~ , ~~6~~ , ~~7~~ , ~~8~~ , ~~9~~ , 10

Exemple



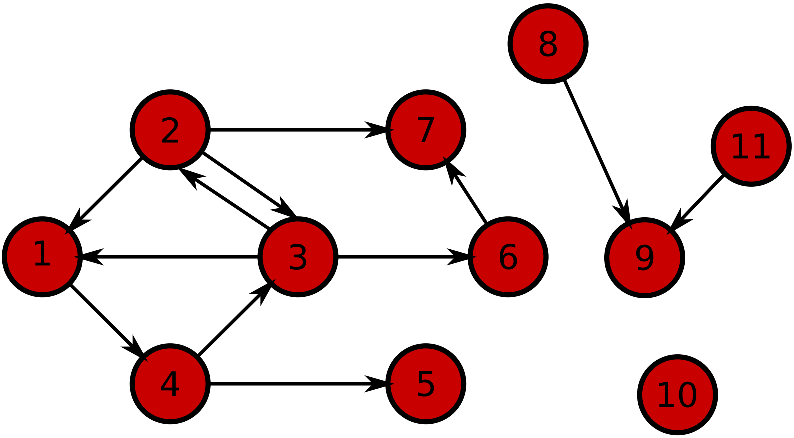
File open $= \cancel{1}\ ,\ \cancel{4}\ ,\ \cancel{3}\ ,\ \cancel{5}\ ,\ \cancel{2}\ ,\ \cancel{6}\ ,\ \cancel{7}\ ,\ \cancel{8}\ ,\ \cancel{9}\ ,\ \cancel{10}$

Exemple



File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, $\cancel{5}$, $\cancel{2}$, $\cancel{6}$, $\cancel{7}$, $\cancel{8}$, $\cancel{9}$, $\cancel{10}$, 11

Exemple



File open = $\cancel{1}$, $\cancel{4}$, $\cancel{3}$, $\cancel{5}$, $\cancel{2}$, $\cancel{6}$, $\cancel{7}$, $\cancel{8}$, $\cancel{9}$, $\cancel{10}$, $\cancel{11}$

Où en est-on ?

- On sait comment parcourir un graphe en largeur
- Et maintenant, qu'est-ce qu'on en fait ?
  - On applique des *opérations* pendant le parcours

Comment utiliser un parcours de graphe ?

- Quelles opérations appliquer ?
  - On peut marquer les sommets visités
  - On peut marquer les arêtes visitées (ou arcs visités)
  - On peut ajouter des *étiquettes* sur les sommets visités
  - On peut ajouter des *étiquettes* sur les arêtes visitées (ou arcs visités)
- Quand appliquer ces opérations ?
  - Au moment où on connaît l'information que l'on veut noter

Que faire ?

- Identifier les endroits où l'information pertinente est connue
- C'est là que l'on va effectuer les traitement nécessaires
- Si besoin, on peut regarder ensuite chaque sommet et chaque arête (ou arc)

Dans la procédure explore

Procedure `exploreBFS`(G,s)
  s.visited ← true
  //Ici on arrive sur s pour le visiter
  `open.add`(s)
  TantQue ¬`open.empty()` **Faire**
    v ← `open.peek()`
    u ← un voisin de v avec u.visited = false
    Si u existe **Alors**
      u.visited ← true
      //Ici on arrive sur u pour le visiter
      //On sait qu'on emprunte l'arête (v, u)
      `open.add`(u)
    Sinon
      //Ici on sait qu'on en a terminé avec le sommet au début de la file
      //Ce sommet ainsi que tous ses voisins sont traités
      `open.remove()`
    FinSi
  FinTantQue
FinProcedure

Que faire ?

- On a identifié les endroits où effectuer des traitements de sommets et arêtes (ou arcs) visités
- On doit ajouter des procédures de traitement

Procédures génériques de traitement

- `previsit(`v`)` : opération sur le sommet v lorsqu'on arrive sur v pour le visiter
- `postvisit(`v`)` : opération sur le sommet v quand on en a terminé avec v (quand on ne reviendra plus dessus)
- `vistEdge(`i, j`)` : opération sur l'arête (ou arc) (i, j) lorsqu'on la visite

Exemple de traitement

- `u.parent` $\leftarrow v$: on indique qu'on est arrivé sur le sommet u depuis le sommet v
- `(v,u).mark` $\leftarrow$ *true* : on indique que l'arête (v, u) est marquée

Parcours d'un graphe avec BFS et traitement des sommets et arêtes (ou arcs)

Procedure `BFS`(G)
  v.visited $\leftarrow$ false $\forall v \in V$
  Pour $(v) \in V$ **Faire**
    Si $\neg v$.visited **Alors**
      `exploreBFS`(G,v)
    FinSi
  FinPour
FinProcedure

Procedure `exploreBFS`(G,s)
  s.visited $\leftarrow$ true
  `previsit`(s)
  `open.add`(s)
  TantQue $\neg$`open.empty()` **Faire**
    $v \leftarrow$ `open.peek()`
    $u \leftarrow$ un voisin de v avec u.visited $=$ false
    Si u existe **Alors**
      u.visited $\leftarrow$ true
      `previsit`(u)
      `visitEdge`(v, u)
      `open.add`(u)
    Sinon
      $w \leftarrow$`open.remove()` *// w est v ici*
      `postvisit`(w)
    FinSi
  FinTantQue
FinProcedure

Comment utiliser un parcours pour identifier les composantes connexes d'un graphe ?

- La procédure `explore(`G`,`s`)` marque tous les sommets accessibles depuis s dans G
- Tous les sommets visités lors du même appel à `explore` sont dans la même composante connexe
- On change de composante connexe à chaque fois que l'on doit faire un nouvel appel à `explore`

Idée

Lors du parcours de graphe :

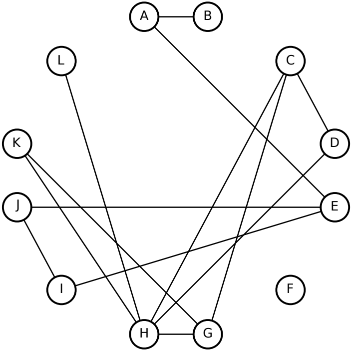
- On attribue à chaque composante connexe un numéro
- On donne à chaque sommet le numéro de la composante connexe à laquelle il appartient
- On peut utiliser les procédures de traitement

BFS pour trouver les composantes connexes d'un graphe non-orienté

Procedure `BFScc`(G)
  v.visited $\leftarrow$ false $\forall v \in V$
  cc$\leftarrow$0
  Pour $(v) \in V$ **Faire**
    Si $\neg v$.visited **Alors**
      cc$\leftarrow$cc+1
      `exploreBFScc`(G,v)
    FinSi
  FinPour
FinProcedure

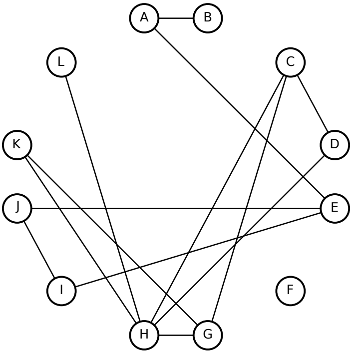
Procedure `previsit`(v)
  v.cc $\leftarrow$ cc
FinProcedure

Procedure `exploreBFScc`(G,s)
  s.visited $\leftarrow$ true
  `previsit`(s)
  open.add(s)
  TantQue $\neg$open.`empty()` **Faire**
    $v \leftarrow$ open.`peek()`
    $u \leftarrow$ un voisin de v avec u.visited = false
    Si u existe **Alors**
      u.visited $\leftarrow$ true
      `previsit`(u)
      open.add(u)
    Sinon
      open.`remove()`
    FinSi
  FinTantQue
FinProcedure



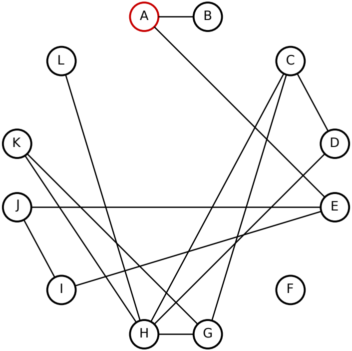
- BFScc(G), $cc = 0$

- $cc = 1$, `exploreBFScc(`G`,`A`)`



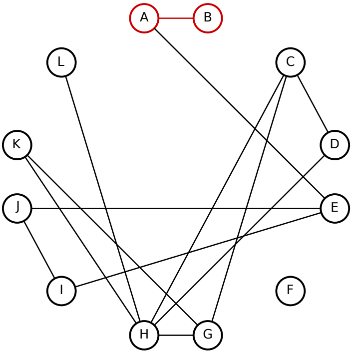
- `BFScc(`G`)`, $cc = 0$

- $cc = 1$, $\texttt{exploreBFScc}(G, A)$
  - open=A



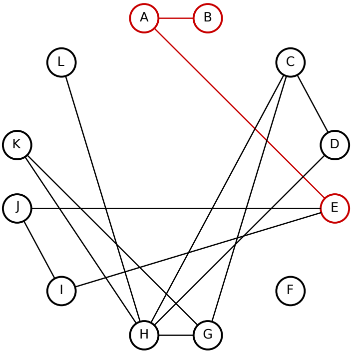
- $\texttt{BFScc}(G)$, $cc = 0$

- $cc = 1$, `exploreBFScc(`G`,`A`)`
  - open=A
  - open=A,B



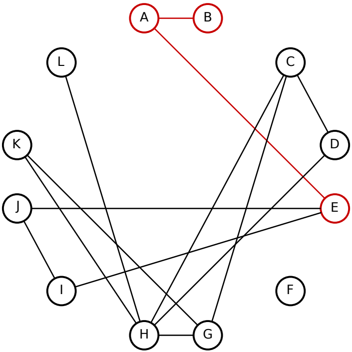
- `BFScc(`G`)`, $cc = 0$

- $cc = 1$, $\texttt{exploreBFScc}(G, A)$
  - open=A
  - open=A,B
  - open=A,B,E

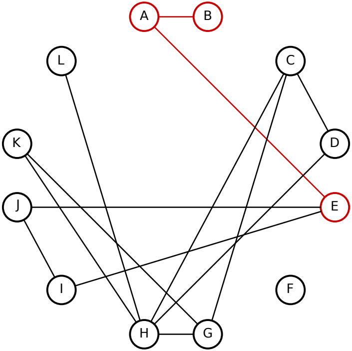


- $\texttt{BFScc}(G)$, $cc = 0$

- $cc = 1$, `exploreBFScc(`G`,`A`)`
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E

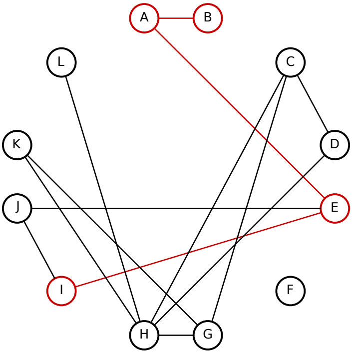


- `BFScc(`G`)`, $cc = 0$



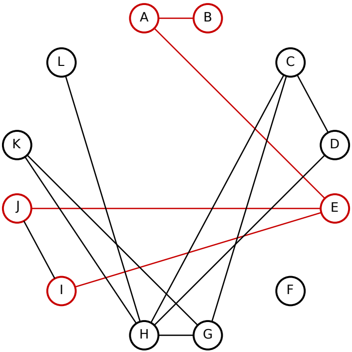
- $cc = 1$, $\texttt{exploreBFScc}(G, A)$
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E

- $\texttt{BFScc}(G)$, $cc = 0$



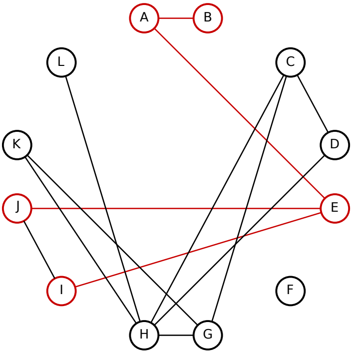
- $cc = 1$, $\texttt{exploreBFScc}(G, A)$
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I

- $\texttt{BFScc}(G)$, $cc = 0$



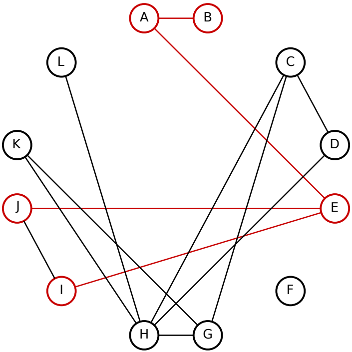
- $cc = 1$, `exploreBFScc`(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J

- `BFScc`(G), $cc = 0$



- BFScc(G), $cc = 0$

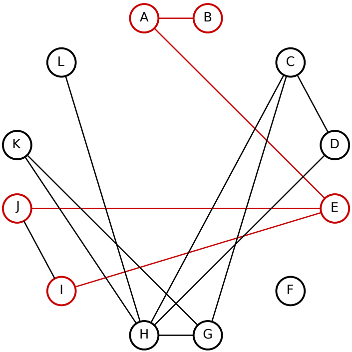
- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J



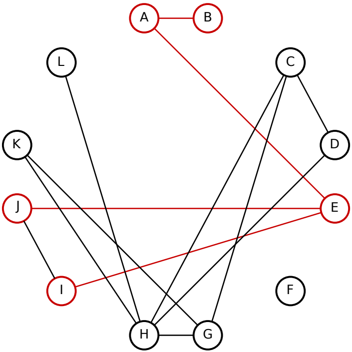
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G,A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J

- $cc = 1$, `exploreBFScc(`G`,`A`)`
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$

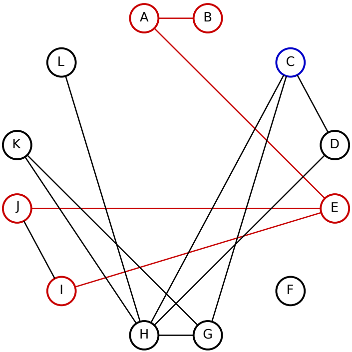


- `BFScc(`G`)`, $cc = 0$



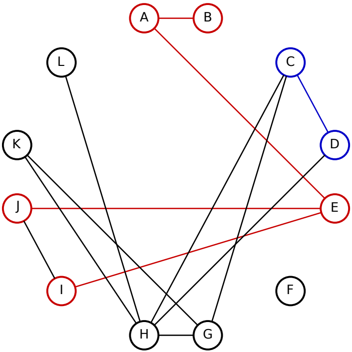
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G,A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G,C)



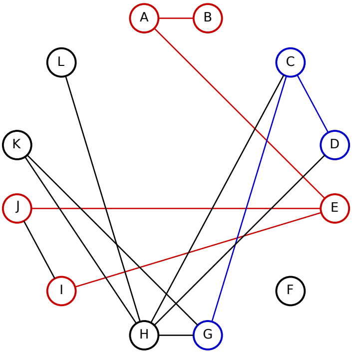
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G, C)
  - open=C



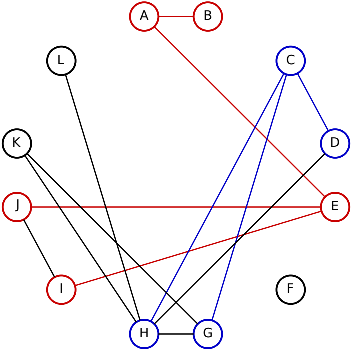
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G,A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G,C)
  - open=C
  - open=C,D



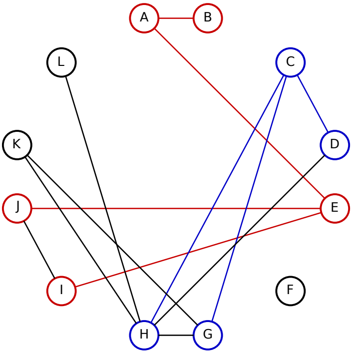
- $cc = 1$, `exploreBFScc(G,A)`
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, `exploreBFScc(G,C)`
  - open=C
  - open=C,D
  - open=C,D,G

- `BFScc(G)`, $cc = 0$



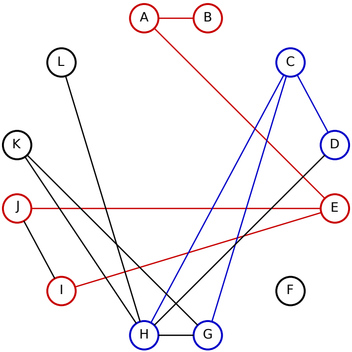
- $cc = 1$, `exploreBFScc(G,A)`
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, `exploreBFScc(G,C)`
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H

- `BFScc(G)`, $cc = 0$



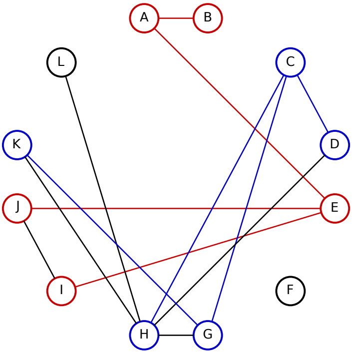
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G, C)
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H



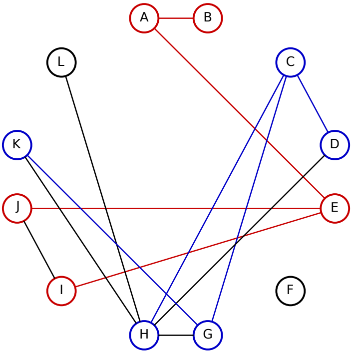
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G, C)
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H



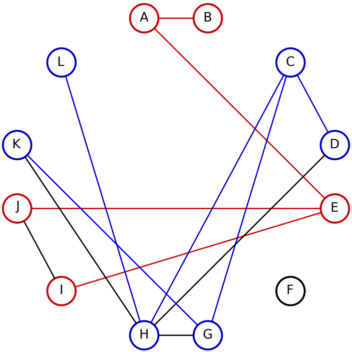
- $cc = 1$, `exploreBFScc(`G`,`A`)`
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, `exploreBFScc(`G`,`C`)`
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K

- `BFScc(`G`)`, $cc = 0$



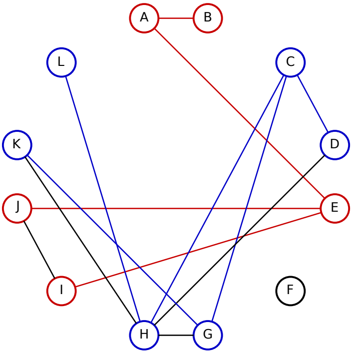
- $cc = 1$, `exploreBFScc`(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, `exploreBFScc`(G, C)
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K

- `BFScc`(G), $cc = 0$



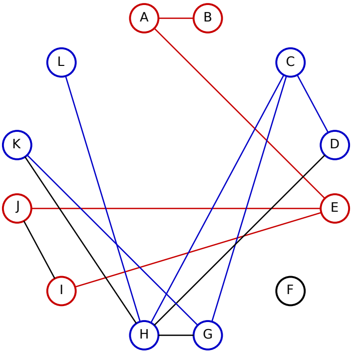
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G, C)
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K
  - open=H,K,L



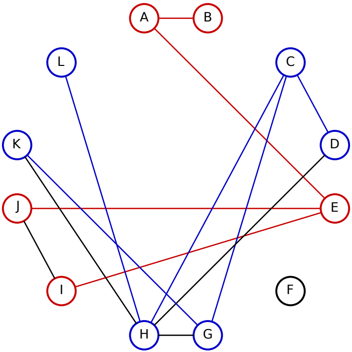
- `BFScc(`G`)`, $cc = 0$

- $cc = 1$, `exploreBFScc(`G`,`A`)`
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, `exploreBFScc(`G`,`C`)`
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K
  - open=H,K,L
  - open=K,L



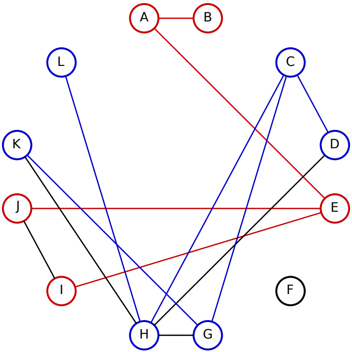
- $\texttt{BFScc}(G)$, $cc = 0$

- $cc = 1$, $\texttt{exploreBFScc}(G, A)$
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, $\texttt{exploreBFScc}(G, C)$
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K
  - open=H,K,L
  - open=K,L
  - open=L



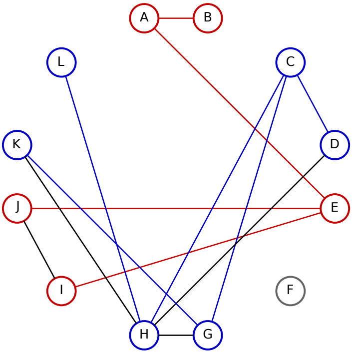
- `BFScc(`G`)`, $cc = 0$

- $cc = 1$, `exploreBFScc(`G`,`A`)`
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, `exploreBFScc(`G`,`C`)`
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K
  - open=H,K,L
  - open=K,L
  - open=L
  - open=$\emptyset$



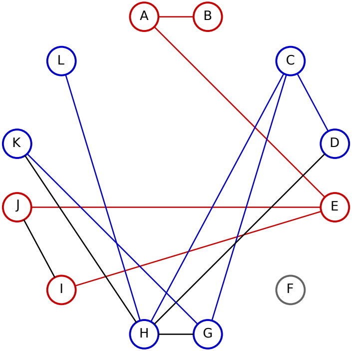
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G, C)
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K
  - open=H,K,L
  - open=K,L
  - open=L
  - open=$\emptyset$
- $cc = 3$, exploreBFScc(G, F)



- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G, C)
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K
  - open=H,K,L
  - open=K,L
  - open=L
  - open=$\emptyset$
- $cc = 3$, exploreBFScc(G, F)
  - open=F



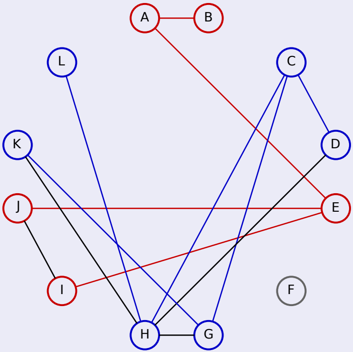
- BFScc(G), $cc = 0$

- $cc = 1$, exploreBFScc(G, A)
  - open=A
  - open=A,B
  - open=A,B,E
  - open=B,E
  - open=E
  - open=E,I
  - open=E,I,J
  - open=I,J
  - open=J
  - open=$\emptyset$
- $cc = 2$, exploreBFScc(G, C)
  - open=C
  - open=C,D
  - open=C,D,G
  - open=C,D,G,H
  - open=D,G,H
  - open=G,H
  - open=G,H,K
  - open=H,K
  - open=H,K,L
  - open=K,L
  - open=L
  - open=$\emptyset$
- $cc = 3$, exploreBFScc(G, F)
  - open=F
  - open=$\emptyset$

À la fin

- cc vaut le nombre de composantes connexes du graphe
- Chaque sommet a une étiquette correspondant au numéro de la composante connexe à laquelle il appartient

Dans l'exemple



- Composante connexe 1 : en rouge
- Composante connexe 2 : en bleu
- Composante connexe 3 : en gris

Rappel sur la notion de bipartition

- Un graphe biparti est *2-coloriable*
- Les sommets sont divisés en deux parties
- Chaque arête est entre un sommet de la première partie et un sommet de la seconde partie

Comment savoir si un graphe est biparti ?

- Utiliser un parcours
- Utiliser les procédures de traitement