Algorithmique des graphes

David Pichardie

12 Mars 2018

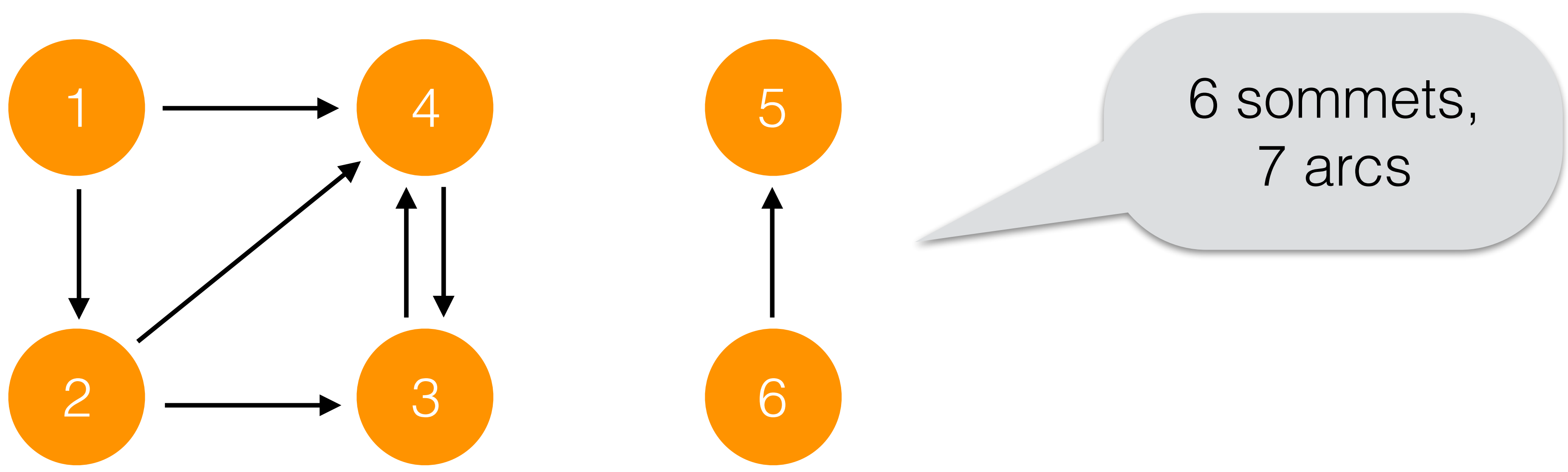
Graphe

Définition

Un *graphe orienté* est un ensemble de *sommets* connectés par des *arcs*. Formellement, un couple (S,A) avec

- S un ensemble de sommets
- A une relation binaire sur S (donc une partie de S x S)

Exemple



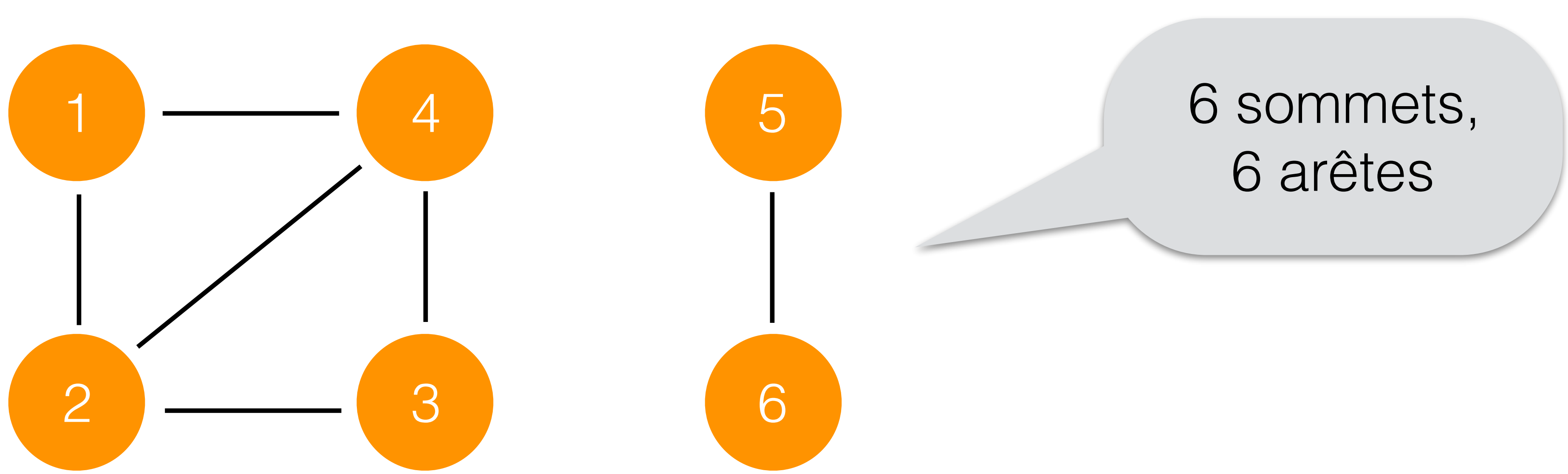
Graphe

Définition

Un *graphe non-orienté* est un ensemble de *sommets* connectés par des *arêtes*. Formellement, un couple (S,A) avec

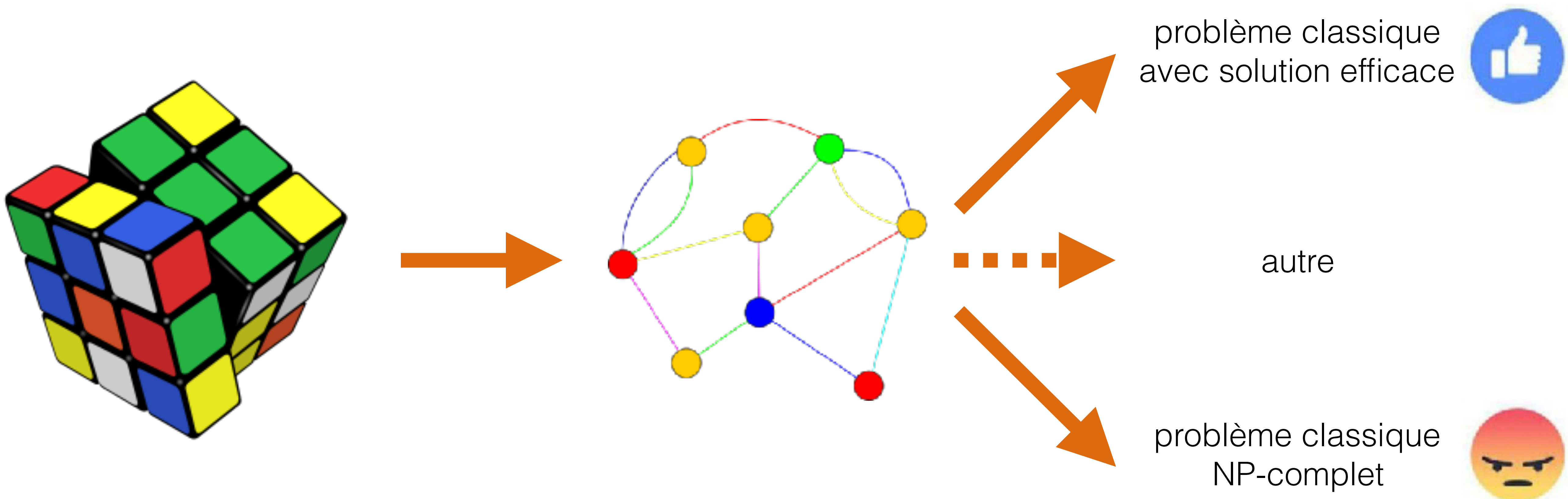
- S un ensemble de sommets
- A un ensemble de paires non-ordonnées de sommets

Exemple



Pourquoi étudier les graphes ?

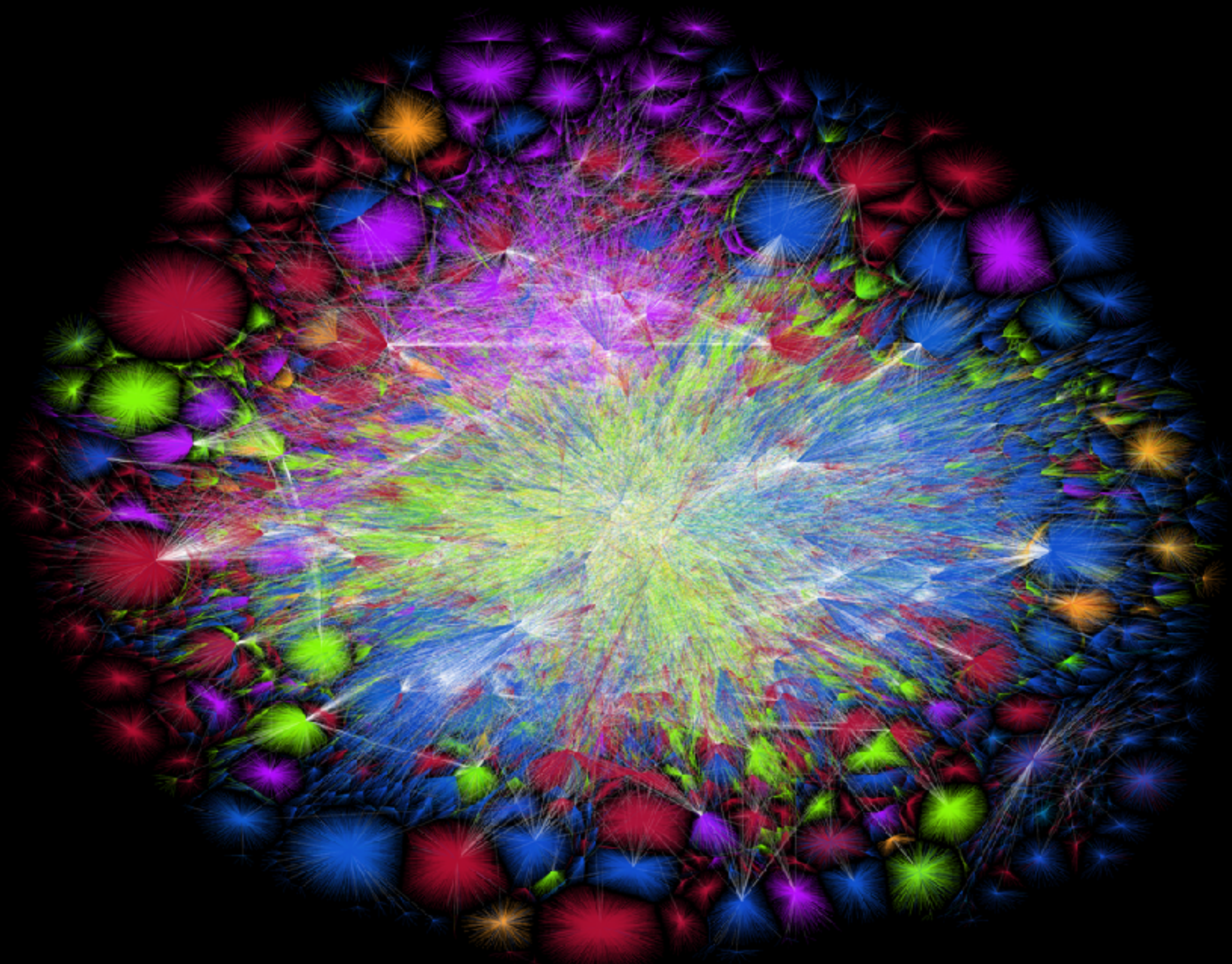
- Les graphes sont des objets abstraits élégants permettant de modeler de nombreux problèmes algorithmiques
- Il existe des algorithmes très astucieux pour résoudre certains problèmes sur les graphes
- Certains problèmes sont notoirement difficiles à résoudre


problème classique
avec solution efficace
autre
problème classique
NP-complet

Les graphes sont partout

Graphe du réseau internet

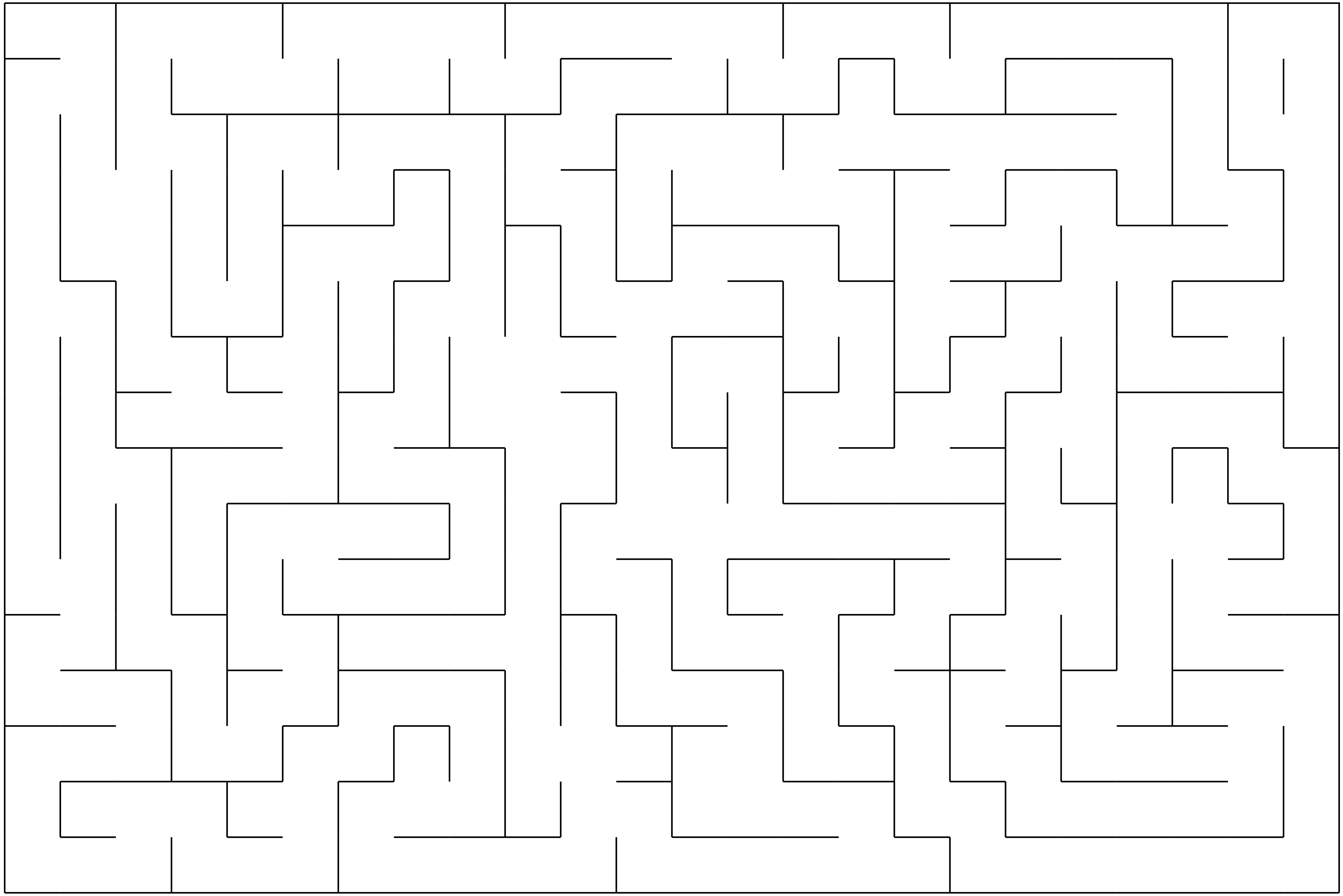
http://www.opte.org



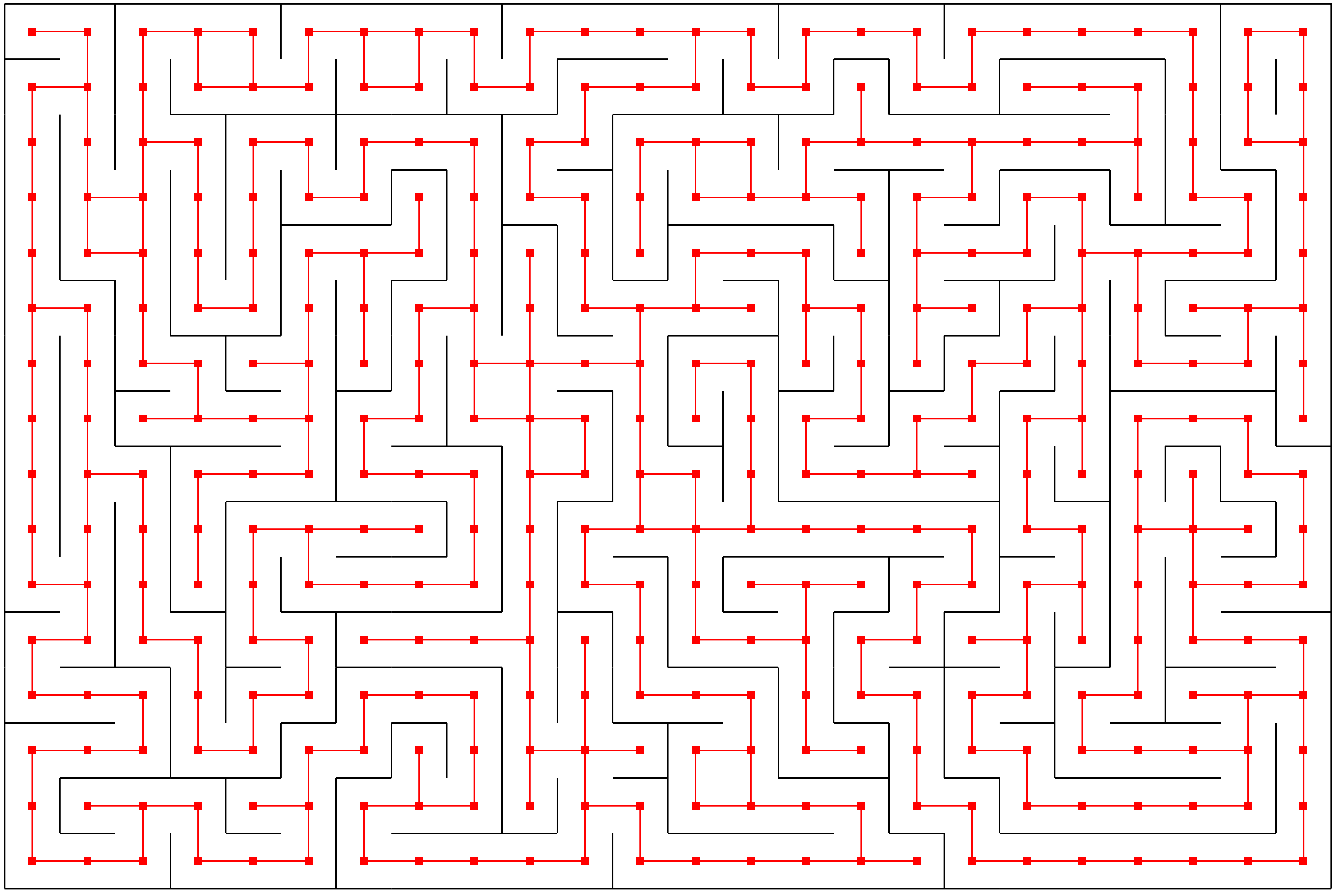
Graphe Facebook

10 millions de noeuds représentés sur les 500 millions d'utilisateurs en 2010
En 2017, Facebook compte plus de 2 milliards d'utilisateurs (actifs)

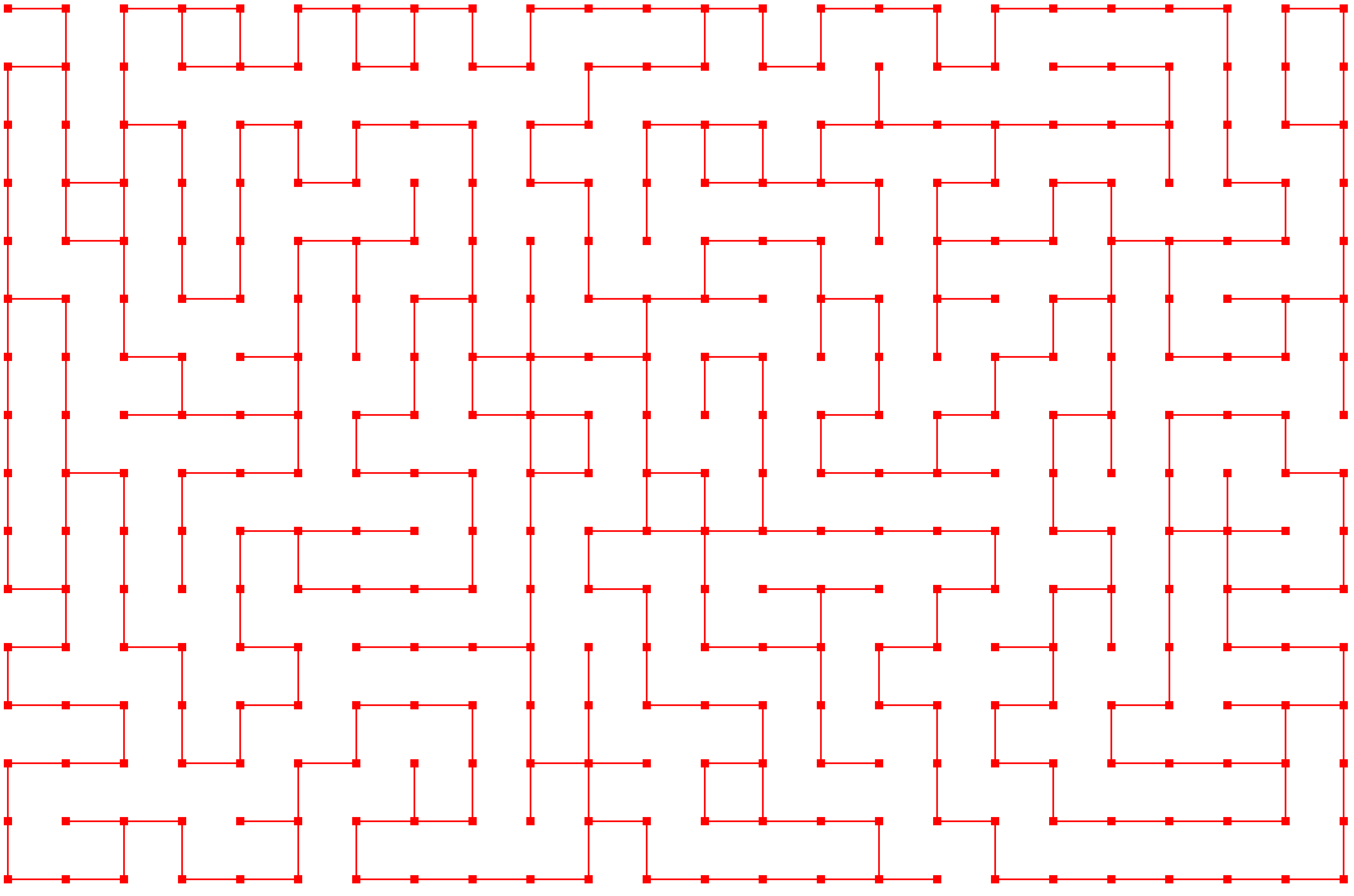
Labyrinthe



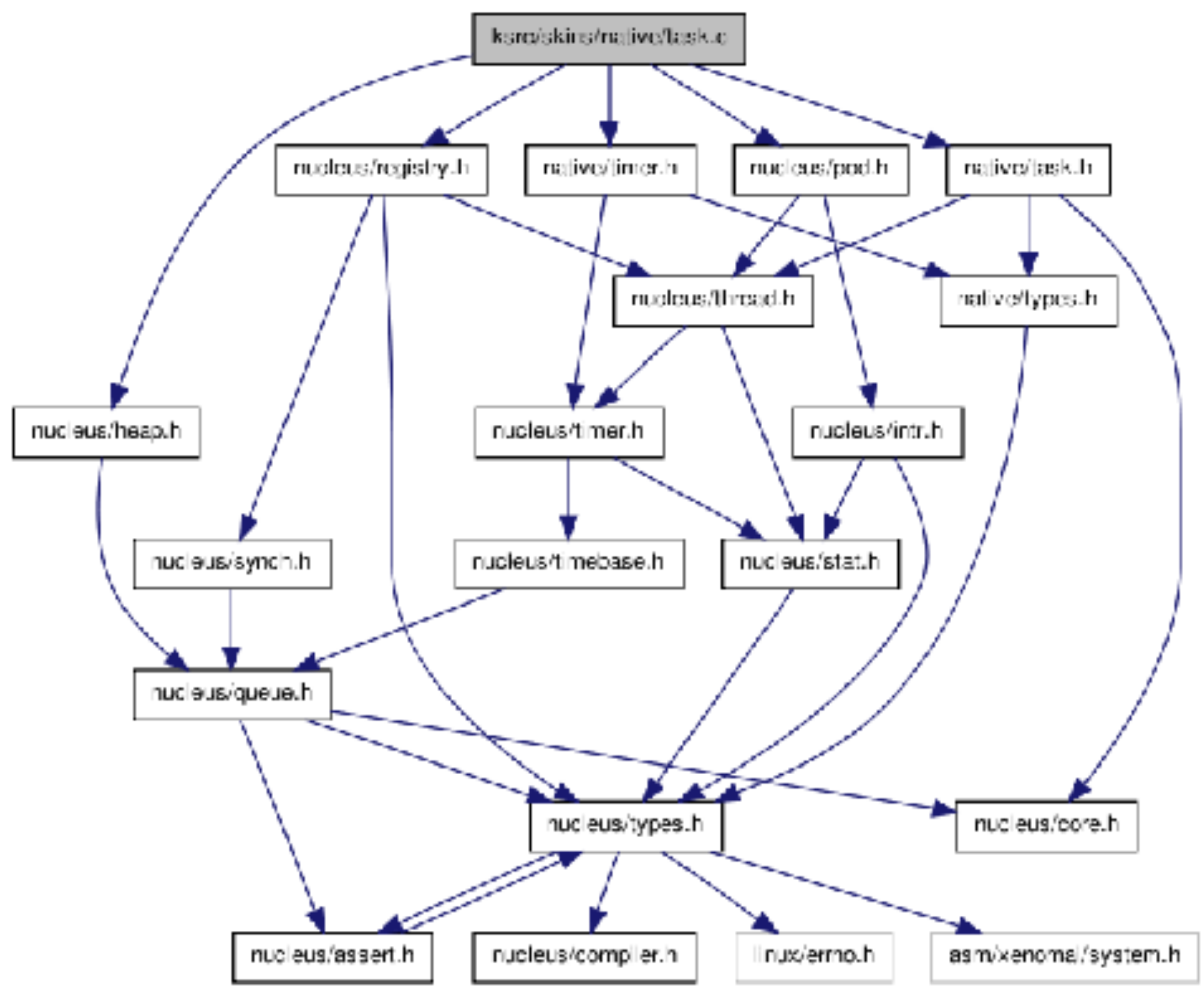
Labyrinthe



Labyrinthe

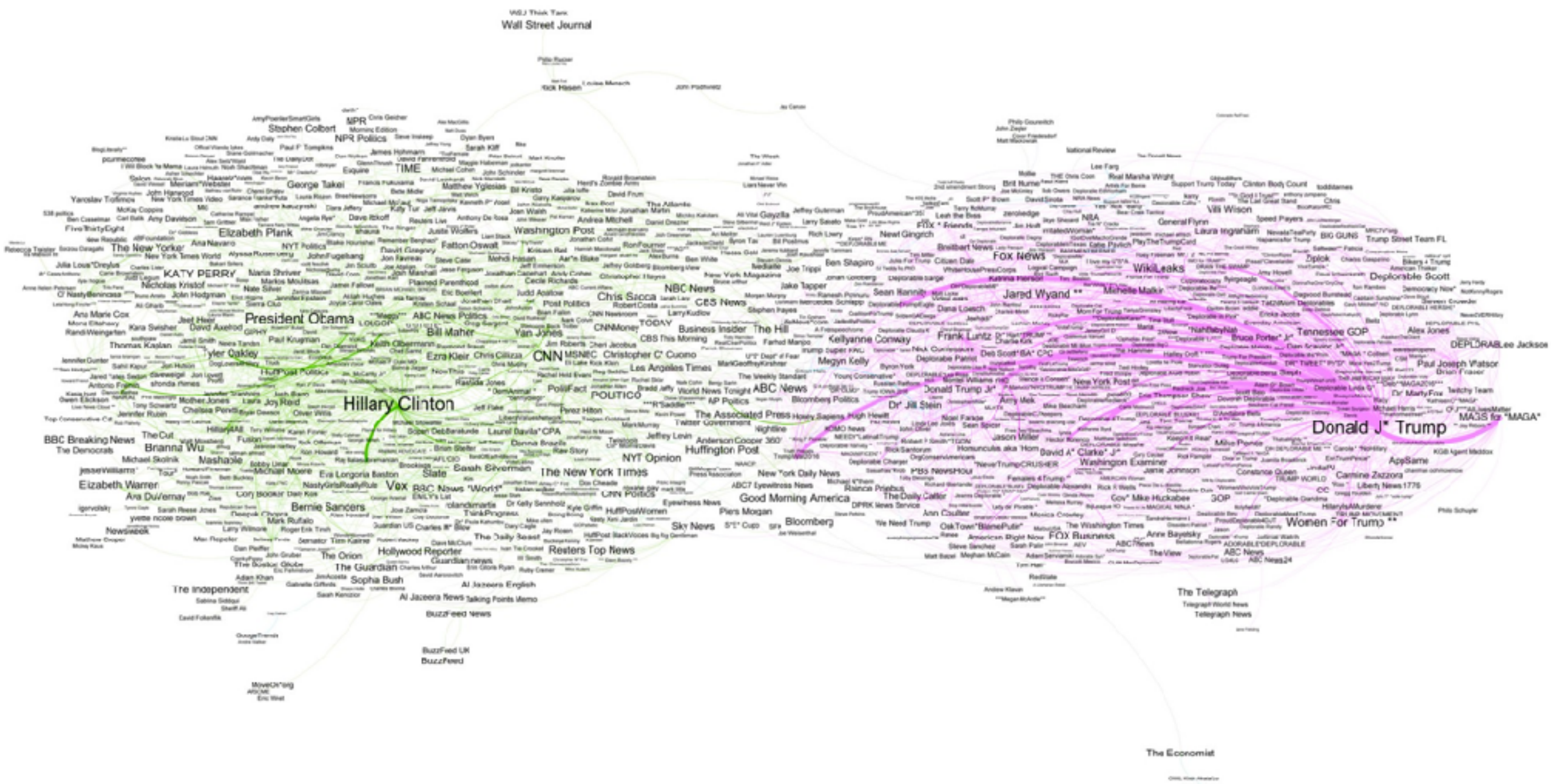


Dépendances entre des fichiers à compiler

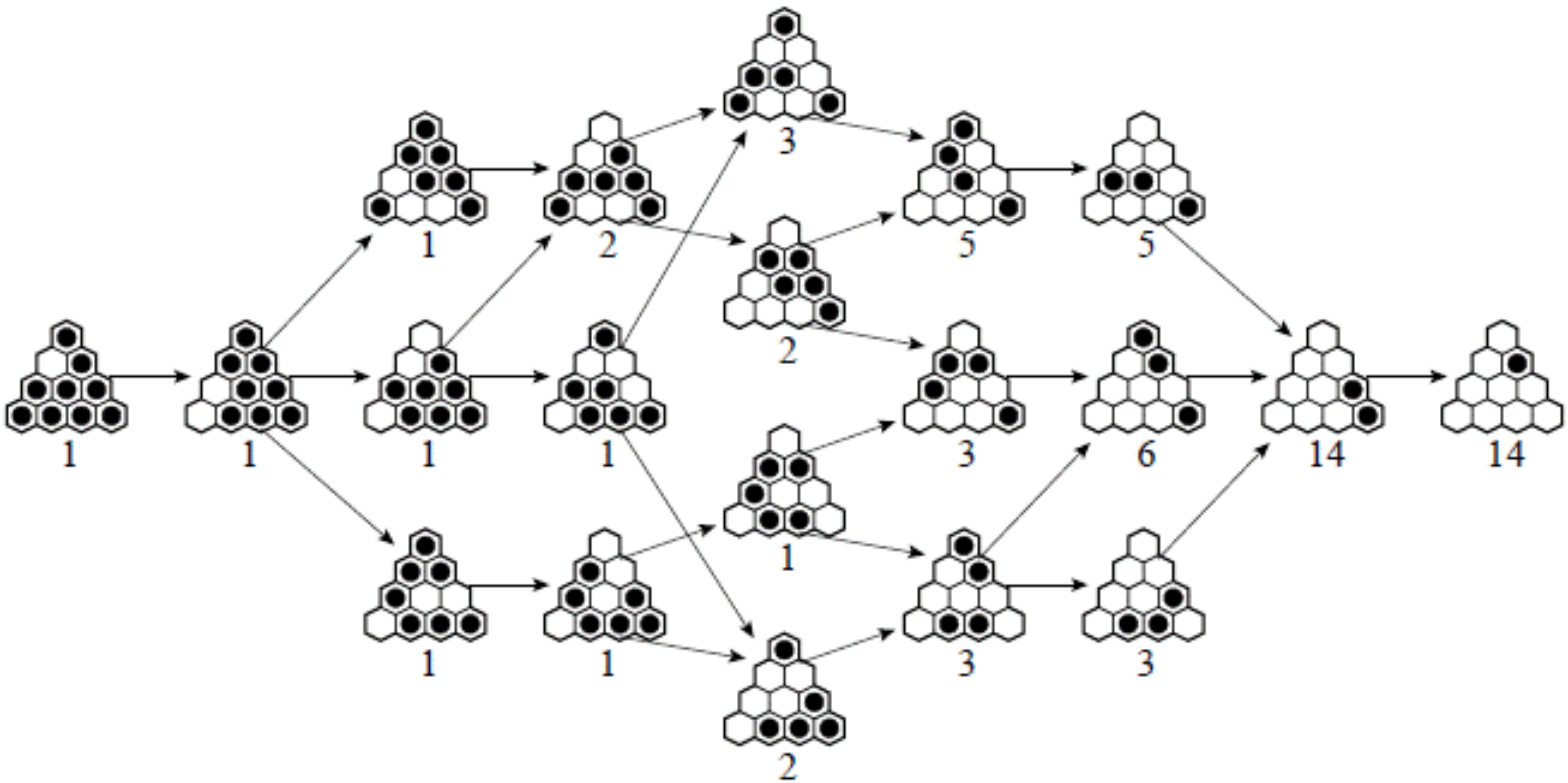


Réseau de transport

Un sous-graphe Twitter



Les configurations d’un jeu de réflexion



Les graphes sont partout

Graphe	Type	Sommets	arcs/arêtes
Internet	orienté	pages webs	hyperliens
Facebook	non-orienté	membres	être ami avec
Labyrinthe	non-orienté	positions	deux positions connexes sans mur entre elles
Compilation	orienté	fichiers	un fichier dépend d'un autre pour être compilé
Métro	non-orienté	stations	liaison directe entre deux stations
Twitter	orienté	membres	être abonné à
Jeux de réflexion	orienté	configuration de jeu	atteignable en un coup

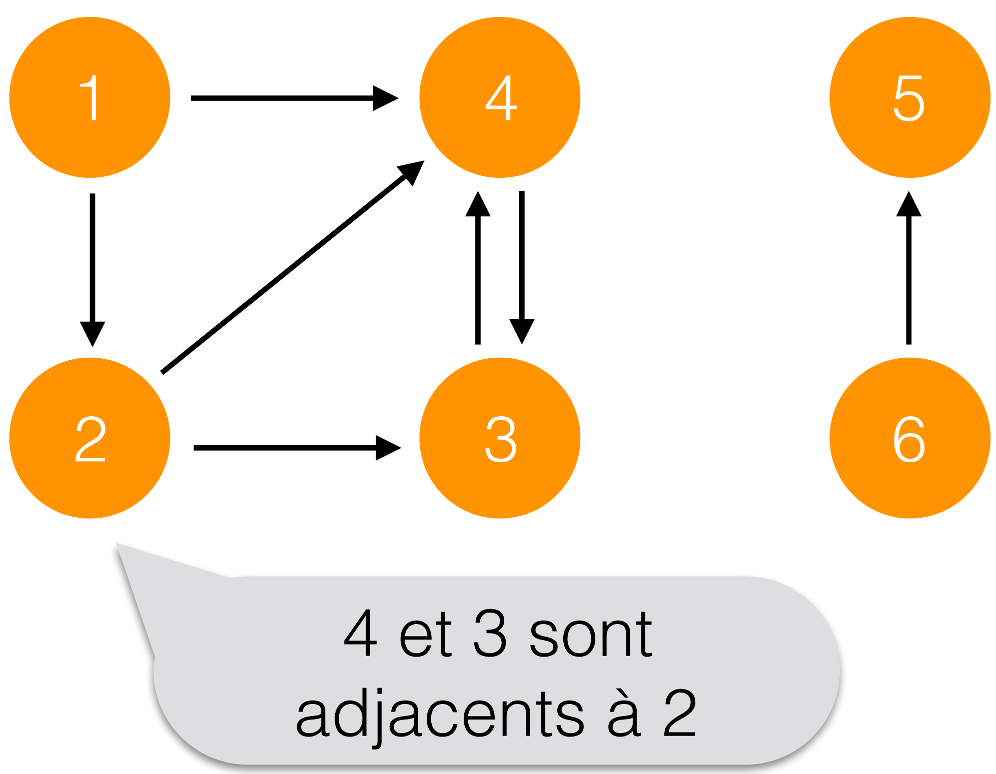
Vocabulaire

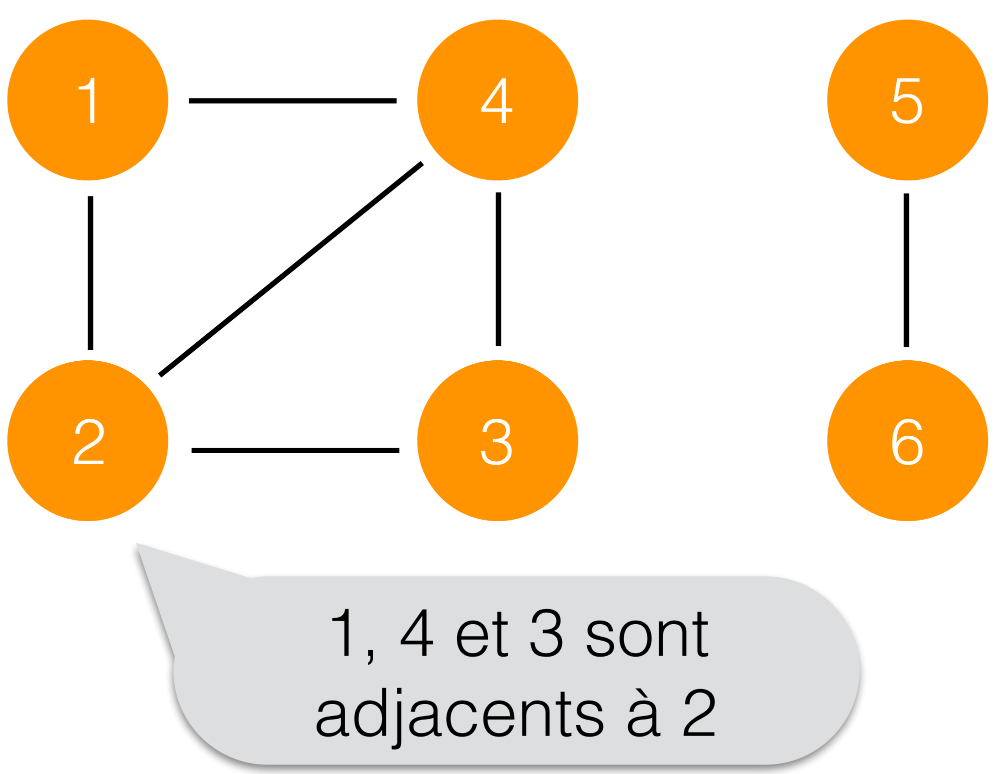
Vocabulaire

Définition

Dans un graphe (S,A) orienté ou non, un sommet y est dit *adjacent* à x, ssi *(x,y)* appartient à *A.*

Exemple



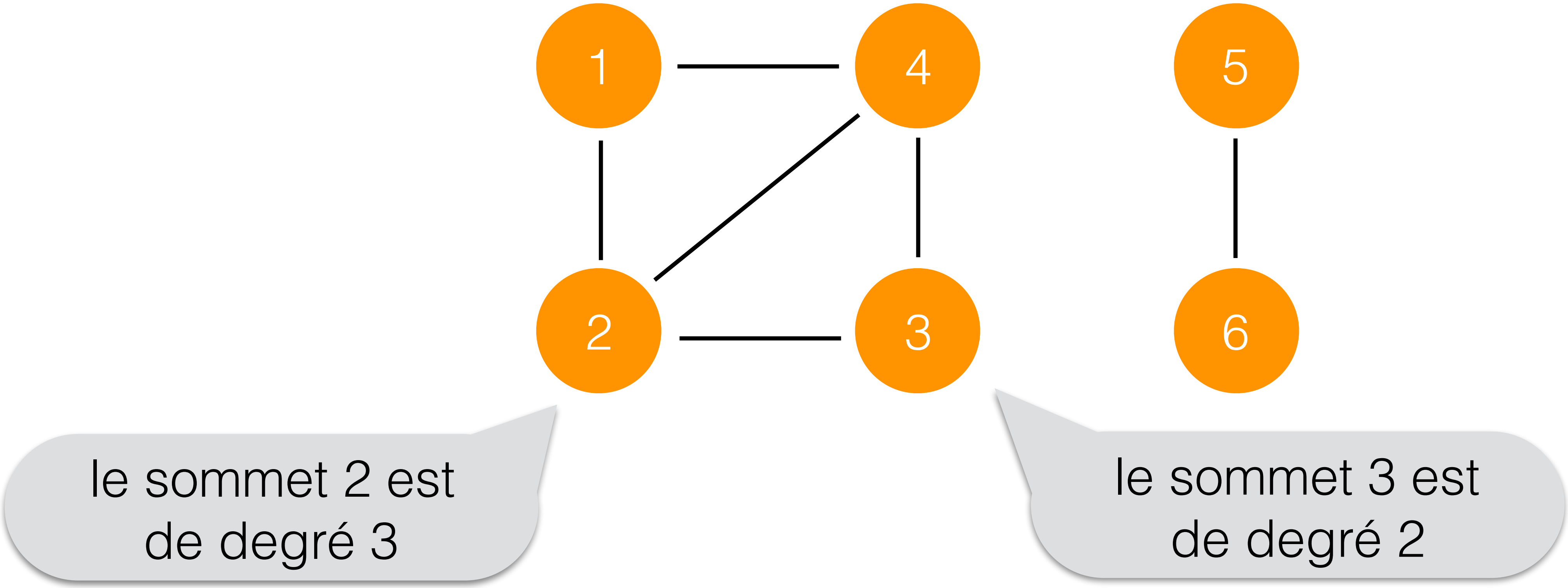


Vocabulaire

Définition

Dans un graphe (S,A) non-orienté, le degré d'un sommet est le nombre de ses sommets adjacents

Exemple



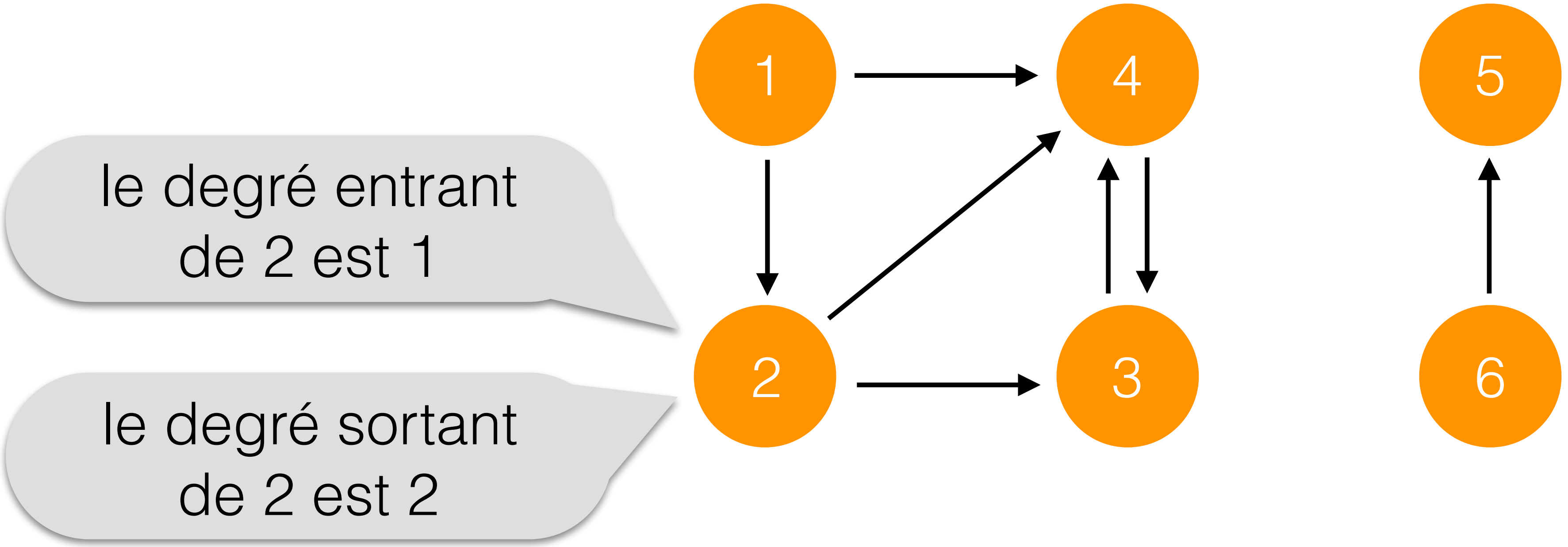
Vocabulaire

Définition

Dans un graphe (S,A) orienté,

- le *degré sortant* d'un sommet est le nombre de ses sommets adjacents,
- son *degré entrant* est le nombre de sommets auxquels il est adjacent.

Exemple

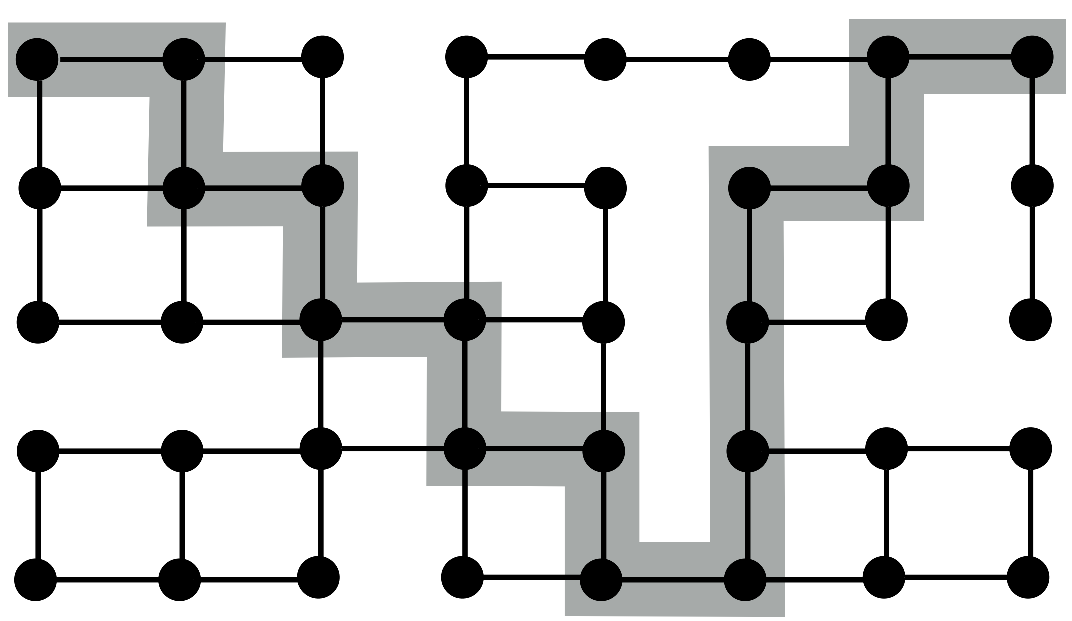


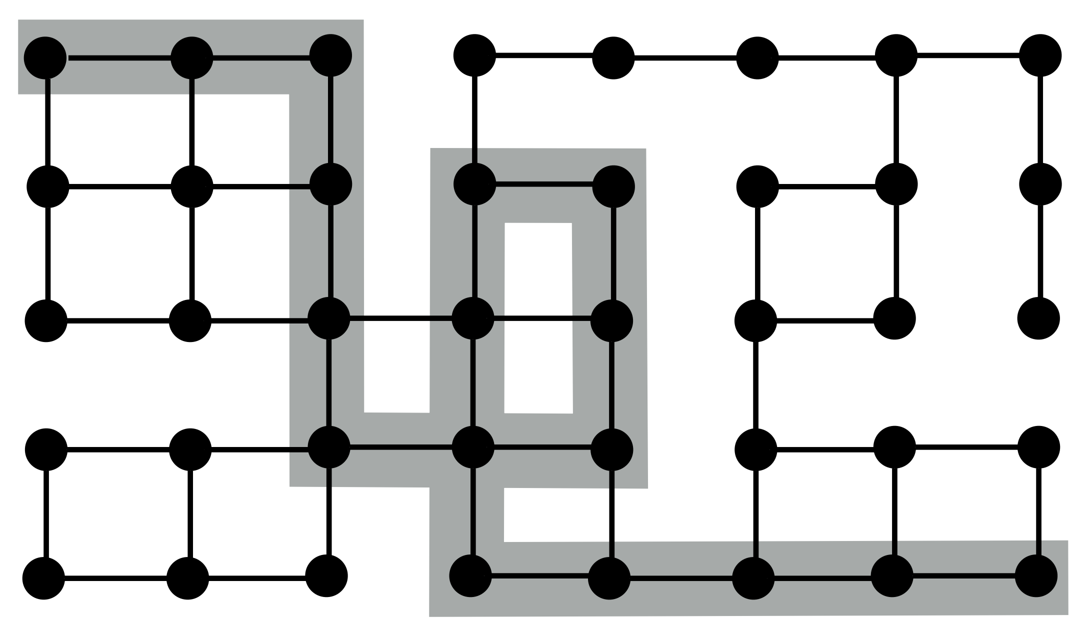
Vocabulaire

Définition

Dans un graphe *(S,A)* orienté ou non, un *chemin* est une séquence de sommets $s_0, \ldots, s_n$ où chaque paire de sommets consécutifs (s_k, s_{k+1}) appartient à *A.*

Exemples



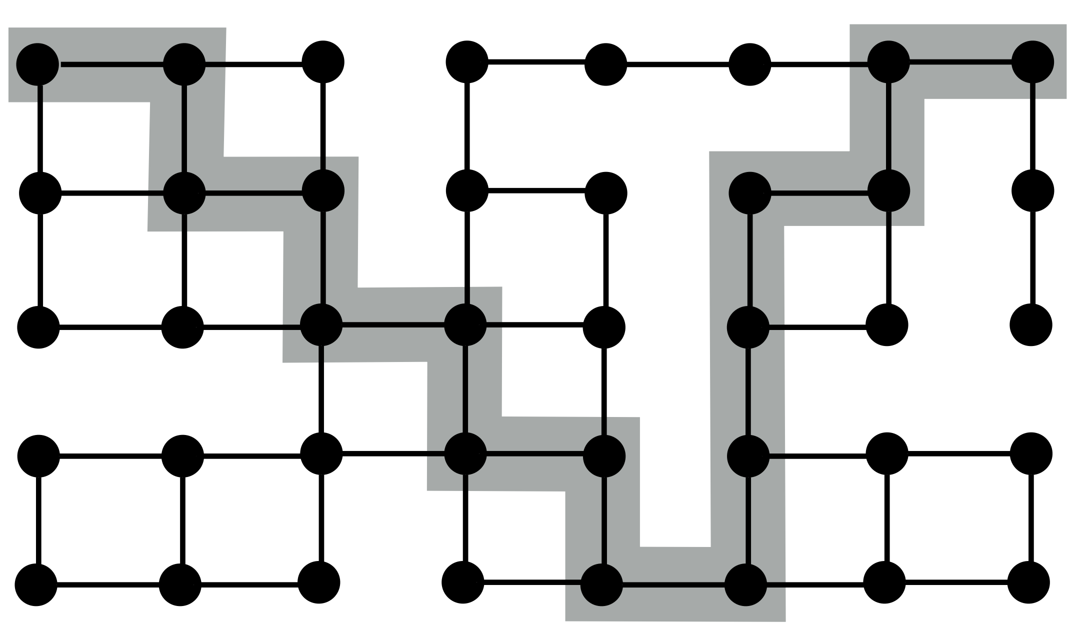


Vocabulaire

Définition

Dans un graphe *(S,A)* orienté ou non, un *chemin simple* est un chemin sans répétitions de sommet.

Exemple

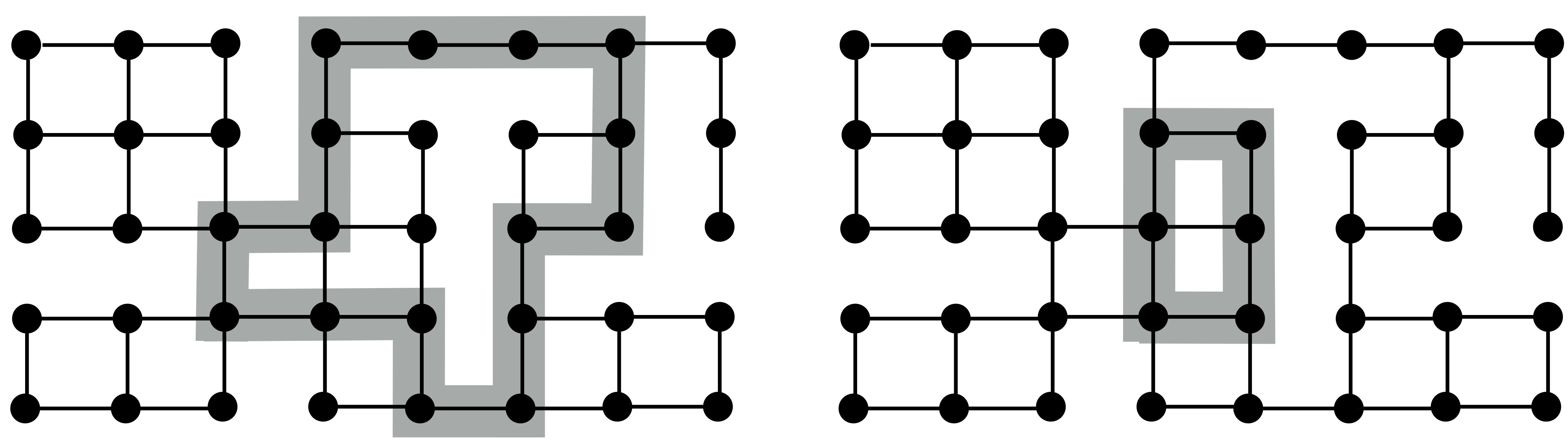


Vocabulaire

Définition

Dans un graphe *(S,A)* orienté ou non, un *cycle* est un chemin $s_0, \ldots, s_n$ où $s_0 = s_n$ et $s_0, \ldots, s_{n-1}$ sont distincts.

Exemples

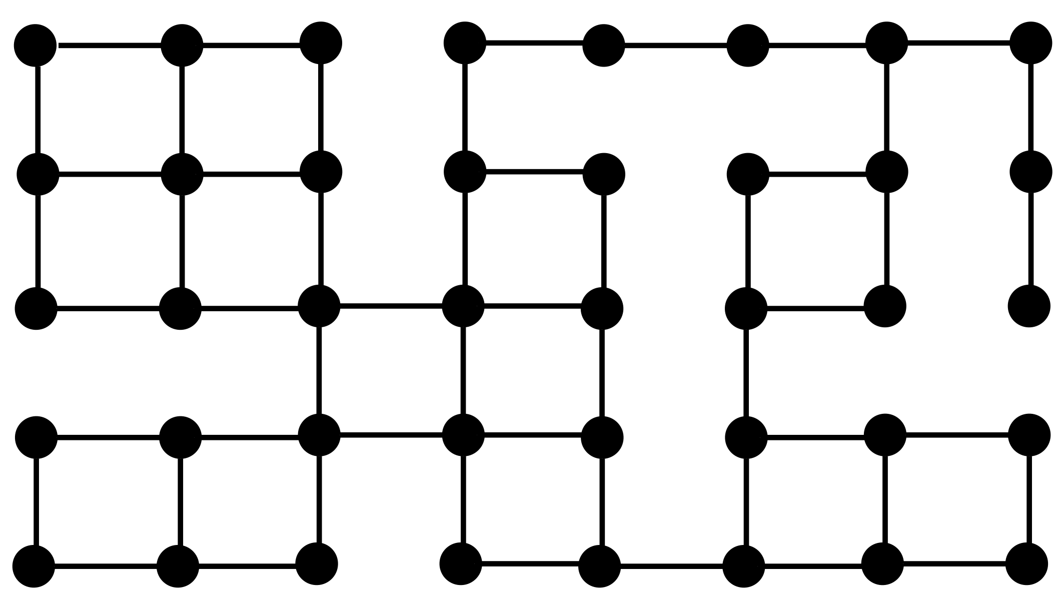


Vocabulaire

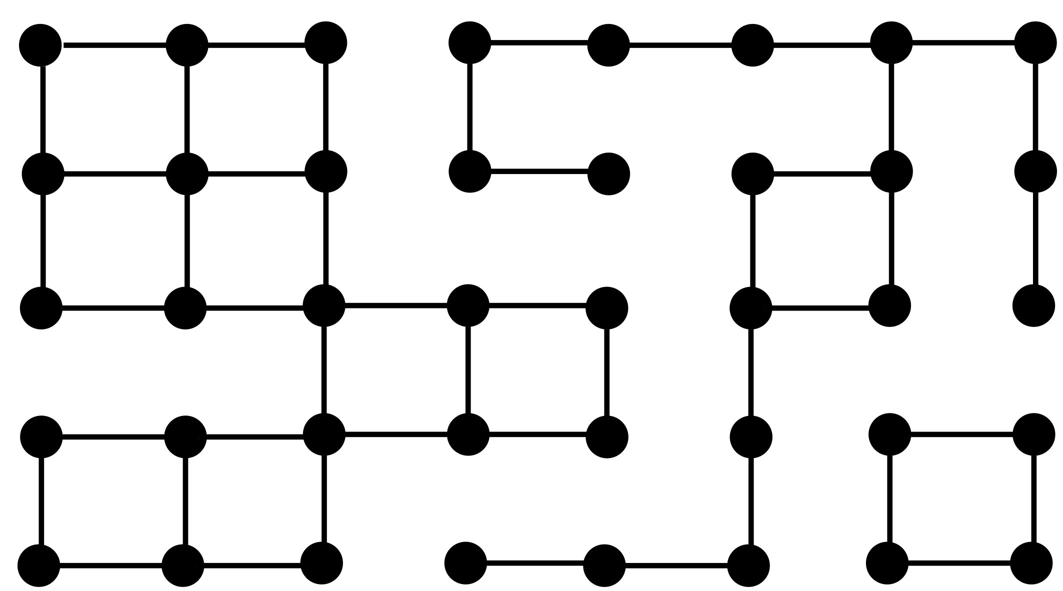
Définition

Un graphe *(S,A)* **non-orienté** est *connexe* ssi toute paire de sommet est relié par au moins un chemin.

Exemple



Contre - Exemple



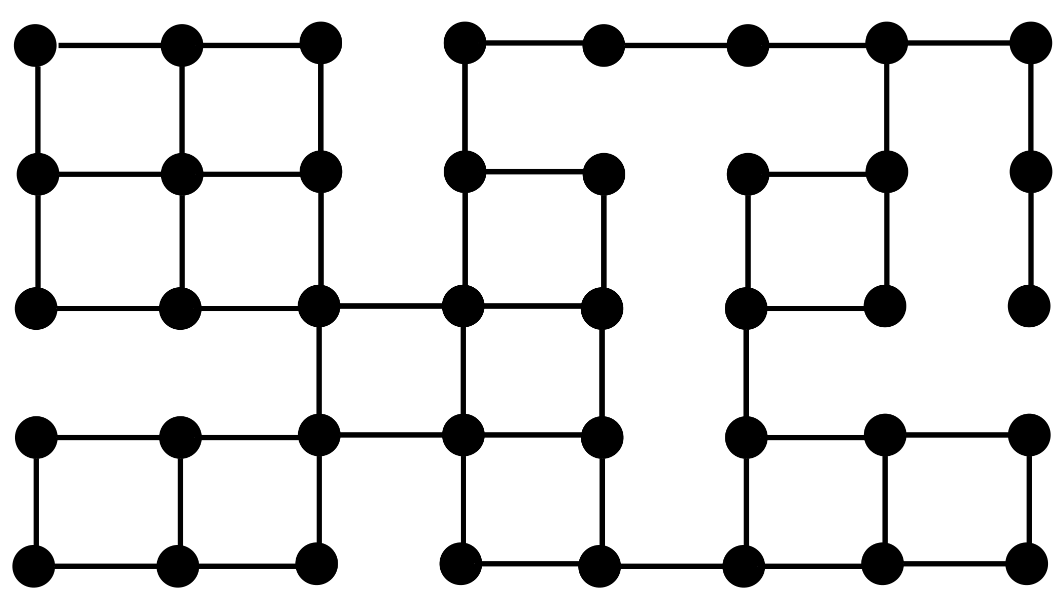
Vocabulaire

pour les graphes orientés, nous parlerons plus tard de *connexité forte*

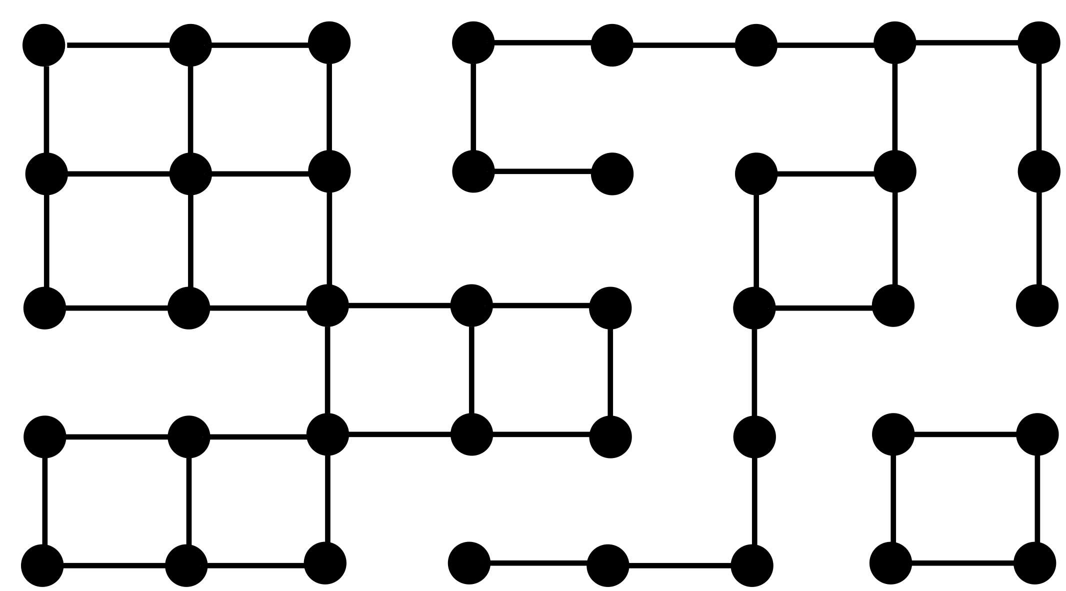
Définition

Un graphe *(S,A)* **non-orienté** est *connexe* ssi toute paire de sommet est relié par au moins un chemin.

Exemple



Contre - Exemple



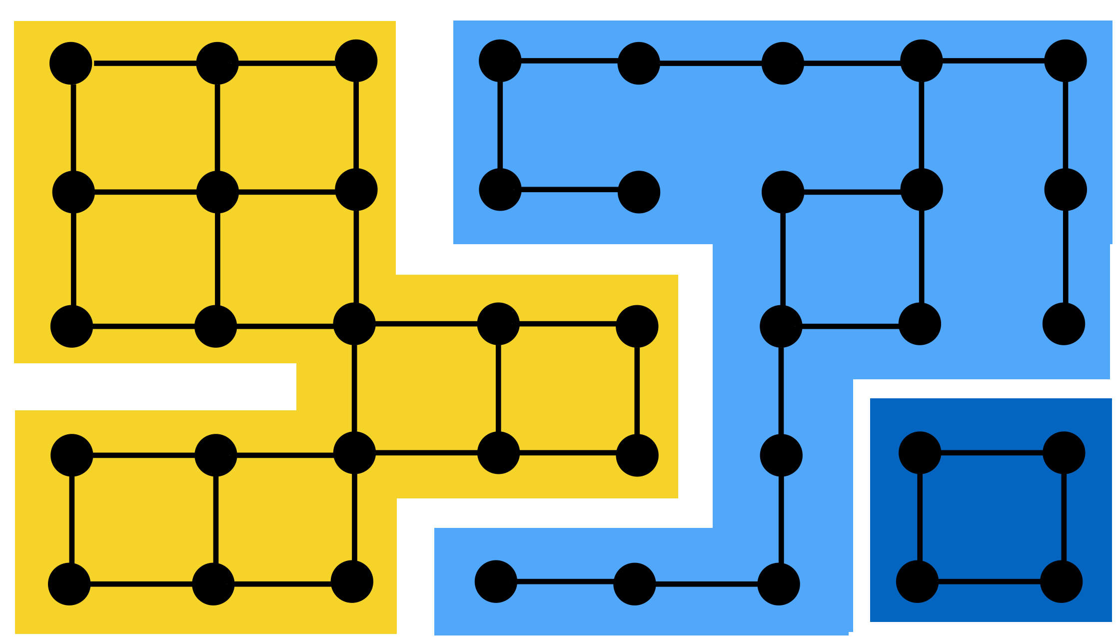
Vocabulaire

Définition

Un graphe *(S,A)* **non-orienté** est *connexe* ssi toute paire de sommet est relié par au moins un chemin.

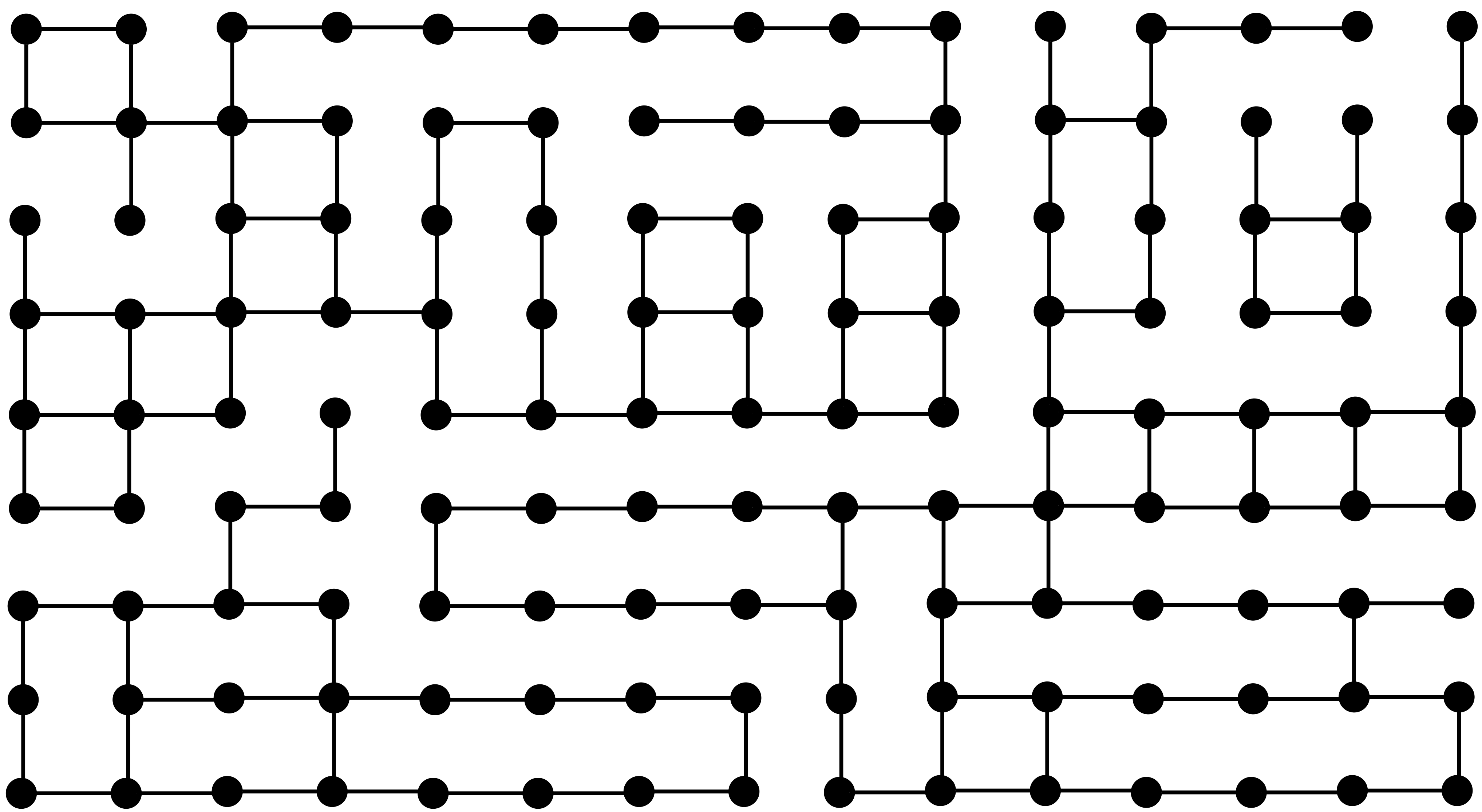
Contre - Exemple

un graphe peut toujours être décomposé en une union disjointe de sous-graphes connexes : *ses composantes connexes*



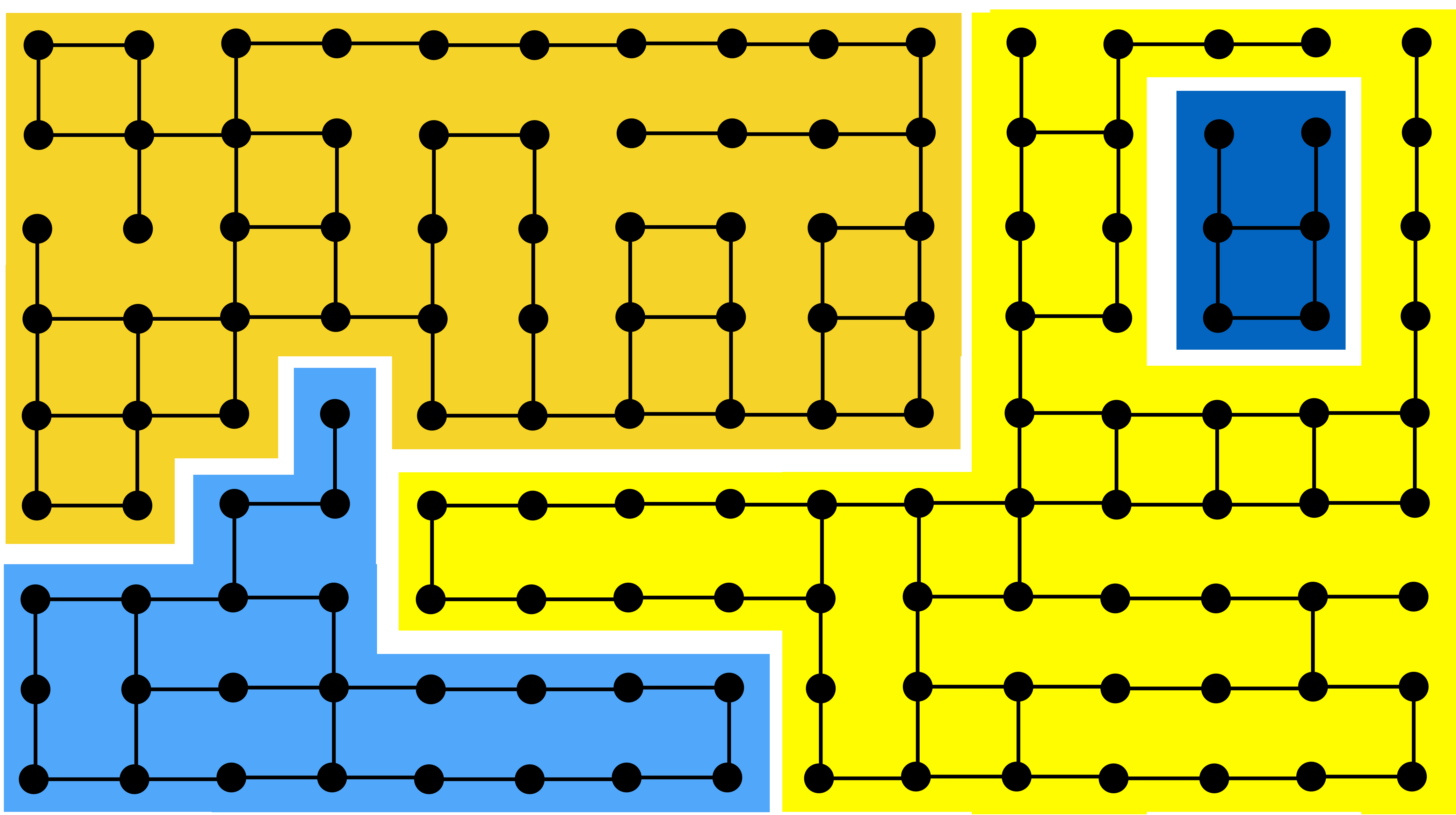
Question

Combien de composantes connexes le graphe suivant comporte-t-il ?



Question

4



Représentation des graphes (orientés ou non)

Représentation des graphes (orientés ou non)

On suppose que
S = [1, N]

même représentation
mais avec de la
redondance

Au besoin, on utilise un dictionnaire pour faire le
lien entre sommets et indexes

Opérations de bases

Opérations de bases

- Créer un graphe avec *N* sommet, sans arcs

Opérations de bases

- Créer un graphe avec *N* sommet, sans arcs
- Ajouter un arc

Opérations de bases

- Créer un graphe avec N sommet, sans arcs
- Ajouter un arc
- Tester l’existence d’un arc

Opérations de bases

- Créer un graphe avec *N* sommet, sans arcs
- Ajouter un arc
- Tester l’existence d’un arc
- Parcourir les sommets adjacents à un sommet

Opérations de bases

- Créer un graphe avec *N* sommet, sans arcs
- Ajouter un arc
- Tester l'existence d'un arc
- Parcourir les sommets adjacents à un sommet
- Parcourir **tous** les sommets en explorant le graphe, tel un labyrinthe → un *parcours* de graphe

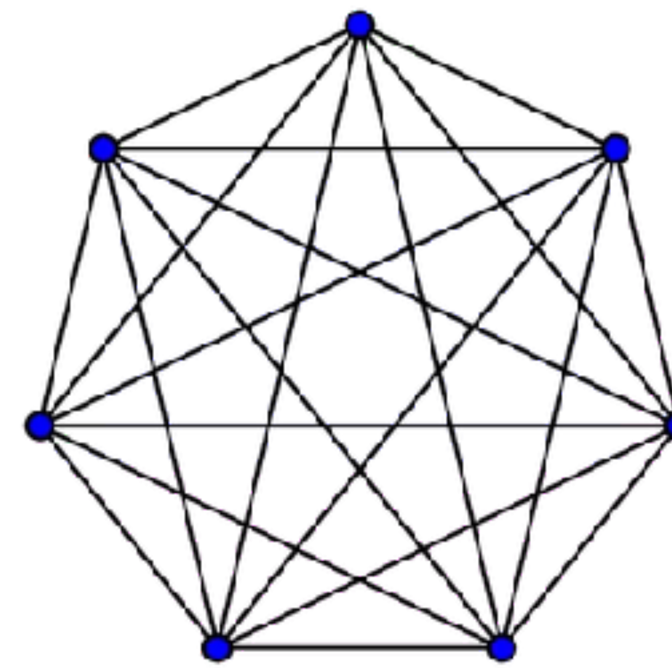
notre premier algorithme dans ce cours

Représentation par matrice d'adjacences

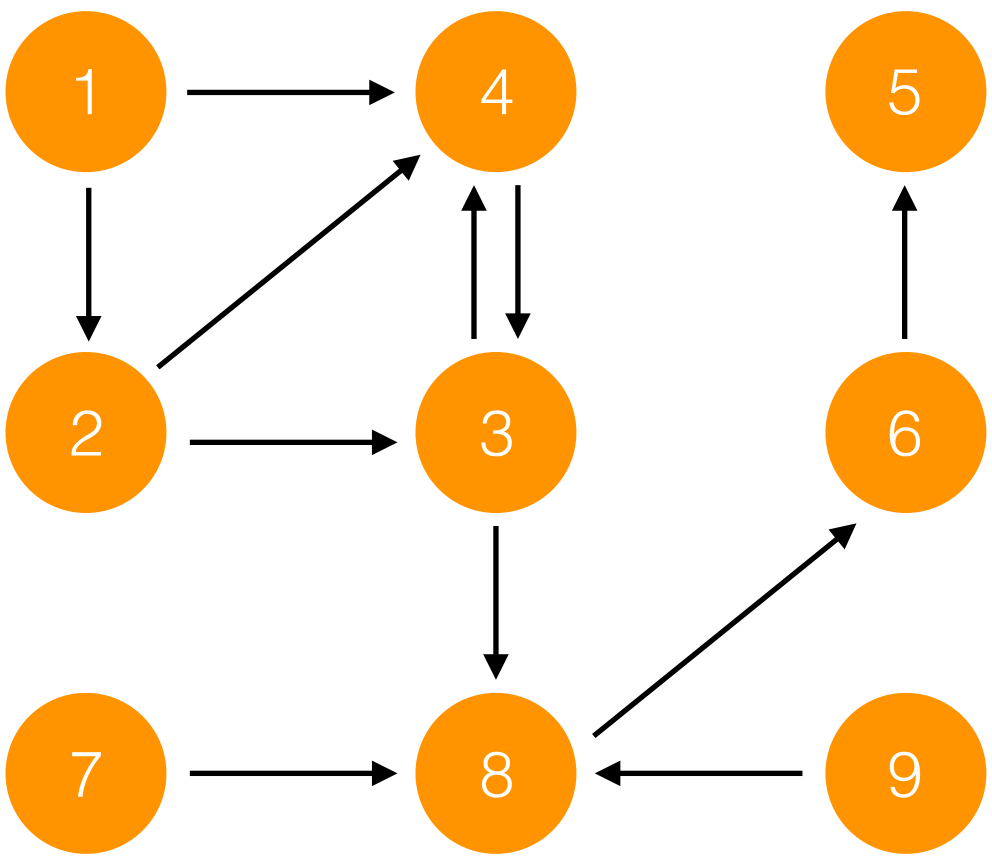
- une matrice *N x N* à valeurs dans *{0,1}*

$$M[i,j] = \begin{cases} 1 & \text{si } (i,j) \in A \\ 0 & \text{sinon} \end{cases}$$

- bien adapté aux graphes denses

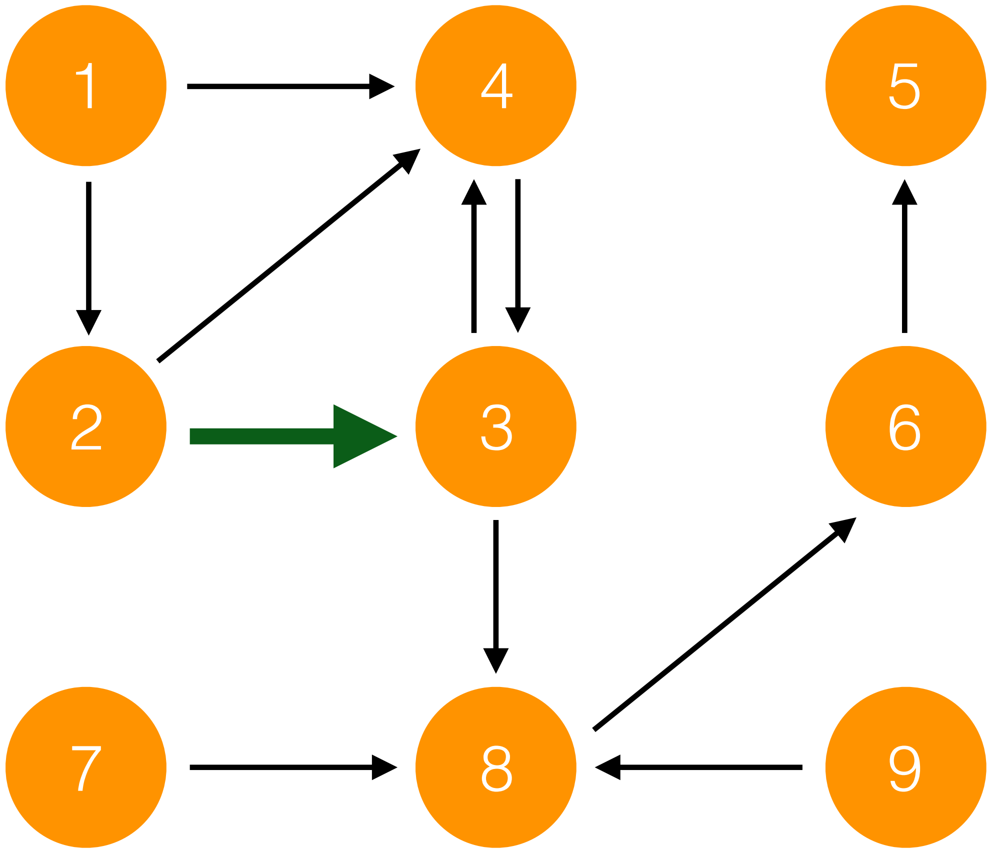


Exemple



	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	0	1	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	1	0

Exemple



	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	0	0	0
2	0	0	1	1	0	0	0	0	0
3	0	0	0	1	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	0	0	1	0

Un peu de dénombrement

$$|A| \leq \; ?$$

Un peu de dénombrement

nombre de sommets

$|A| \leq$?

- si $|S|=n$, comment majorer $|A|$?

Un peu de dénombrement

nombre de sommets

$|A| \leq \;?$

- si $|S|=n$, comment majorer $|A|$?
- dans un graphe orienté, le nombre maximum d'arcs est n^2 : chaque paire de sommets est alors relié par un arc

Un peu de dénombrement

nombre de sommets

$|A| \leq$?

- si $|S|=n$, comment majorer $|A|$?
- dans un graphe orienté, le nombre maximum d'arcs est n^2 : chaque paire de sommets est alors relié par un arc
- dans un graphe non-orienté, le nombre maximum d'arêtes est $n(n+1)/2$

Caractérisation des graphes denses

Un graphe est dense si

$$|A| \sim n^2$$

de l'ordre de

Connaissez vous un exemple de graphe dense ?

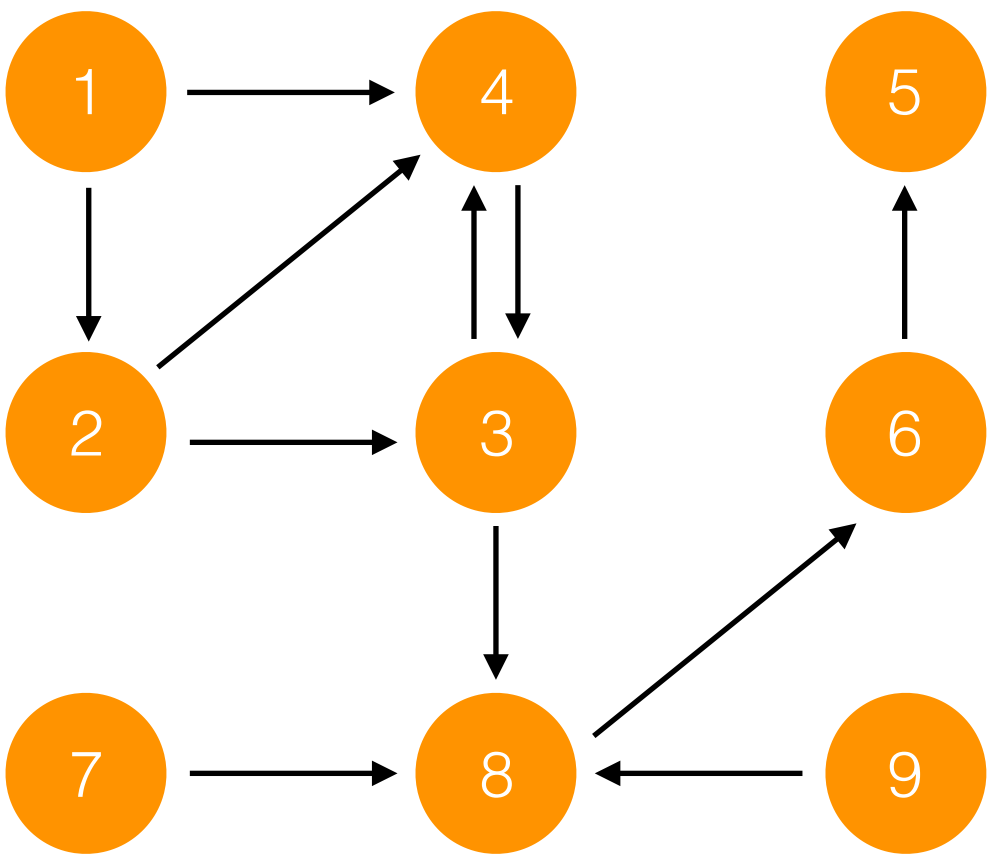
Représentation par liste d'adjacences

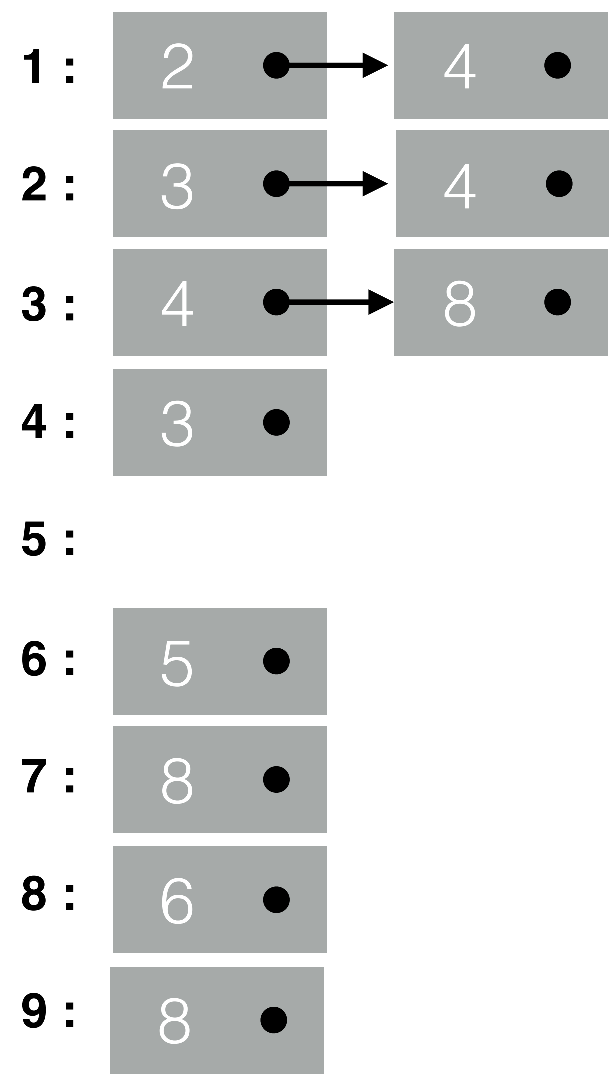
- un tableau *Adj* de $|S|$ listes tel que, pout tout sommet *i*, *Adj[i]* contient les adjacents de *i.*

$$Adj[i] = \{j \mid (i, j) \in A\}$$

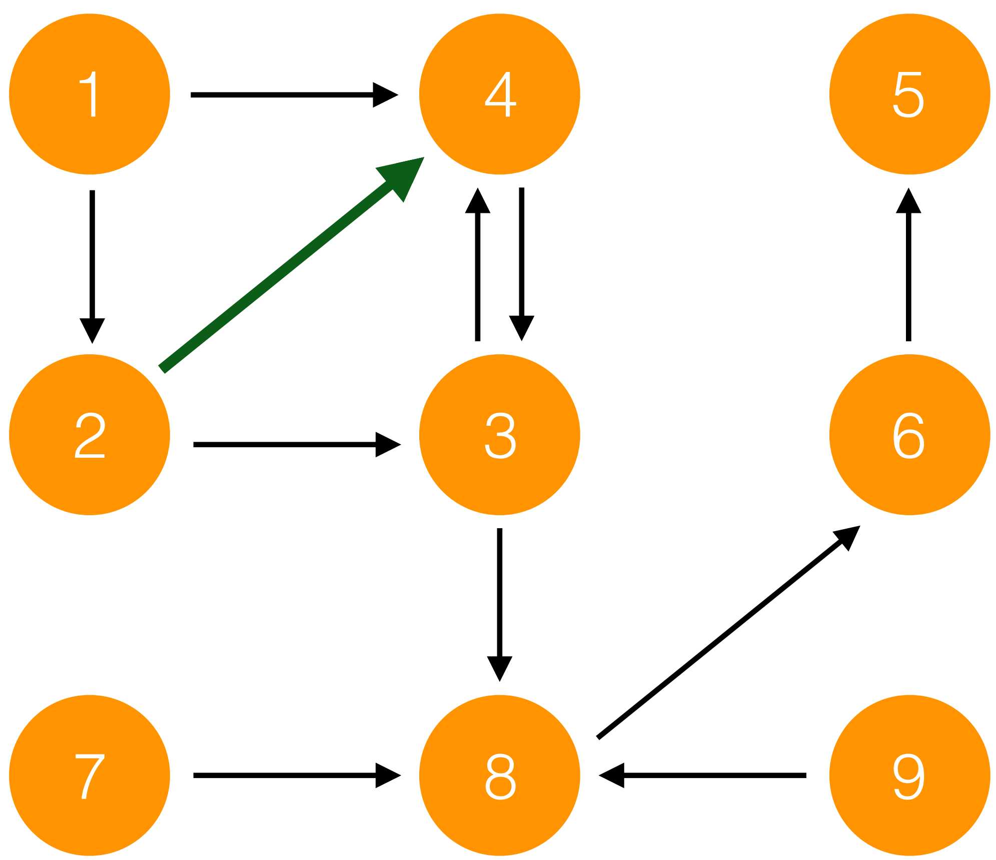
- bien adapté aux graphes peu denses (*sparses*)
  - web, réseaux sociaux…

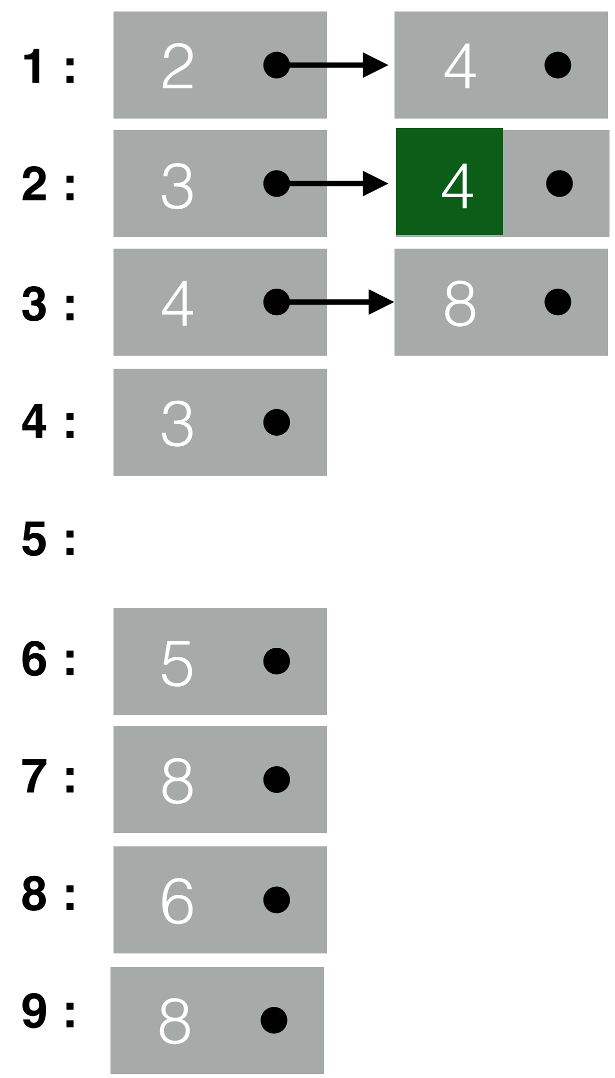
Exemple





Exemple





Autres représentations

- une seule liste de tous les arcs/arêtes
- un tableau des adjacents, mais en utilisant des ensembles plutôt que des listes simplement chaînées

Efficacités des représentations

Efficacités des représentations

Représentation	Espace	Tester si *(i,j)* appartient à *A* ?	Parcourir les adjacents de *i*

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs			

Efficacités des représentations

Représentation	Espace	Tester si *(i,j)* appartient à *A* ?	Parcourir les adjacents de *i*
liste des arcs	O(|A|)		

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	

Efficacités des représentations

Représentation	Espace	Tester si *(i,j)* appartient à *A* ?	Parcourir les adjacents de *i*
liste des arcs	O(|A|)	O(|A|)	O(|A|)

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence			

Efficacités des représentations

Représentation	Espace	Tester si *(i,j)* appartient à *A* ?	Parcourir les adjacents de *i*
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|$^{2)}$		

Efficacités des représentations

Représentation	Espace	Tester si *(i,j)* appartient à *A* ?	Parcourir les adjacents de *i*
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|$^{2)}$	O(1)	

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|2))	O(1)	O(|S|)

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	$O(\|A\|)$	$O(\|A\|)$	$O(\|A\|)$
matrice d'adjacence	$O(\|S\|^{2)}$	$O(1)$	$O(\|S\|)$
listes d'adjacence			

Efficacités des représentations

Représentation	Espace	Tester si *(i,j)* appartient à *A* ?	Parcourir les adjacents de *i*
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|2))	O(1)	O(|S|)
listes d'adjacence	O(|A|+|S|)		

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d’adjacence	O(|S|2)	O(1)	O(|S|)
listes d’adjacence	O(|A|+|S|)	O(degré(i))	

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|2)	O(1)	O(|S|)
listes d'adjacence	O(|A|+|S|)	O(degré(i))	O(degré(i))

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|^{2})	O(1)	O(|S|)
listes d'adjacence	O(|A|+|S|)	O(degré(i))	O(degré(i))
ensembles d'adjacence			

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|2)	O(1)	O(|S|)
listes d'adjacence	O(|A|+|S|)	O(degré(i))	O(degré(i))
ensembles d'adjacence	O(|A|+|S|)		

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|$^{2)}$	O(1)	O(|S|)
listes d'adjacence	O(|A|+|S|)	O(degré(i))	O(degré(i))
ensembles d'adjacence	O(|A|+|S|)	O(log(degré(i)))	

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	$O(\|S\|^2)$	O(1)	O(|S|)
listes d'adjacence	O(|A|+|S|)	O(degré(i))	O(degré(i))
ensembles d'adjacence	O(|A|+|S|)	O(log(degré(i)))	O(degré(i))

Efficacités des représentations

Représentation	**Espace**	**Tester si *(i,j)* appartient à *A* ?**	**Parcourir les adjacents de *i***
liste des arcs	O(|A|)	O(|A|)	O(|A|)
matrice d'adjacence	O(|S|[2)]	O(1)	O(|S|)
listes d'adjacence	O(|A|+|S|)	O(degré(i))	O(degré(i))
ensembles d'adjacence	O(|A|+|S|)	O(log(degré(i)))	O(degré(i))

En pratique, les graphes sont souvent *sparses*

- listes d'adjacence si le degré des sommets n'est pas trop grand
- ensembles d'adjacence sinon

Parcours de graphe

Parcours de graphe

Comment parcourir un graphe de façon exhaustive ?

Une routine de base en algorithmique des graphes

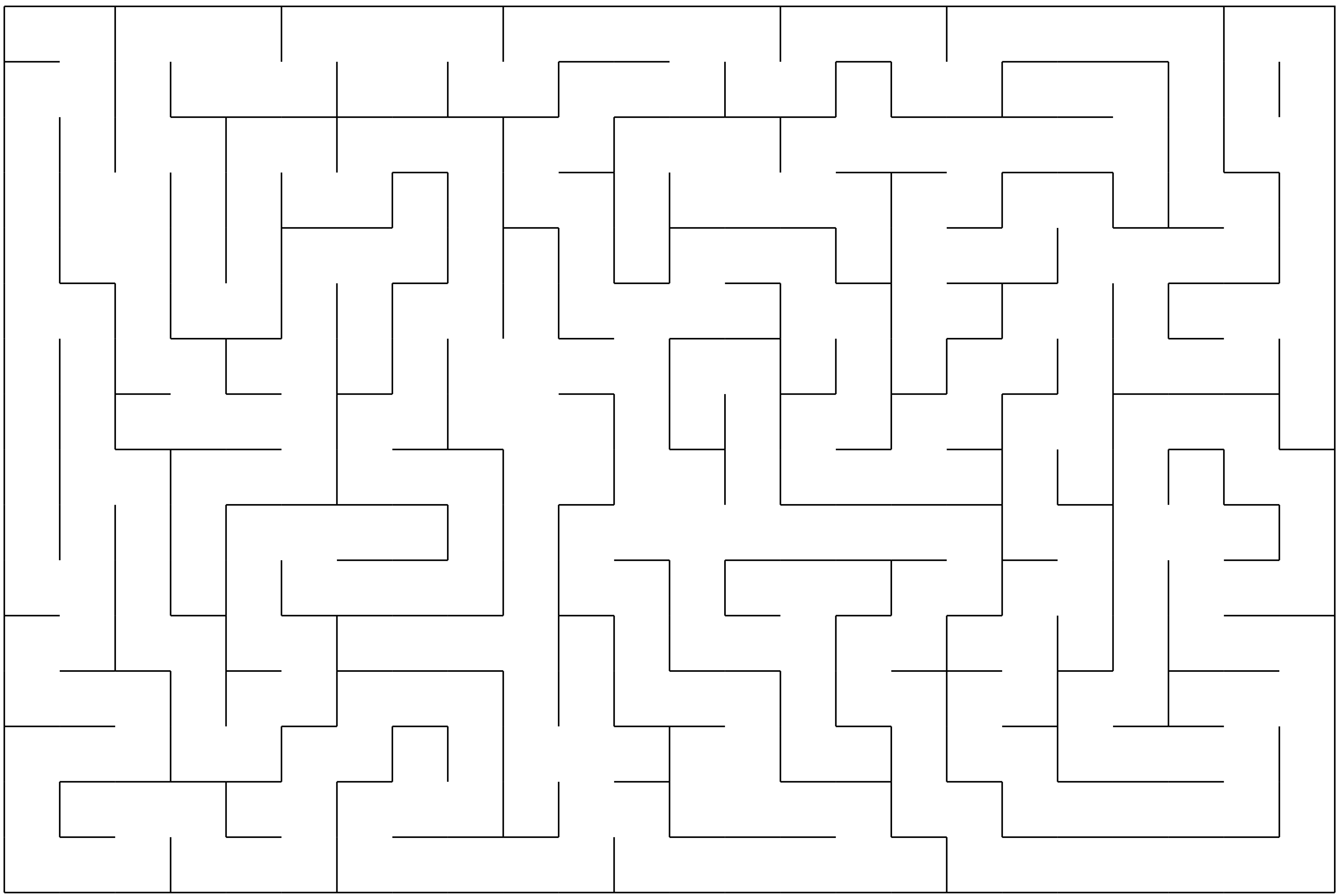
But d’un parcours de graphe

- Explorer tous les sommets
  - en partant d’une origine

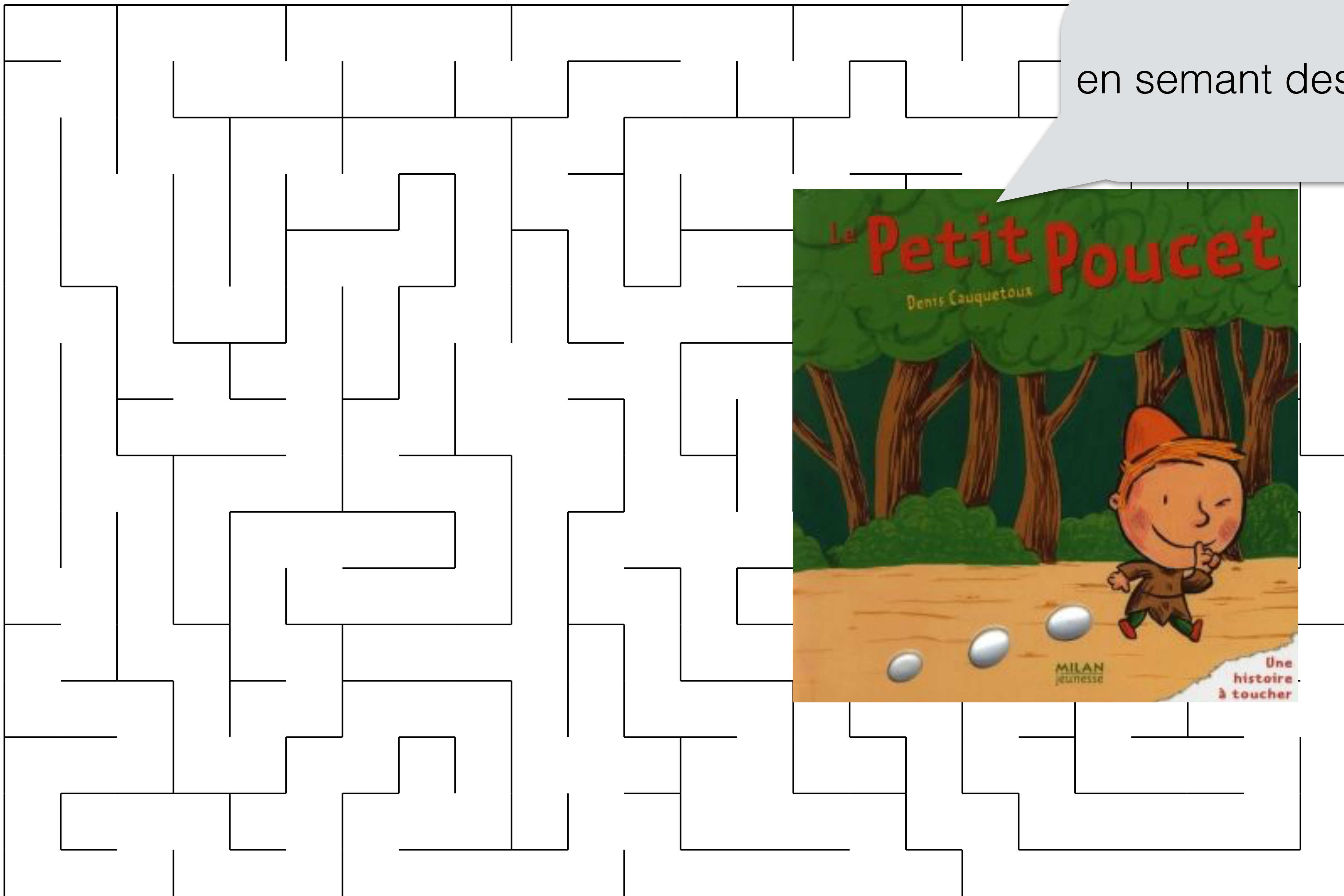
    puis plusieurs si le graphe n’est pas connexe / fortement connexe

  - en suivant uniquement les arcs qui sortent d’un sommet déjà exploré
  - en construisant un historique de son parcours : notion d’arbre *de parcours*

Comment ?



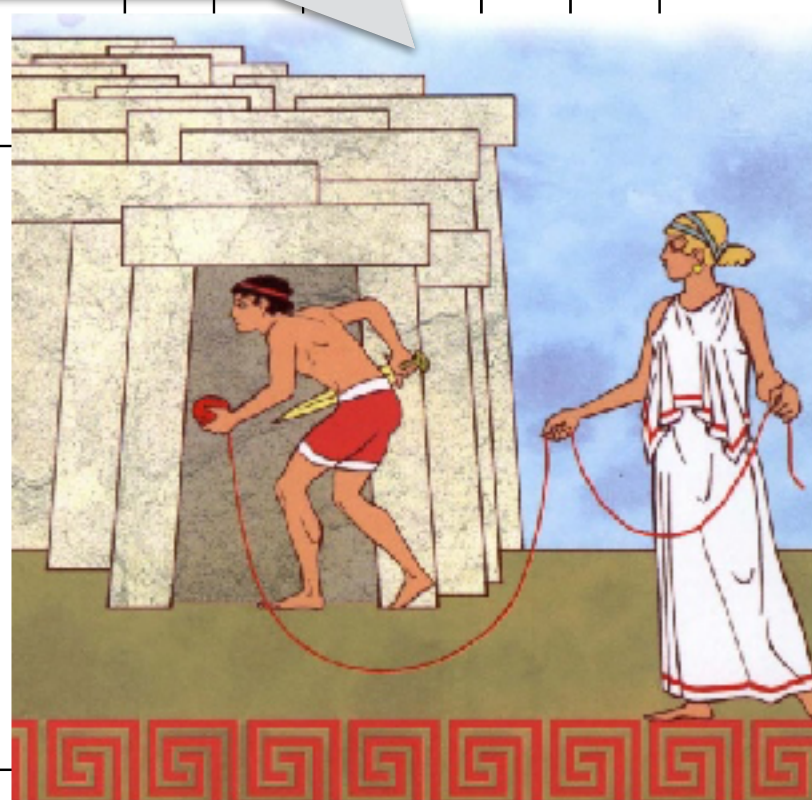
Comment ?



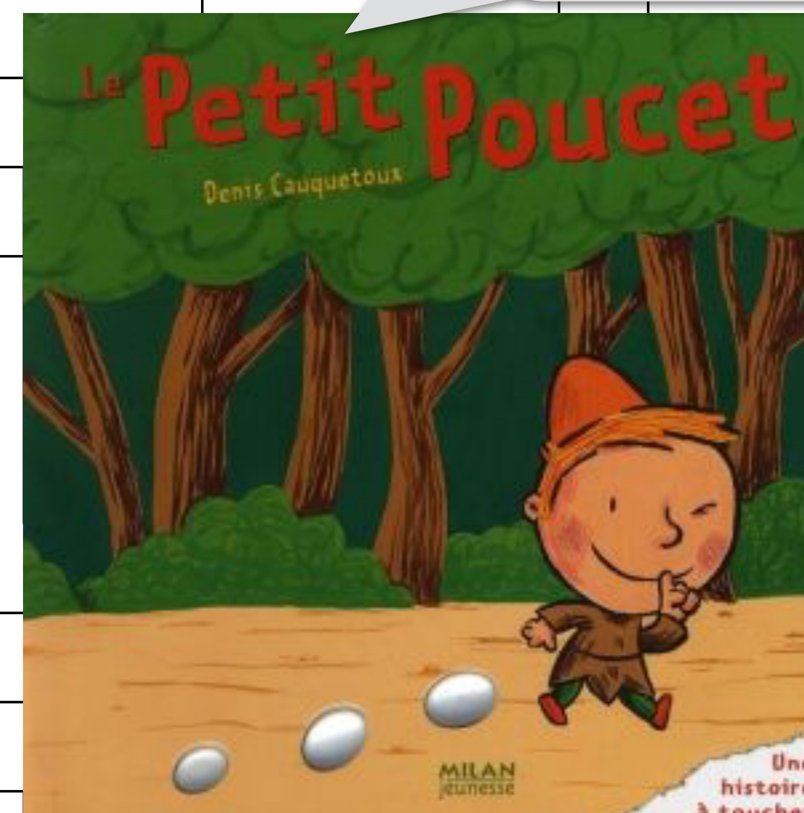
en semant des caillous

Comment ?

en déroulant un fil d’Ariane

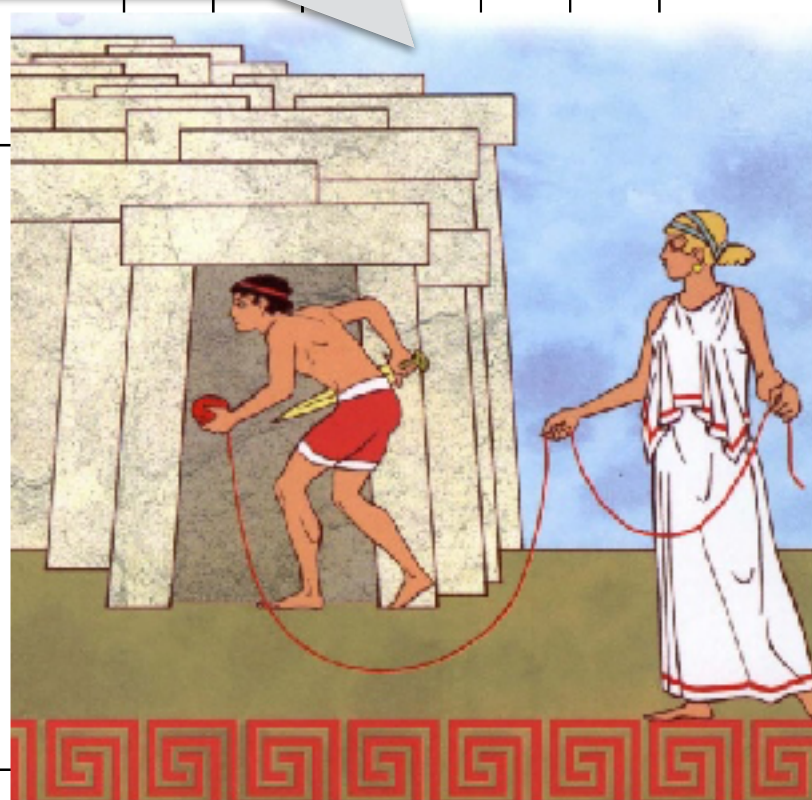


en semant des caillous

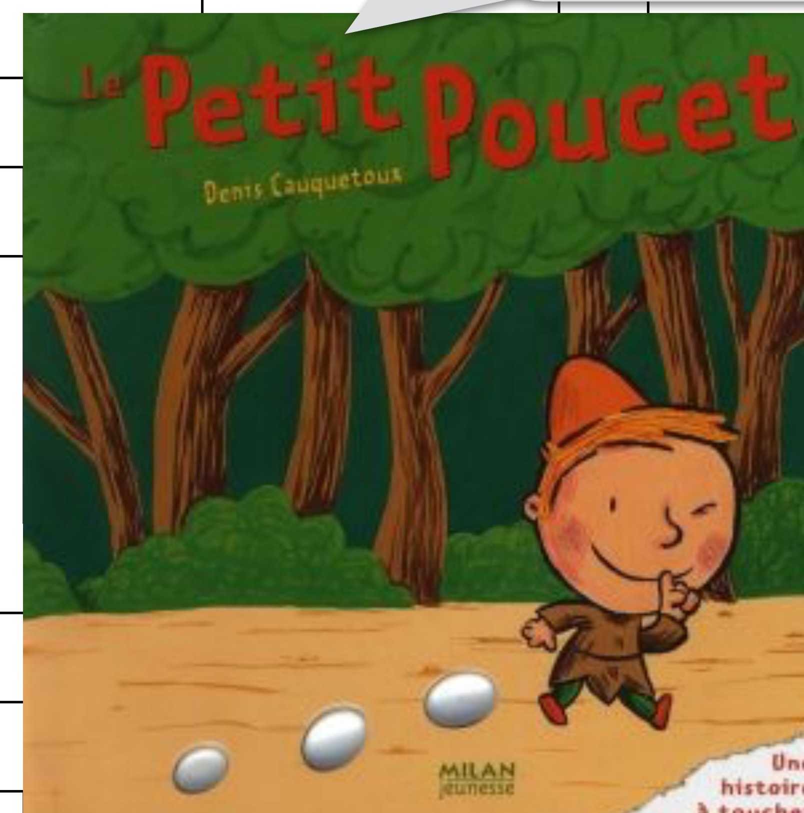


Comment ?

en déroulant un fil d'Ariane



en semant des caillous



on peut ainsi revenir sur ses pas pour trouver la dernière intersection, où il restait des directions à explorer cf. fichier `maze_dfs_rec.pdf`

Parcours en profondeur (procédure recursive)

Version de base

« les cailloux »

- On utilise un tableau `VU` de *N* booléens
    - `VU[i]=vraie` dès que le sommet `i` a été visité
    - on définit une fonction récursive `VISITE(i)` qui va explorer les sommets non vus à partir de `i` (`i` compris)

Version de base

```
VISITE(i) =

  VU[i] <- vraie
  pour tout j ∈ Adj[i]
     si non VU[j] alors VISITE(j)
```

Version avec dates début/fin

- On ajoute des informations pour comptabiliser les temps de passage
  - une variable globale entière `date`, initialisée à `0` et incrémentée à chaque relevé de temps.
  - un tableau `DEBUT` de *N* dates qui contiendra les dates de première arrivée sur chaque sommet (début de `VISITE(i)`)
  - un tableau `FIN` de *N* dates qui contiendra les dates de dernier départ de chaque sommet (fin de `VISITE(i)`)

```
VISITE(i) =

  VU[i] <- vraie
  pour tout j ∈ Adj[i]
     si non VU[j] alors VISITE(j)
```

Exemple

	1	2	3	4	5	6
VU	faux	faux	faux	faux	faux	faux

VISITE(2)

VU	faux	vraie	faux	faux	faux	faux

VISITE(3)

VU	faux	vraie	vraie	faux	faux	faux

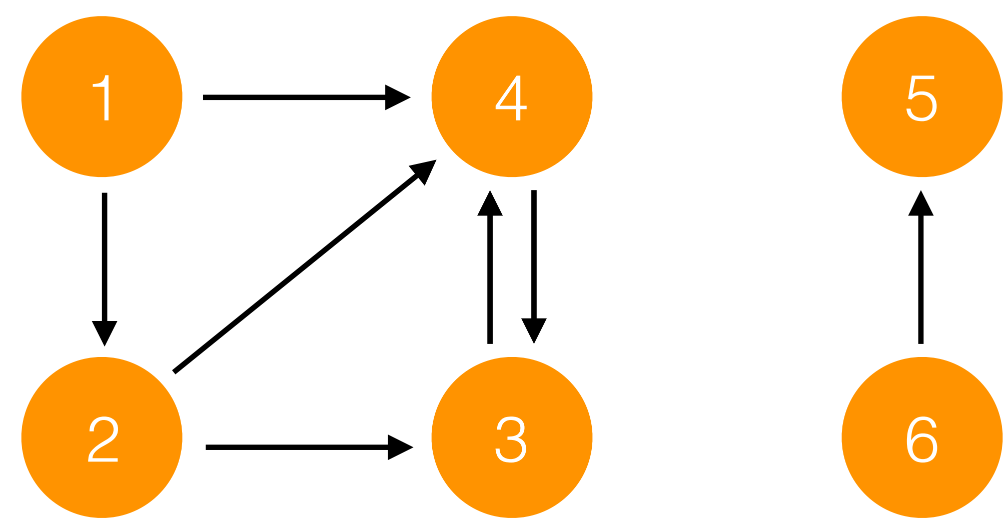
VISITE(4)

VU	faux	vraie	vraie	vraie	faux	faux

On ne (re)visite pas 3

On ne (re)visite pas 4

```
VISITE(i) =
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
```



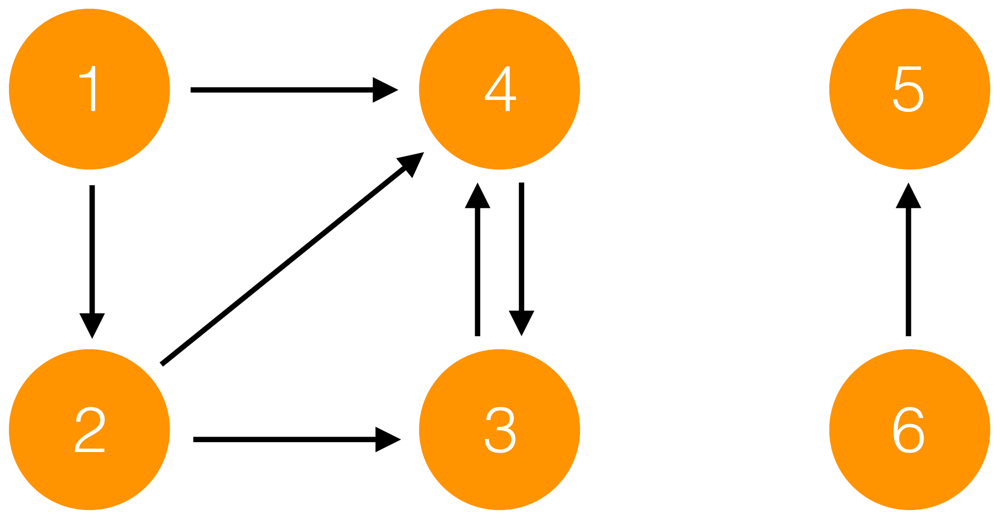
Version avec dates début/fin

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∈ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

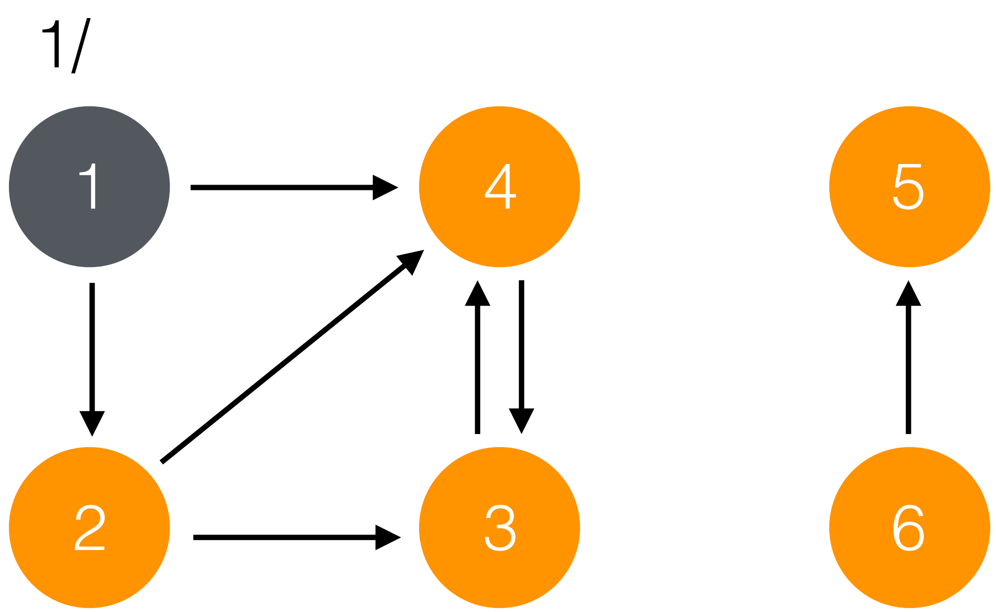


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

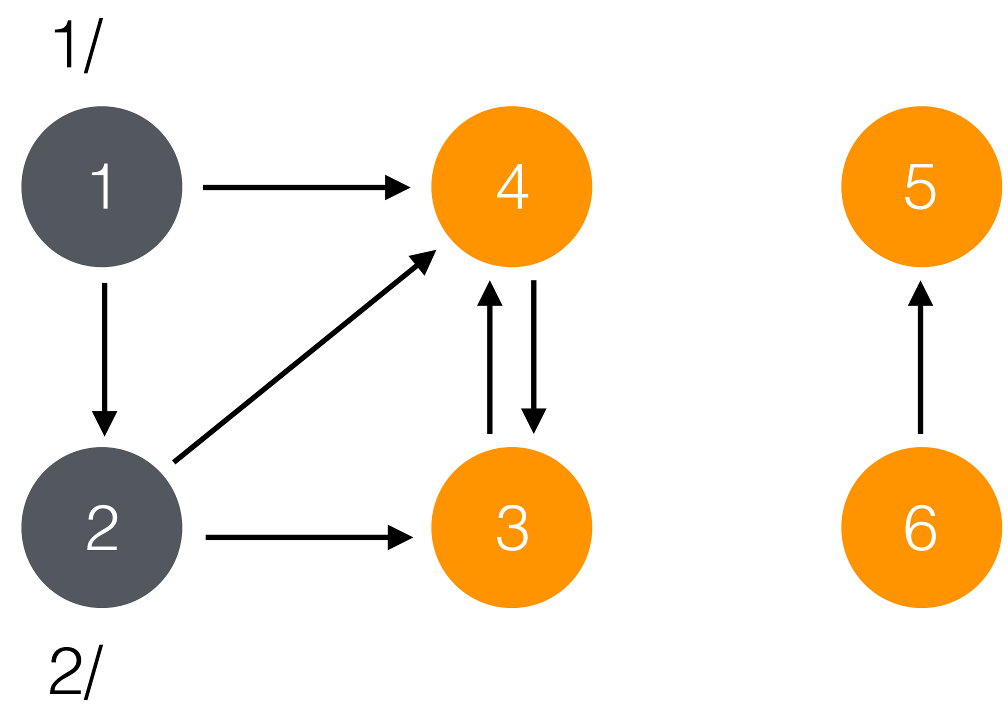


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

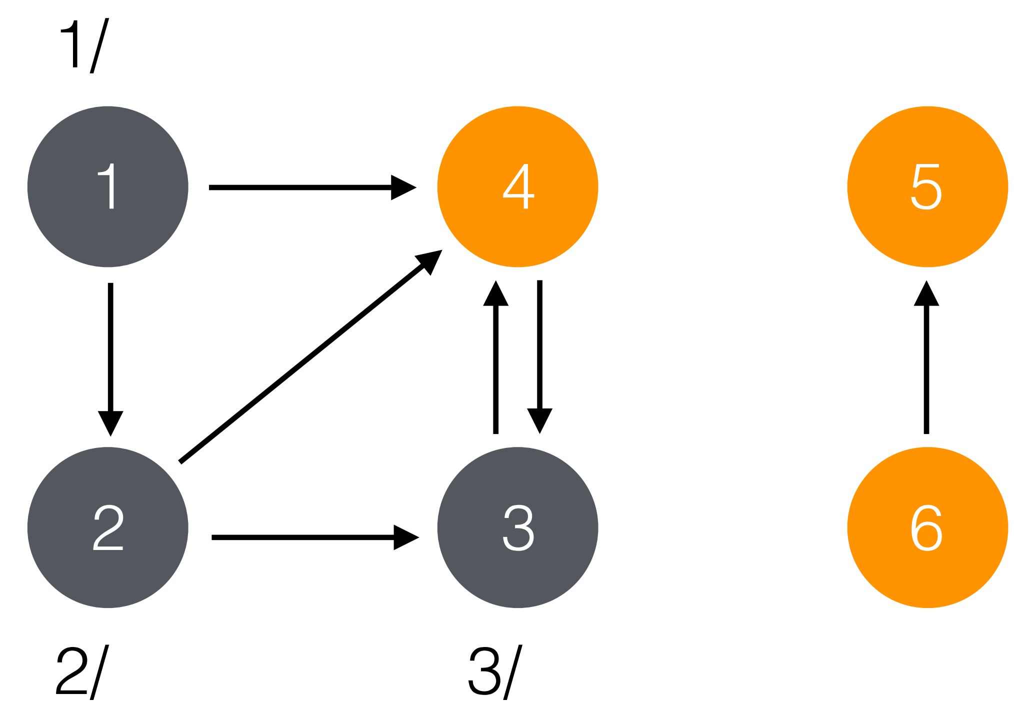


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

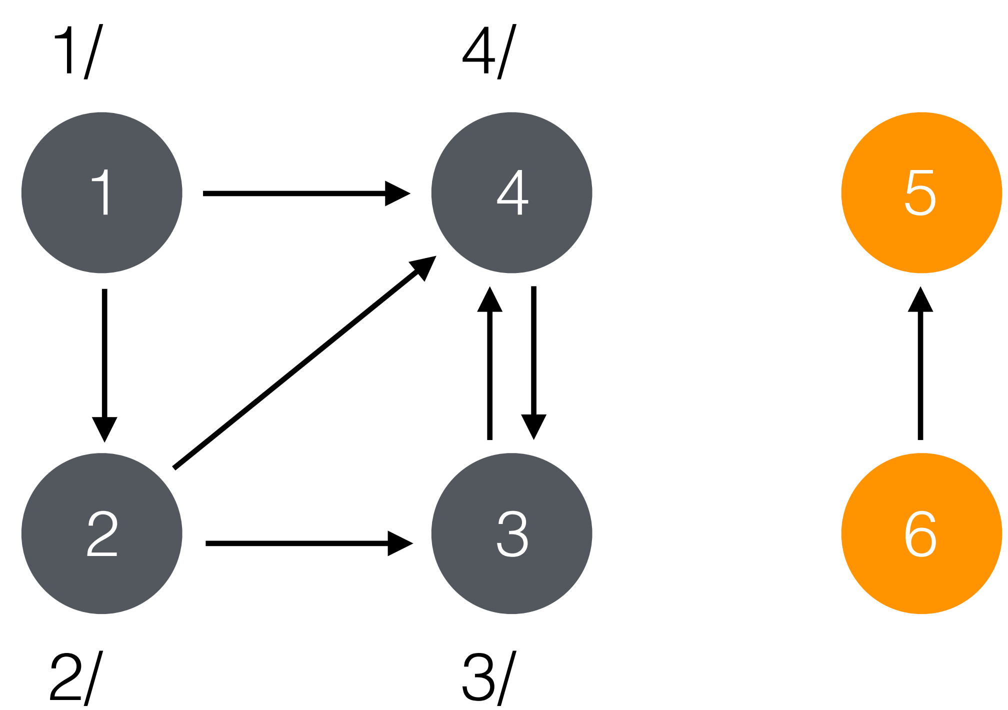


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

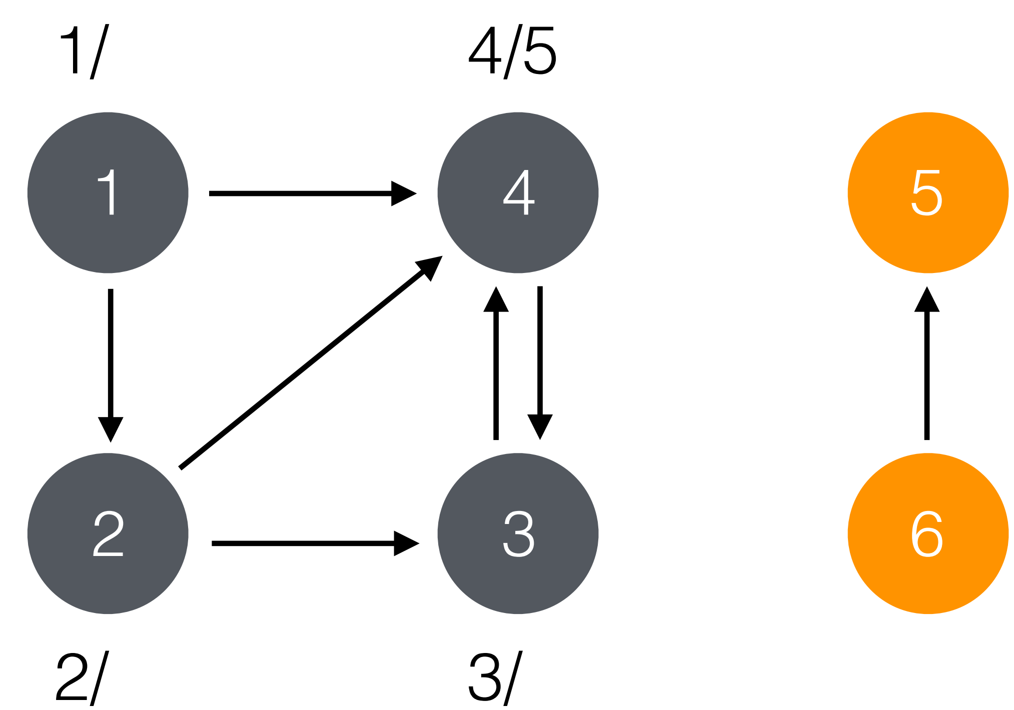


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

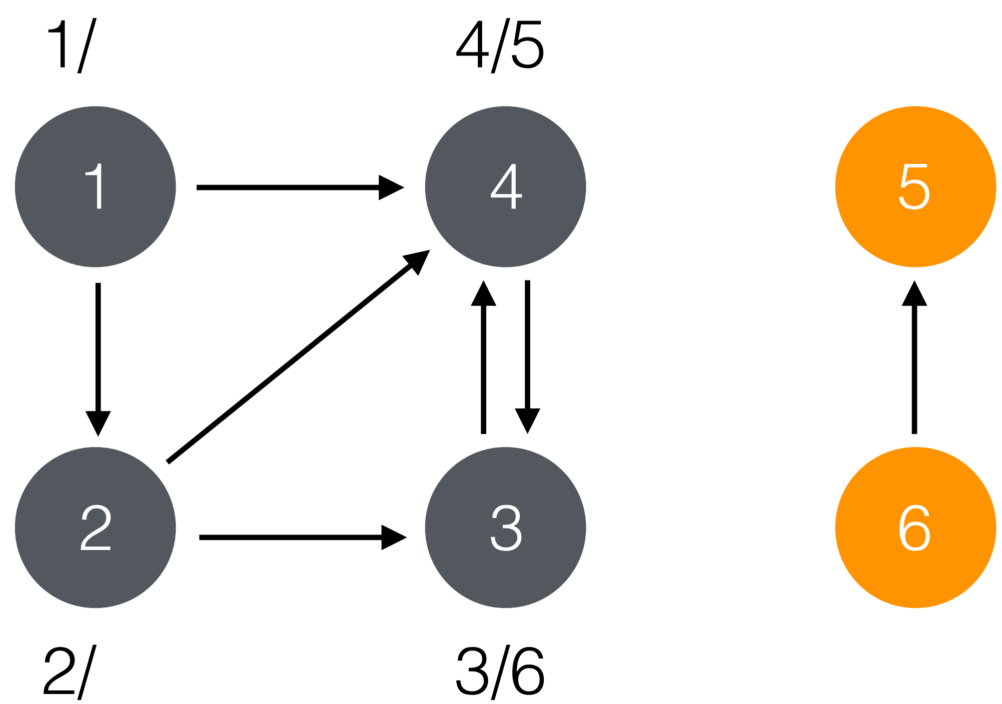


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∈ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

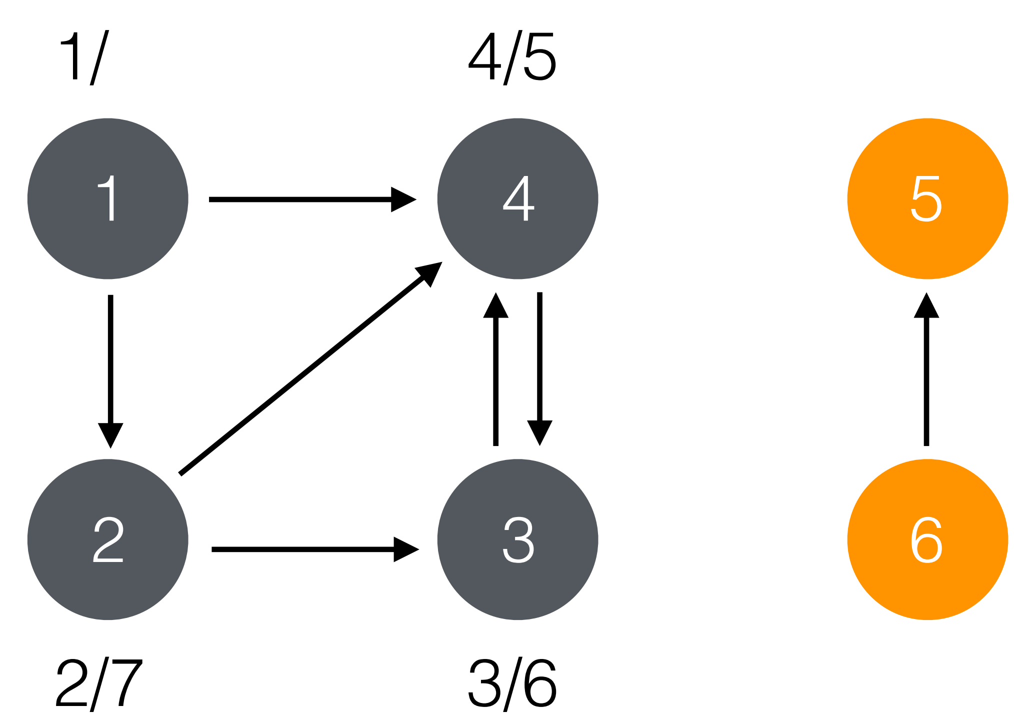


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```

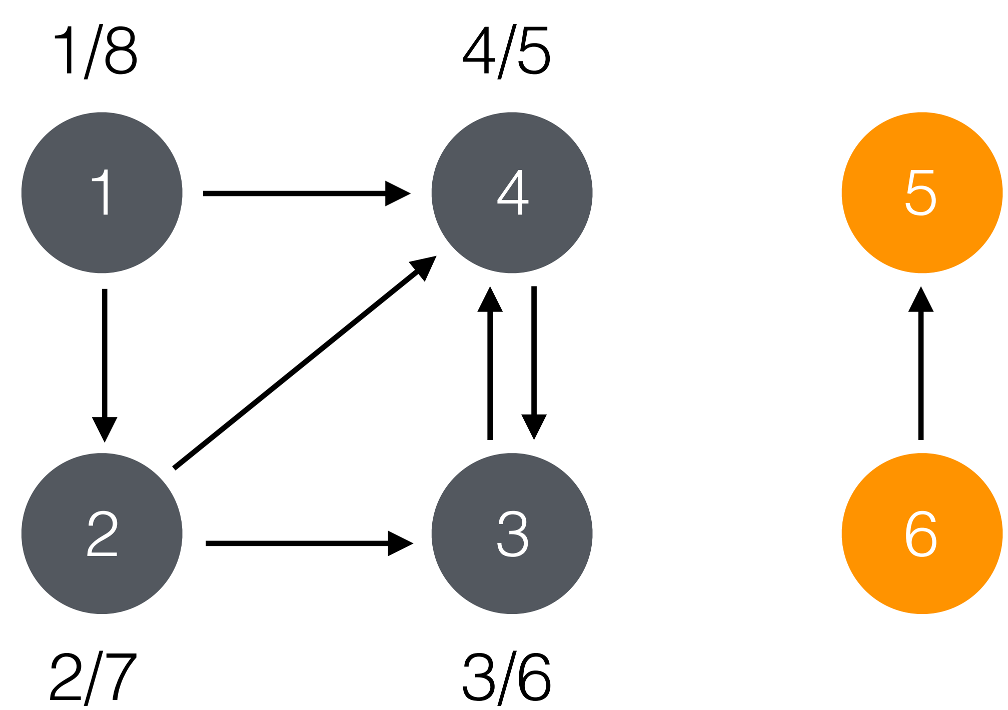


Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∊ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date
```



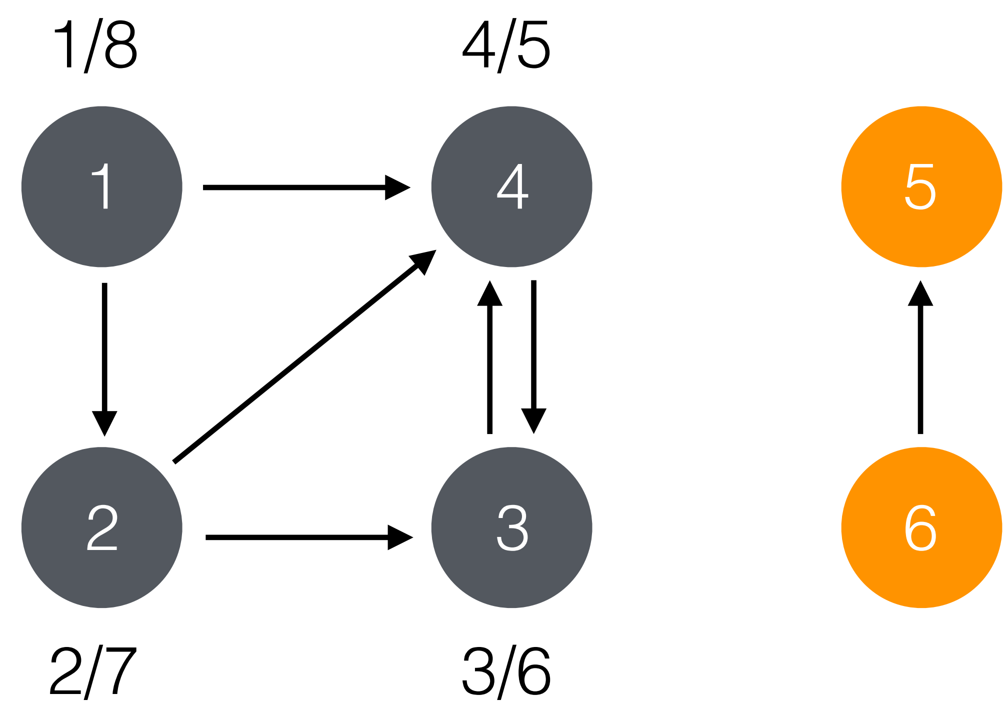
Sur chaque sommet `i`, nous notons `DEBUT[i]/FIN[i]` quand ils sont calculés

Exemple

Représentons le calcul de VISITE(1)

Le calcul est terminé mais nous n'avons pas visité les sommets 5 et 6 !

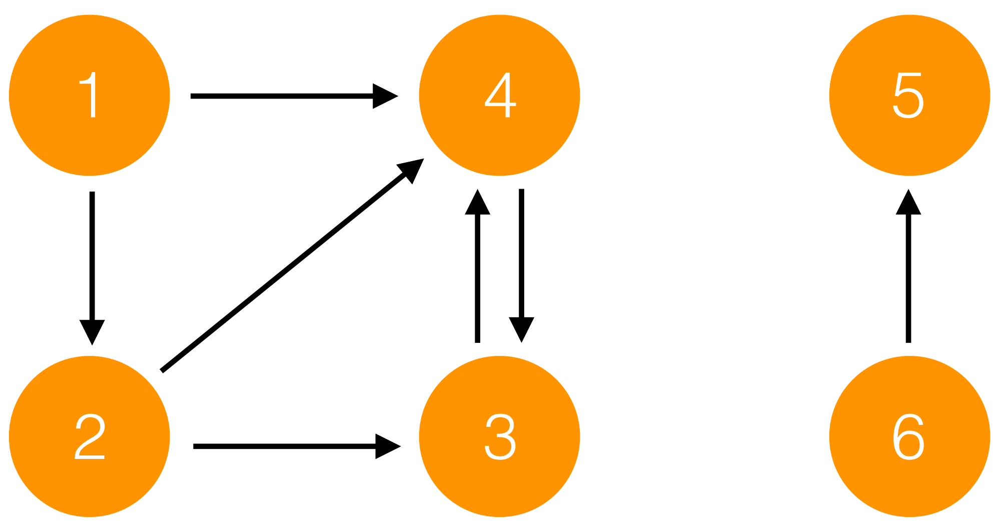
Nous allons ajouter une boucle principale pour s'assurer que l'on visite tous les sommets



Parcours en profondeur (récursif)

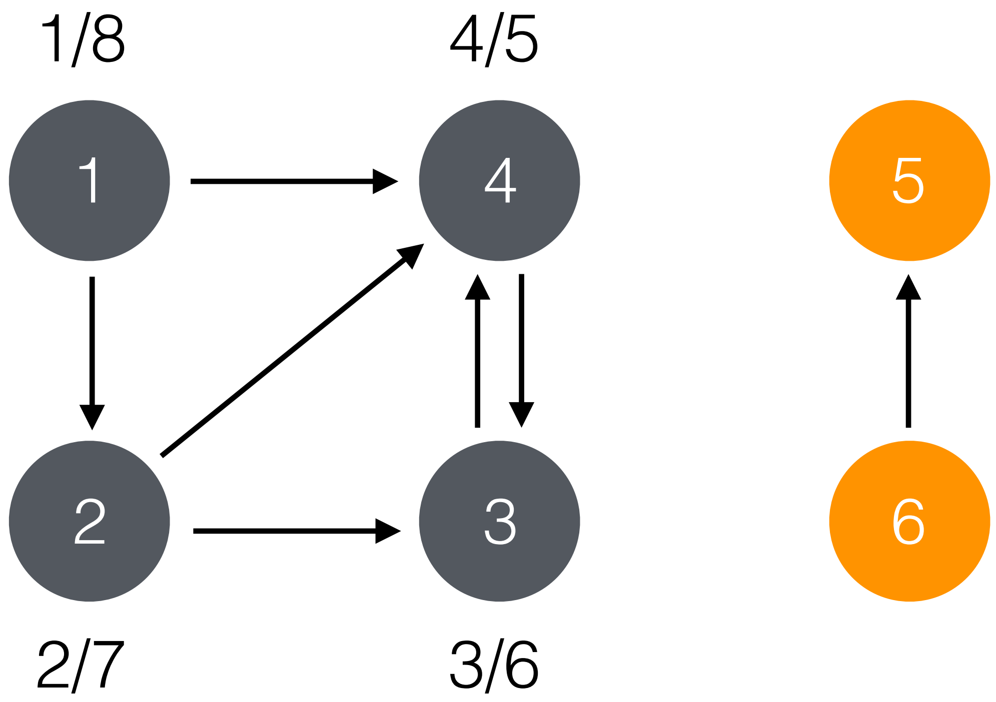
```
VISITE(i) =
  date <- date + 1
  DEBUT[i] <- date
  VU[i] <- vraie
  pour tout j ∈ Adj[i]
     si non VU[j] alors VISITE(j)
  date <- date + 1
  FIN[i] <- date

VU <- [faux, …, faux]
date <- 0
pour tout i ∈ S
   si non VU[i] alors VISITE(i)
```

Exemple



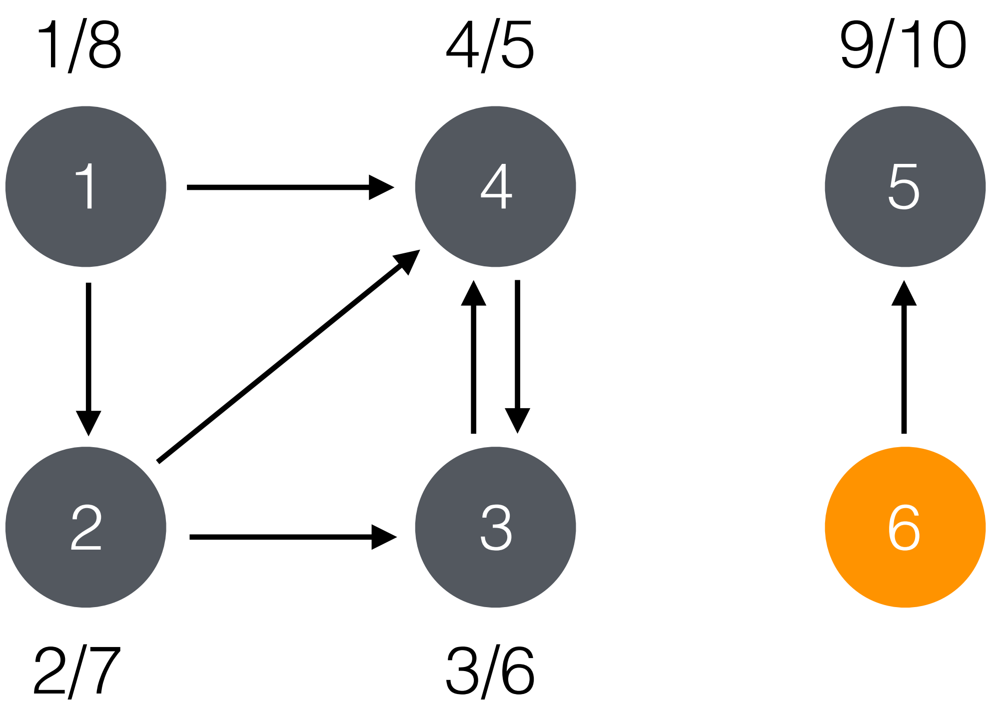
Exemple

- VISITE(1)



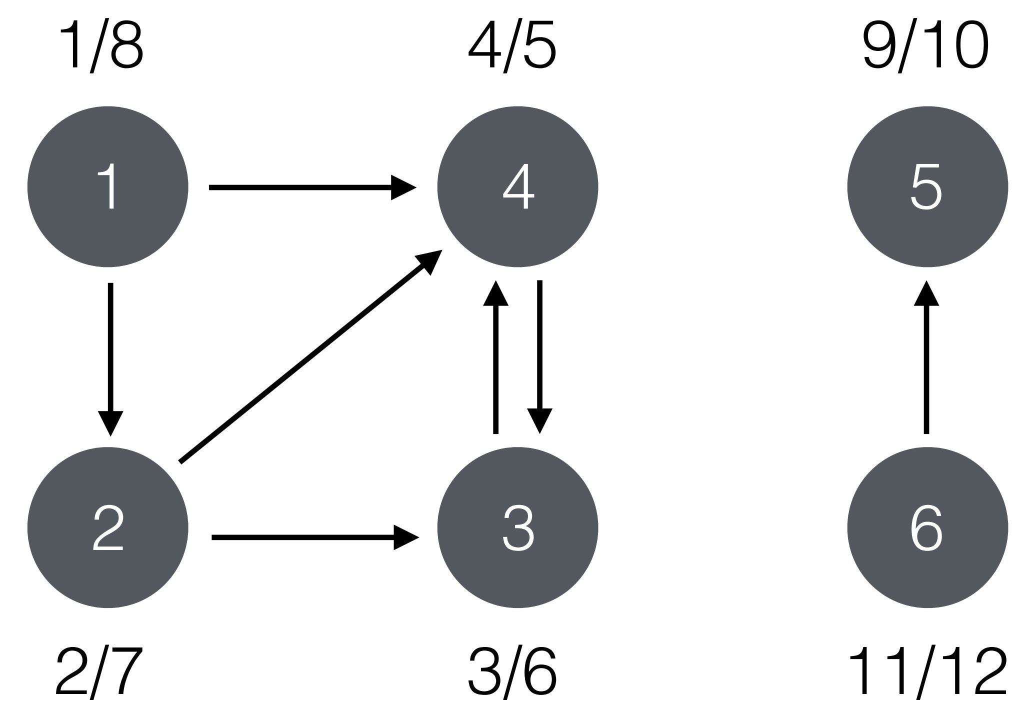
Exemple

- VISITE(1)
- puis VISITE(5)



Exemple

- VISITE(1)
- puis VISITE(5)
- puis VISITE(6)



Conventions importantes

- Pour dérouler ces algorithmes (TD, quizz, devoirs), nous devons nous mettre d'accord sur des ordres d'itérations

```
pour tout j ∈ Adj[i]
```

on parcourt les adjacents par ordre croissant

dans ce cours, les sommets sont des entiers !

```
pour tout i ∈ S
```

on parcourt les sommets par ordre croissant

Prochain cours

- Jeudi prochain 8:00
- Quizz1 à faire sur moodle avant !