ALGORITHMIQUE

INTRODUCTION

Laboratoire Informatique Image Interaction (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

ENSEIGNANTS

Jean-Loup Guillaume

- Responsable de l'UE
- Bureau 107bis
- Plutôt les cours sur les graphes

Karell Bertet

- Plutôt les cours sur la complexité

Antoine Huchet

- Interventions surtout en TD

OBJECTIFS DU COURS – ALGORITHMIQUE AVANCÉE

Deux sous objectifs

Algorithmique des graphes

- Cours 1 : Définition et parcours (largeur, profondeur, tri topo)
- Cours 2 : Algos sur les chemins (MST, PCC)
- Cours 3 : Flots et algorithmes d'approximation sur les graphes

Complexité et structures de données

- Cours 1 : Calcul de complexité (algo récursif)
- Cours 2 : Complexité et structures de données
- Cours 3 : Classes de complexité et décidabilité
- Cours 4 : Heuristiques et approximation

PROGRAMME

Cours magistraux (vendredi apm)

- 7 séances (4 sur la complexité, 3 sur les graphes)

TD (mardi matin)

- 5 séances (3 sur la complexité, 2 sur les graphes)

TP + TEA

- TP de prise en main et projet lié au cours
- Les créneaux indiqués dans l'edt sont facultatifs (pas le travail)

NOTATION

Examen final : 40%

Exercices (en TD ou à la maison) : 20%

Projet : 40%

Temps de travail estimé :

- 3 crédits ECTS => au moins 75h de travail
- 22h30 de travail avec enseignants => reste 50h environ
- Refaire exercices de TD + exercices à la maison + révisions + projet

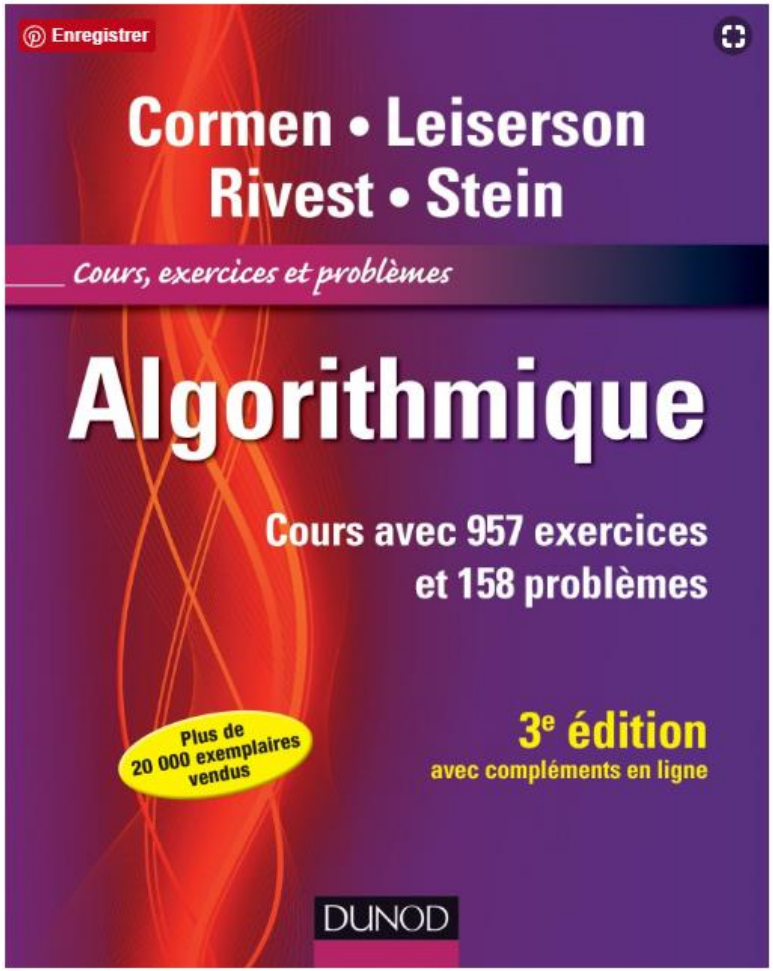
RÉFÉRENCE(S)

Algorithmique - 3ème édition

Cours avec 957 exercices et 158 problèmes Broché

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein

Cours 1 = chapitre 22



ALGORITHMIQUE

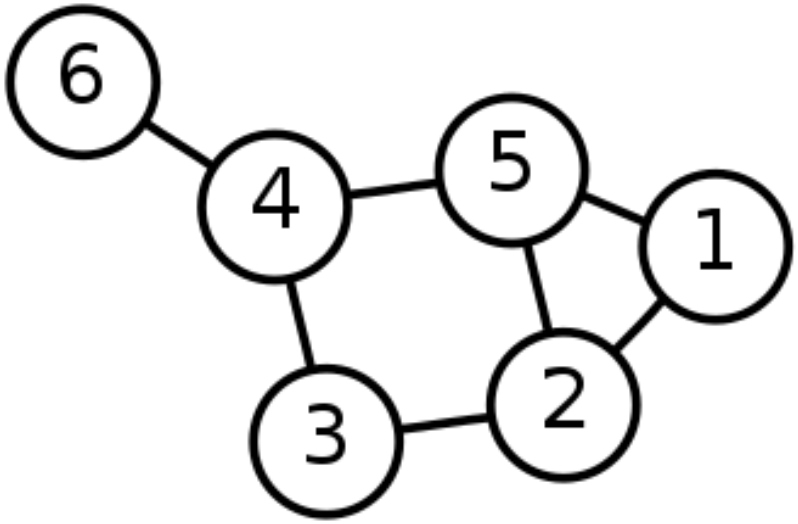
COURS 1 – CONCEPTS DE BASE SUR LES GRAPHES

Laboratoire Informatique Image Interaction (L3I)
Université de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 France
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

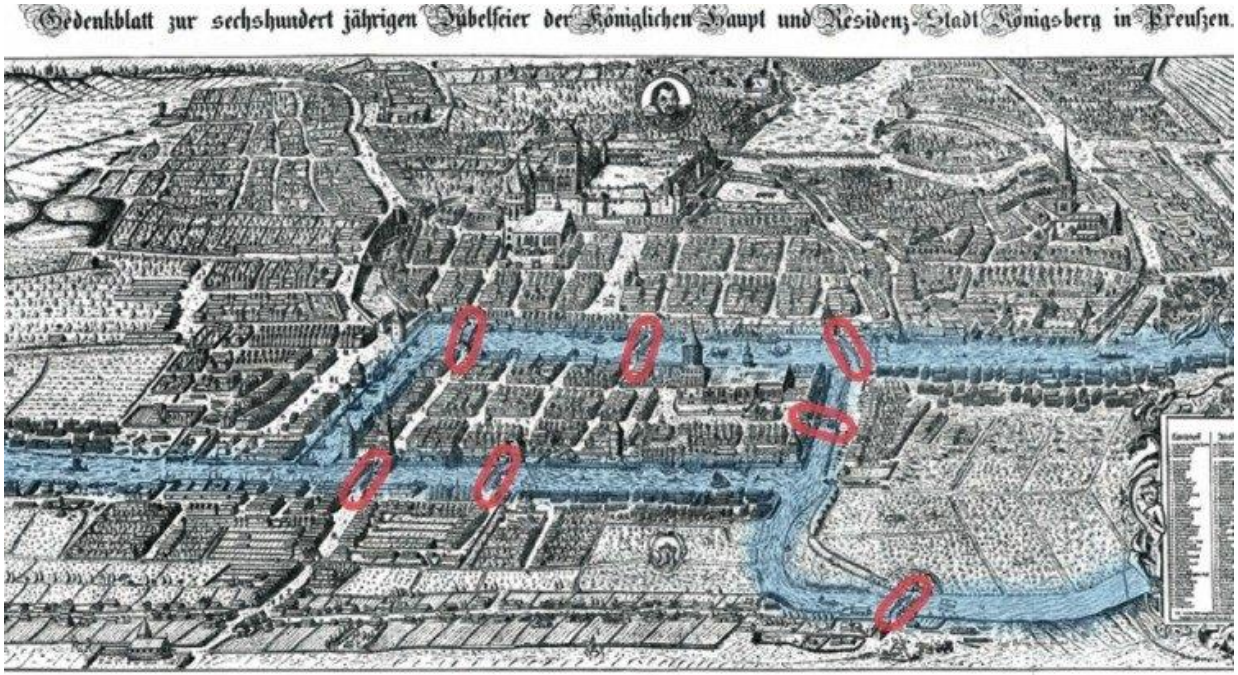
QU'EST-CE QU'UN GRAPHE ?

Un ensemble de sommets connectés par des arêtes

- Un sommet représente un objet
- Les arêtes représentent des interactions/connections entre ces objets



VILLE DE KÖNIGSBERG



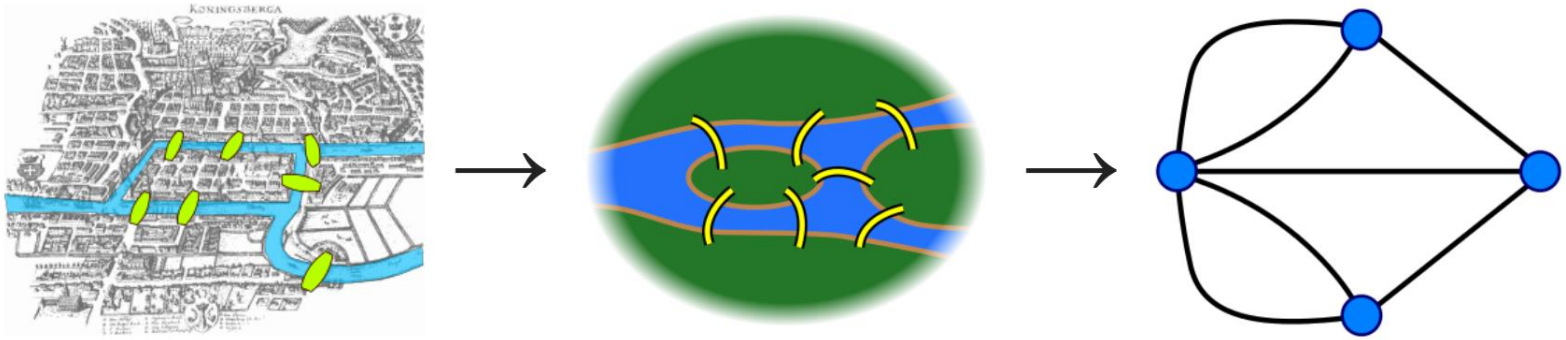
Problème : est-il possible de traverser tous les ponts une seule fois et de revenir à son point de départ ?

MODÉLISATION

Problème : est-il possible de traverser tous les ponts une seule fois et de revenir à son point de départ ?

Modélisation de la ville par un graphe (Leonhard Euler - 1736)

- Les sommets représentent les zones de la ville
- Les arêtes représentent les ponts



Preuve formelle en 1873 (Karl Hierholzer)

- Condition nécessaire et suffisante ?

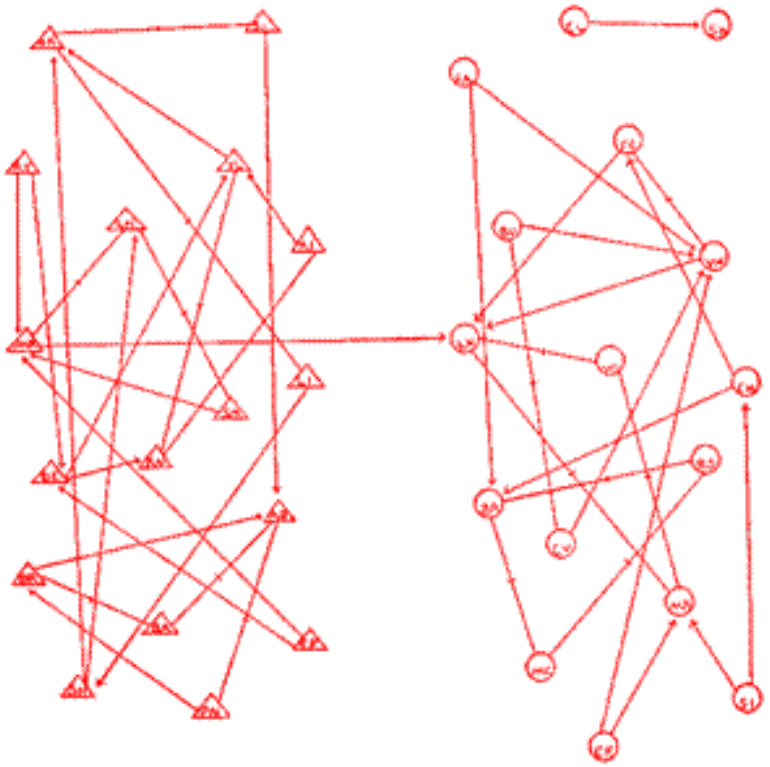
SOCIOLOGIE

Etude des élèves d'une crèche et d'une école primaire (Moreno - 1933)

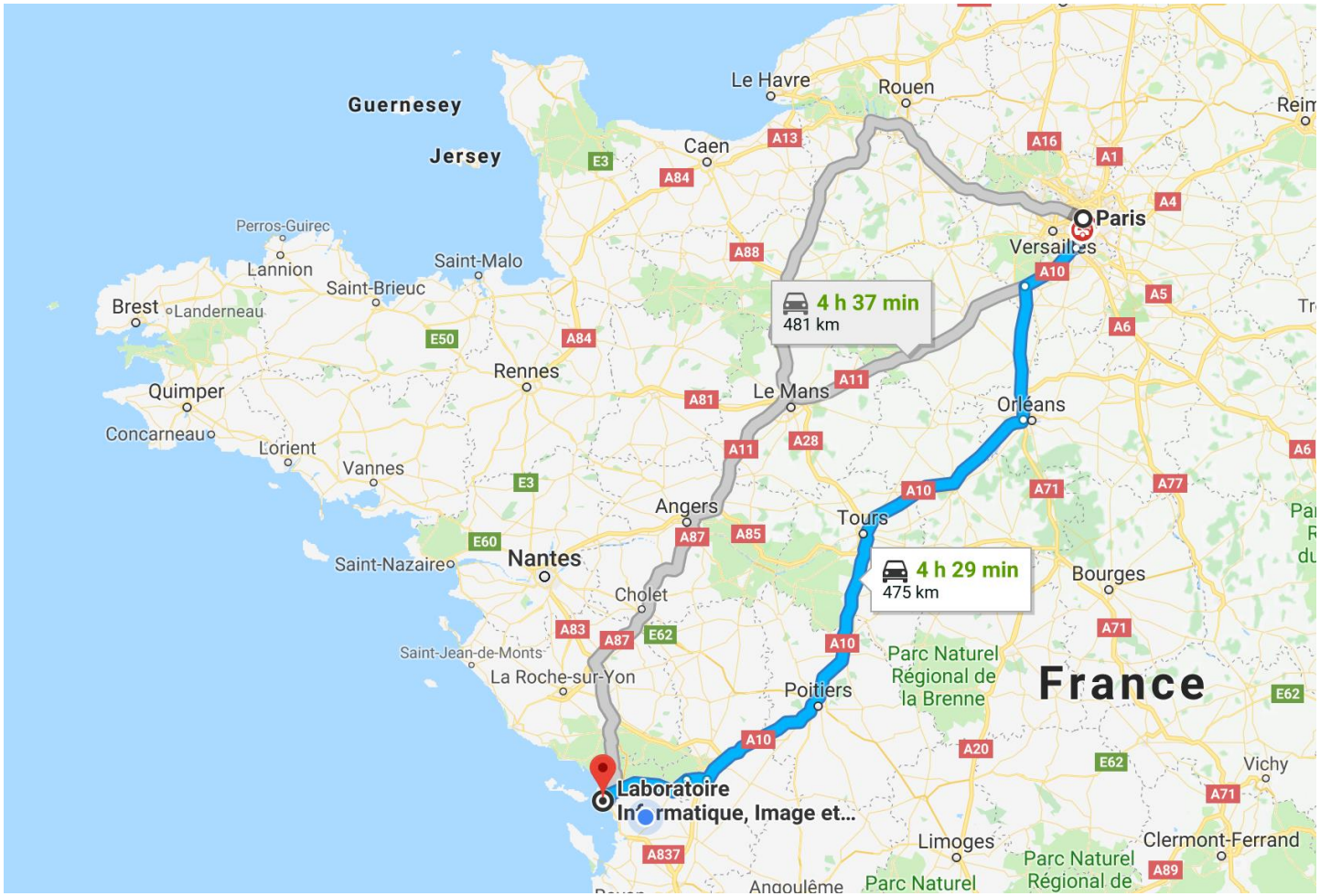
EMOTIONS MAPPED BY NEW GEOGRAPHY

Charts Seek to Portray the Psychological Currents of Human Relationships.

New York Times
April 3, 1933



CALCUL DE TRAJETS



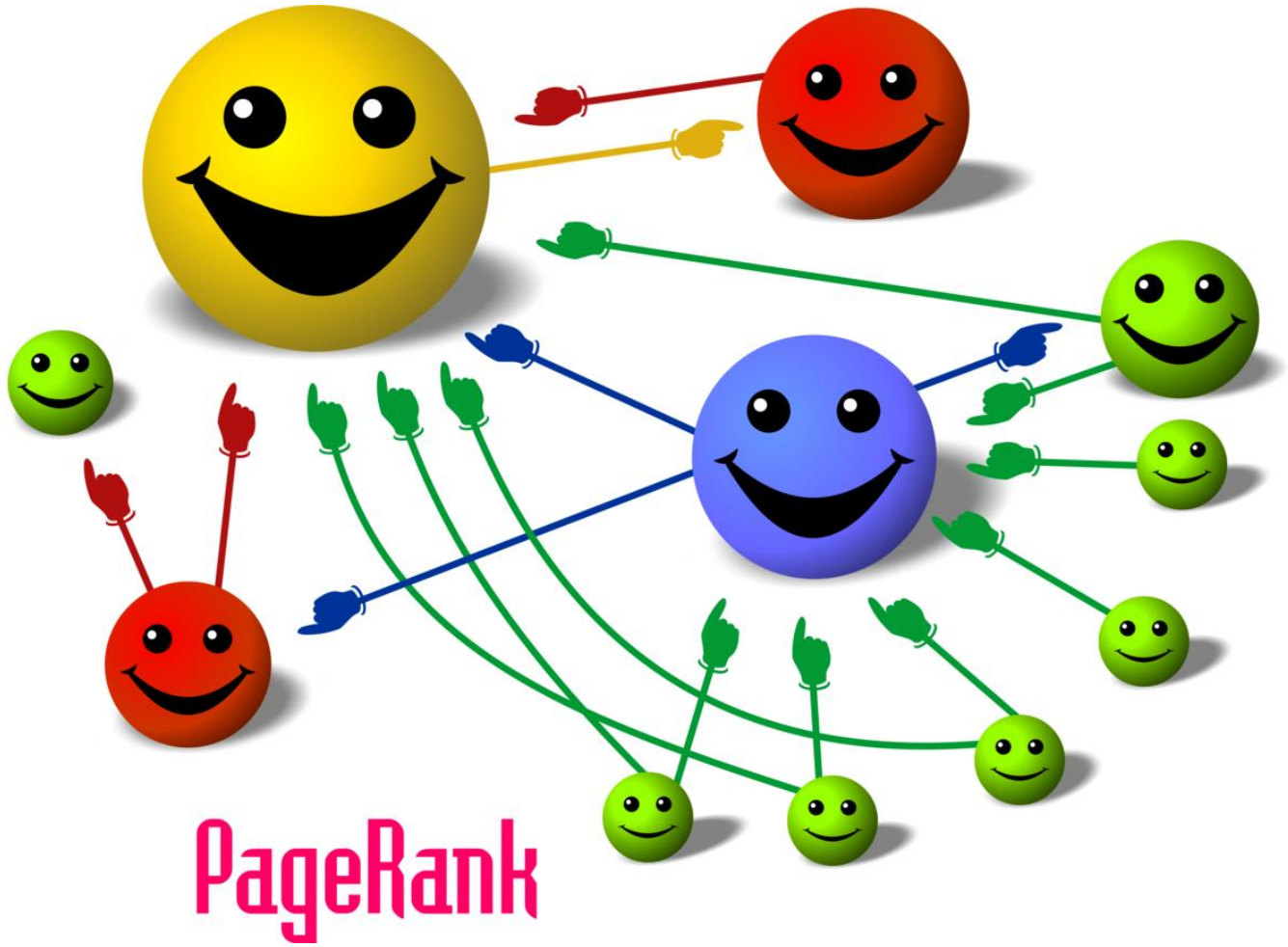
CALCUL DE ROUTES

Modélisation du réseau routier par un graphe

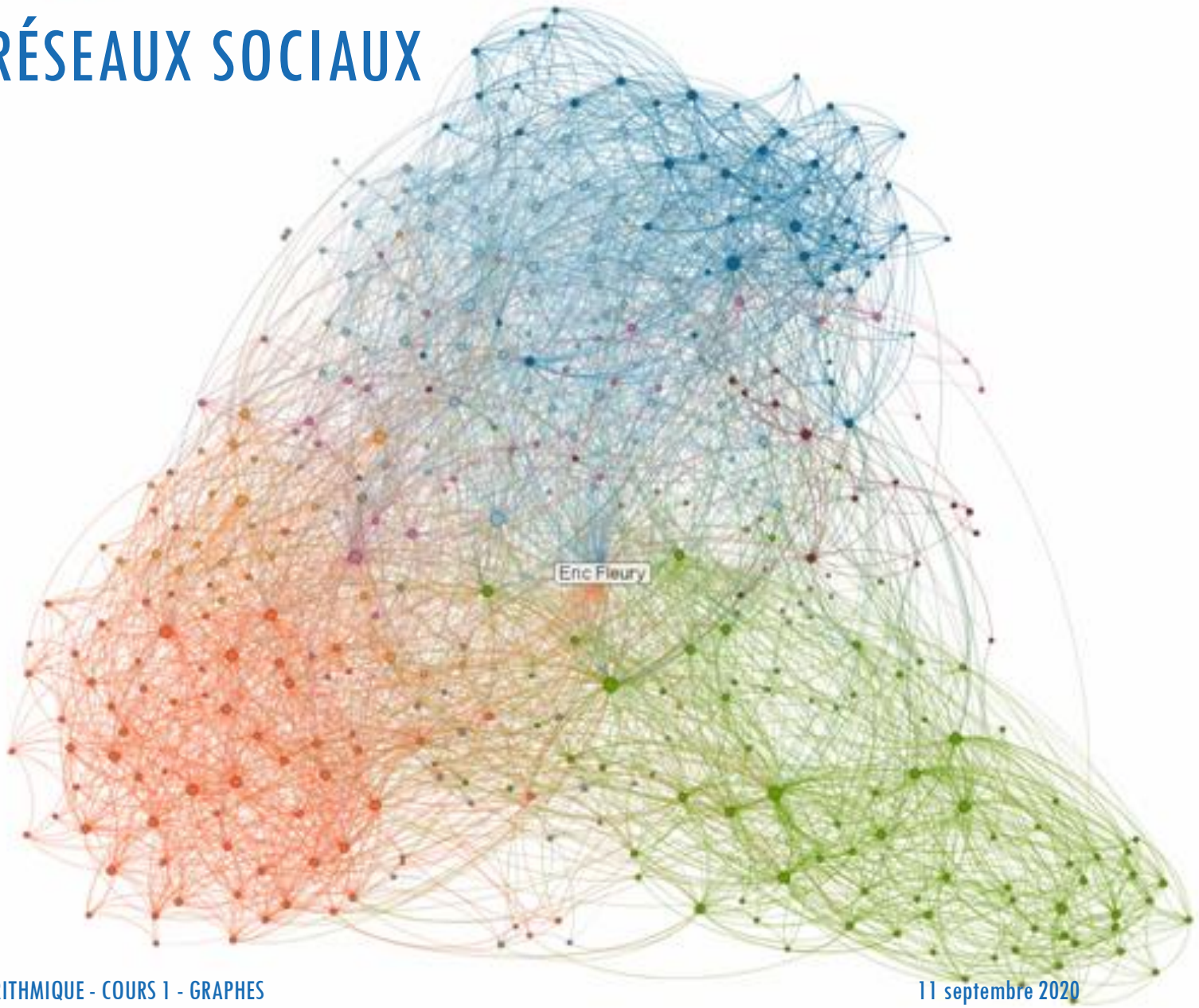
- Les sommets représentent les intersections
- Les arêtes représentent les routes

Qu'est-ce qu'un plus court chemin dans ce graphe, comment le calculer ?

GOOGLE ET LE PAGERANK



RÉSEAUX SOCIAUX



BEAUCOUP D'AUTRES RÉSEAUX

Informatique : internet, web, usages (P2P), …

Sciences sociales : collaboration, amitié, contacts sexuels, échanges, économie, …

Biologie : cerveau, gènes, protéines, écosystèmes, …

Linguistique : synonymie, co-occurrence, …

Transport : routier, aérien, électrique, …

etc.

Dans tous ces cas les graphes modélisent naturellement une situation réelle

DE MANIÈRE PLUS GÉNÉRALE

On modélise des données par des graphes quand

- Il y a une relation dans l'espace des objets
- Il y a des interactions, des relations, des conflits entre des personnes/objets
- Il y a un ordre temporel entre des objets, des événements
- Il y a une hiérarchie entre les objets

Mais pas seulement

DÉFINITIONS

Laboratoire Informatique Image Interaction (L3I)
sité de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 Fra
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

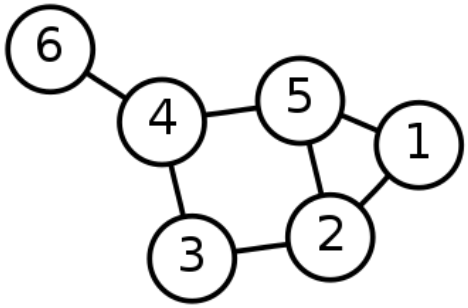
GRAPHE – SOMMET - ARÊTE

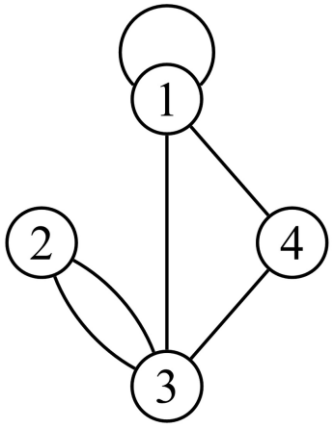
Un **graphe** G=(V,E) est constitué

- D'un ensemble $V = \{v_1, v_2, \ldots, v_n\}$ de **sommets**
- D'un ensemble $E = \{e_1, e2, \ldots, e_k\}$ d'**arêtes**
- Chaque arête est associée à deux sommets, ses **extrémités**

Un graphe est **simple** s'il est sans boucle est sans arêtes multiples

- Une **boucle** est une arête qui va d'un sommet à lui-même
- Un graphe a des **arêtes multiples** si des sommets peuvent être connectés plusieurs fois

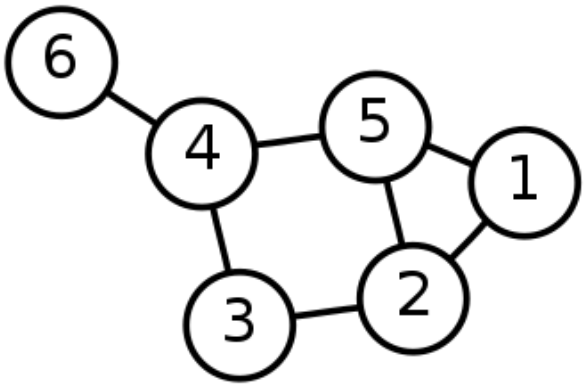


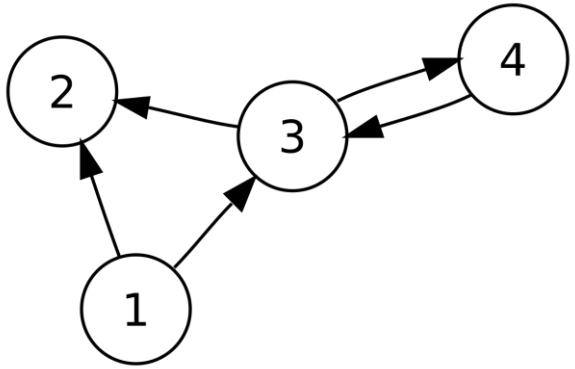


ORIENTATION

Un graphe peut être orienté ou non

- S'il est orienté, les **arcs** ont un sens, ce sont des couples de sommets
  - On distingue alors la **source** et l'**extrémité** des arcs
  - On parle également de **prédécesseur** et de **successeur** pour les sommets
- Sinon, les **arêtes** n'ont pas de sens, ce sont des paires de sommets





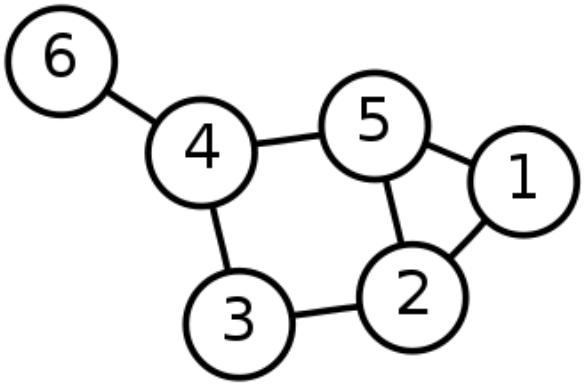
ADJACENCE ET DEGRÉ

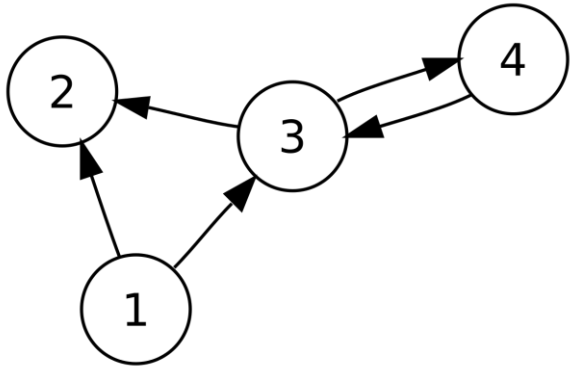
Deux sommets sont dits **adjacents** s'il y a une arête entre eux

- Si u et v sont adjacents on dit aussi qu'ils sont **voisins**
- Pour un graphe orienté u est adjacent à v si u est l'extrémité de l'arc

Le degré d'un sommet u est son nombre de voisins

- Pour un graphe orienté on distingue le **degré entrant** (u est l'extrémité) et le **degré sortant** (u est la source)

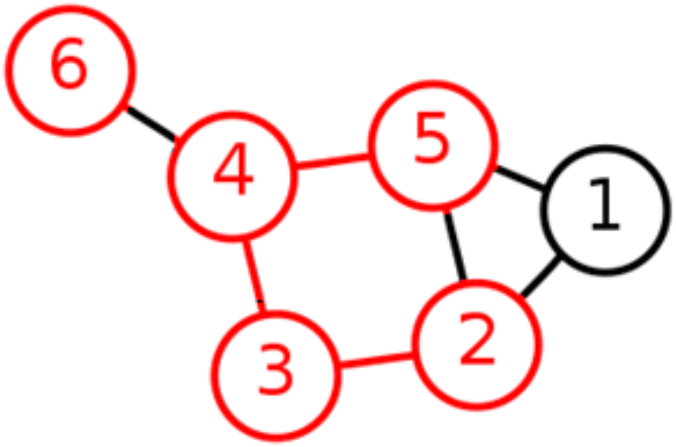


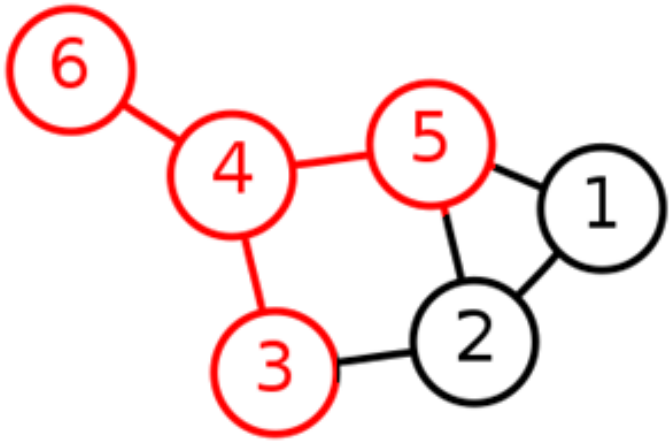


SOUS-GRAPHE (INDUIT)

Un **sous-graphe** G'=(V',E') de G=(V,E) est tel que V' et E' sont des sous-ensembles de V et E et les arêtes de E' ont leurs extrémités dans V'

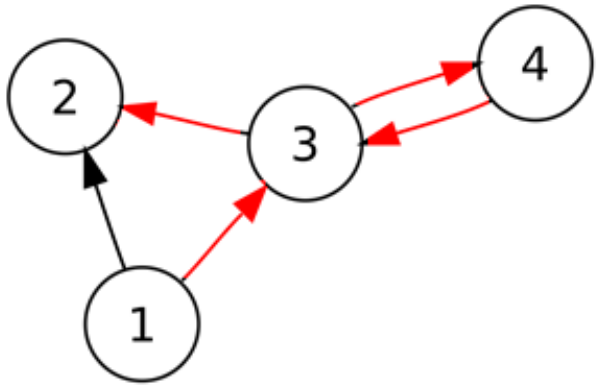
- Un sous-graphe est dit **induit par un ensemble de sommets V'** s'il contient toutes les arêtes de E entre des sommets de V'





CHEMINS

Un **chemin** entre u et v est une suite d'arcs (l'origine de chaque arc est l'extrémité du précédent) allant de u à v



La **longueur** d'un chemin est le nombre d'arcs qui le composent

CHEMINS ET PLUS COURTS CHEMINS

On définit également :

- Un chemin est **simple** si tous les arcs sont distincts
- Un chemin est **élémentaire** si tous les sommets sont distincts
- Un chemin qui commence et termine sur le même sommet est un **circuit**

Dans un graphe non orienté on parle de chaine et de cycle

Un **plus court chemin** est un chemin de longueur minimale

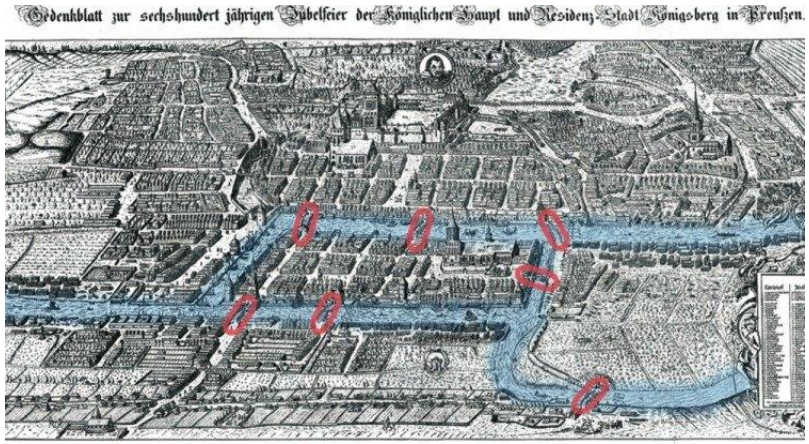
- La **distance** entre u et v est la longueur d'un plus court chemin entre eux (non définie ou ∞ s'il n'existe pas de chemin)
- Il peut y avoir plusieurs plus courts chemins entre deux sommets

Le **diamètre** d'un graphe (connexe) est la longueur du plus long plus court chemin

EULER ET HAMILTON

Le problème d'Euler est de trouver une chaine qui passe par toutes les arêtes une et une seule fois et revient au point de départ

- C'est donc un cycle simple utilisant toutes les arêtes
- Aussi appelé **cycle Eulérien**

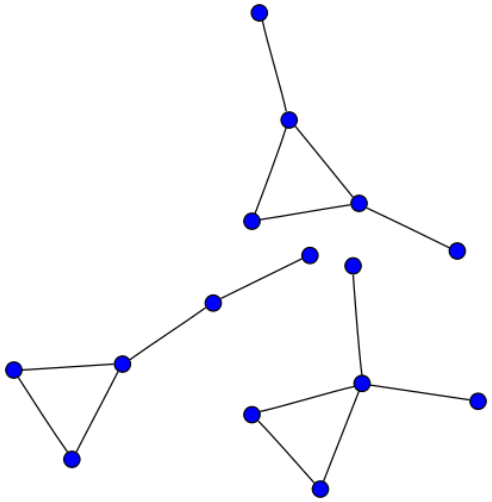


Un cycle qui passe par tous les sommets une et une seule fois est un **cycle Hamiltonien**

CONNEXITÉ

Un graphe est **connexe** s'il existe une chaîne entre toute paire de sommets

- Si un graphe n'est pas connexe, chacun des sous-graphes induit maximaux connexes est appelé une **composante connexe**



Pour un graphe orienté de parle de forte connexité et de composantes fortement connexes

PONDÉRATION

Tous les graphes peuvent être pondérés

- Les arêtes/arcs ont un poids au lieu de l'information présent/absent

Toutes les définitions se généralisent dans ce cas, on parle alors :

- Poids d'une arête/arc
- Degré pondéré (somme des poids des arêtes adjacentes)
- La longueur d'un chemin peut-être pondérée (somme des poids des arêtes qui composent le chemin)
- Etc.

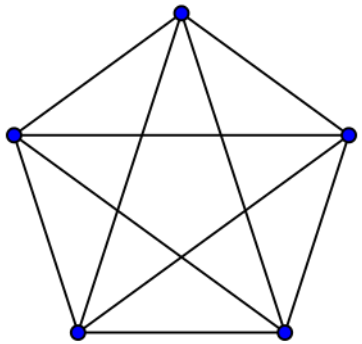
QUELQUES GRAPHES PARTICULIERS

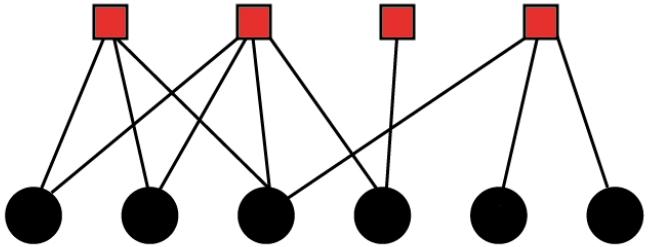
Un graphe est **complet** s'il contient toutes les arêtes possibles

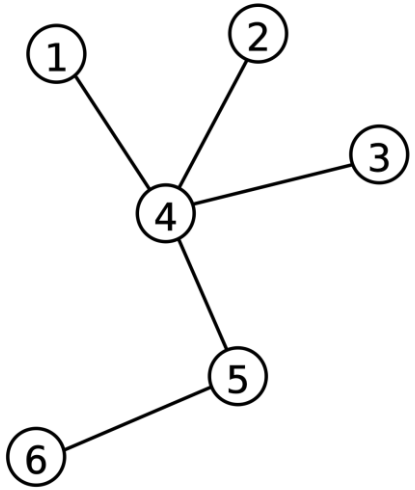
- Un sous-graphe complet est appelé une **clique**

Un graphe est **biparti** si V peut être partitionné en deux ensembles A et B de sorte que chaque arête ait une extrémité dans A et une dans B

Un **arbre** est un graphe connexe sans cycle, un ensemble d'arbre est une forêt







REPRÉSENTATION DES GRAPHES

Liste d'arêtes

- Tableau, liste chaînée, etc.

$$[(1,2),(1,4),(2,3),(3,4),(4,2)]$$

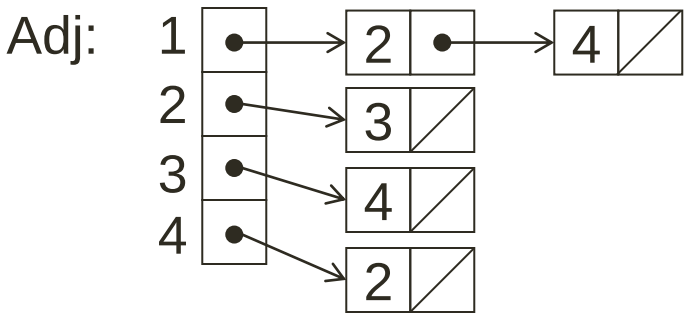
Matrice d'adjacence : $a_{ij} = 1$ s'il y a une arête entre i et j, 0 sinon

- Tableau à deux dimensions

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Liste d'adjacence : adj[u] liste les voisins de u

- Tableau de tableaux ou tableau de listes chaînées



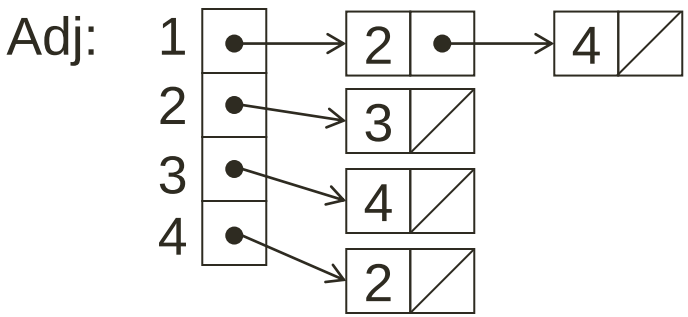
QUELLE REPRÉSENTATION POUR QUELLE UTILISATION

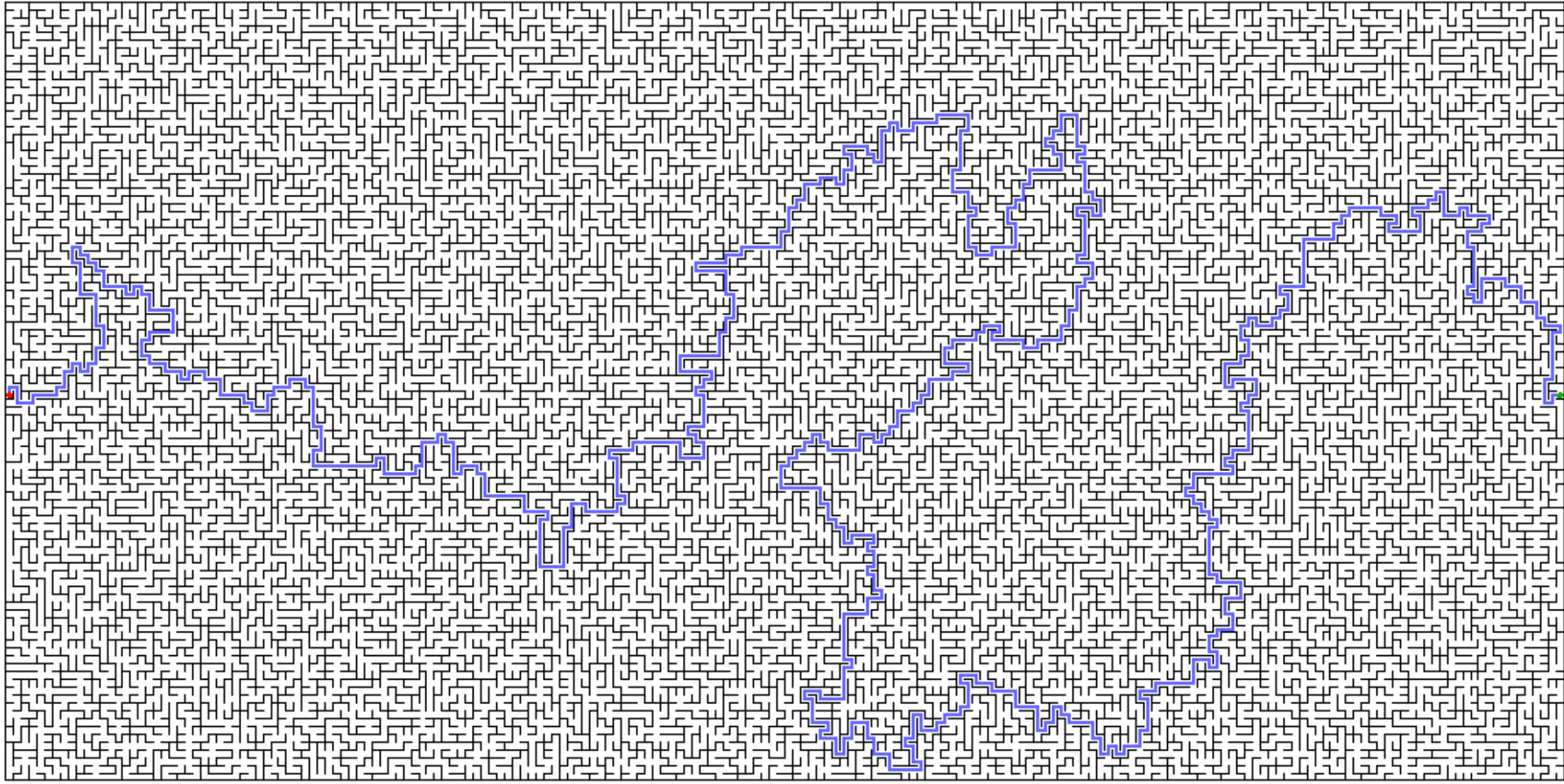
Occupation mémoire vs efficacité pour les opérations élémentaires

- Combien d'arêtes ?
- Quels sont les voisins d'un sommet u
- Y-a-t'il une arête entre u et v ?
- Etc.

$$[(1,2),(1,4),(2,3),(3,4),(4,2)]$$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$



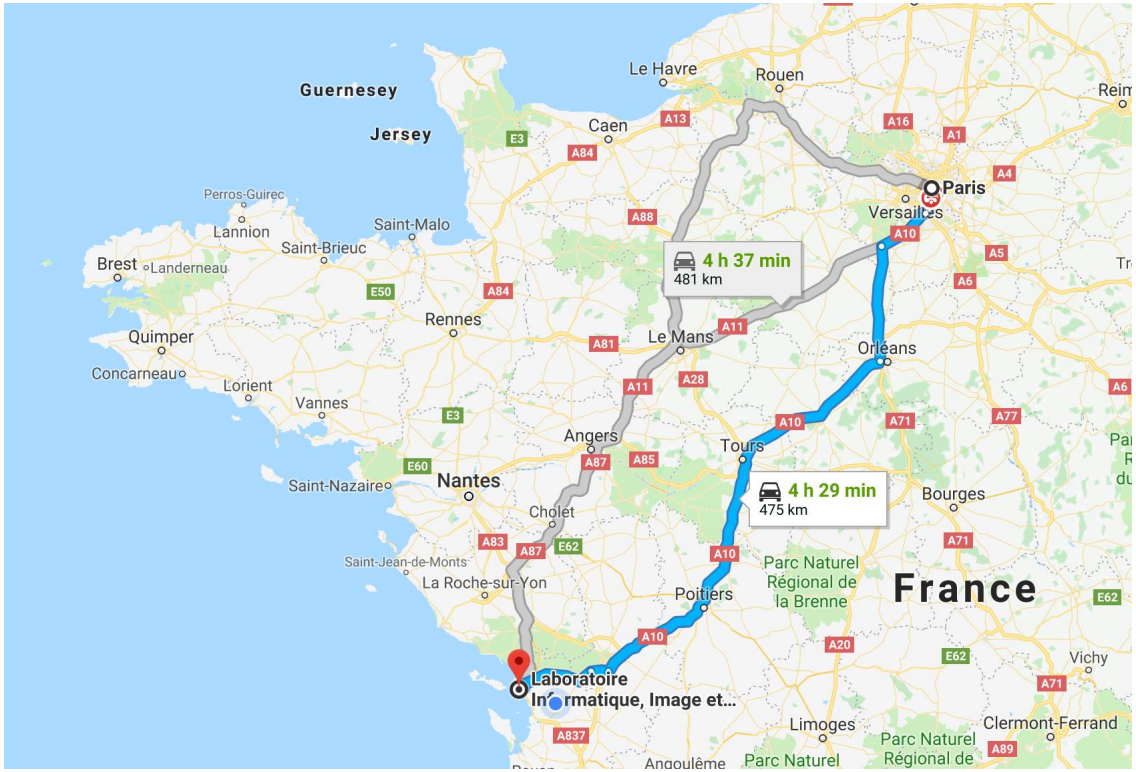


PARCOURS

Laboratoire Informatique Image Interaction (L3I)
sité de La Rochelle - Pôle Sciences et Technologie - Avenue Michel Crépeau - 17042 LA ROCHELLE CEDEX 1 Fra
Tél : +33 (0)5 46 45 82 62 – Fax : 05.46.45.82.42 – Site internet : http://l3i.univ-larochelle.fr/

POURQUOI FAIRE DES PARCOURS

La plupart des algorithmes sur les graphes nécessitent d'explorer tout ou partie du graphe



DEUX MÉTHODES DE PARCOURS

Parcours en profondeur

- On avance dans le graphe tant qu'on peut puis on revient en arrière quand on est bloqué
- DFS en anglais (depth-first search)

Parcours en largeur

- On visite tous les voisins, puis tous les voisins des voisins, etc.
- BFS en anglais (breadth-first search)

Similaire (ou presque) aux parcours sur les arbres

Et de nombreuses autres méthodes

PARCOURS EN PROFONDEUR (DFS)

Initialement tous les sommets sont blancs

```
DFS(u)
    couleur[u] ← noir
    pour chaque voisin v de u
        si couleur[v] = blanc
            DFS(v)
```

Que fait cet algorithme ?

POUR ALLER PLUS LOIN

Il faut gérer le cas des graphes non (fortement) connexes

Il est plus intéressant de garder trace de l'exploration

- A quel moment chaque sommet a été découvert
- A quel moment on a du faire marche arrière
- Construire l'arbre (la forêt) de l'exploration du graphe pour savoir d'où on a exploré chaque sommet

Certains algorithmes plus complexes utilisent ces informations

LE PARCOURS EN PROFONDEUR

Pour chaque sommet u on enregistre

- Le moment auquel u a été découvert (d[u])
- Le moment auquel on a fini de visiter tous les voisins de u (f[u])
- Le sommet depuis lequel on a découvert u (π[u])
- La couleur de u (blanc=pas découvert, gris= en cours, noir=fini)

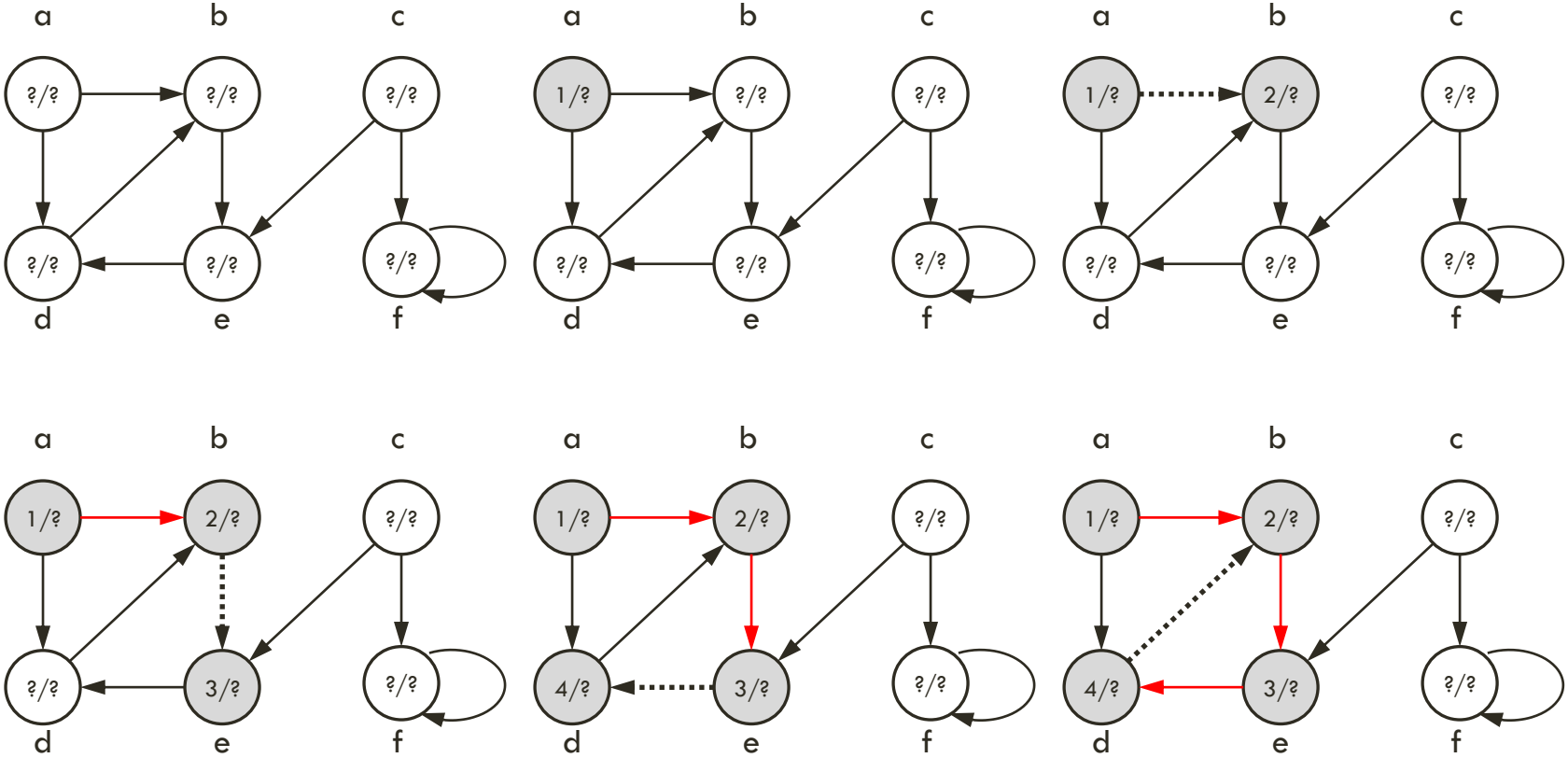
```
DFS-VISIT(u)
1  color[u] ← GRAY          ▷ White vertex u has just been discovered.
2  time ← time +1
3  d[u] ← time
4  for each v ∈ Adj[u]      ▷ Explore edge (u, v).
5       do if color[v] = WHITE
6             then π[v] ← u
7                  DFS-VISIT(v)
8  color[u] ← BLACK         ▷ Blacken u; it is finished.
9  f[u] ← time ← time +1
```

LE PARCOURS EN PROFONDEUR

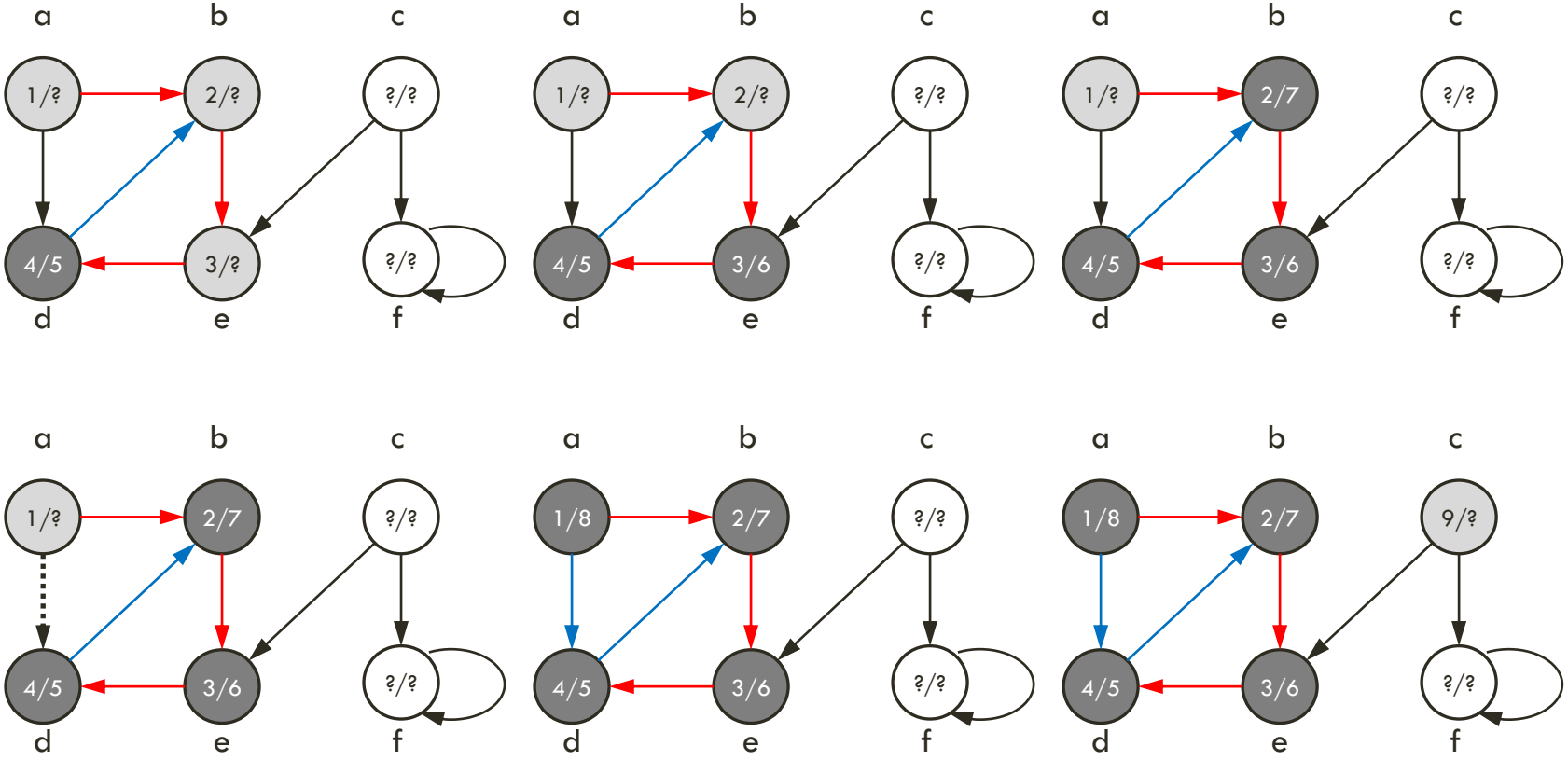
Initialisation et gestion des composantes connexes

```
DFS(G)
1  for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3          π[u] ← NIL
4  time ← 0
5  for each vertex u ∈ V[G]
6       do if color[u] = WHITE
7             then DFS-VISIT(u)
```

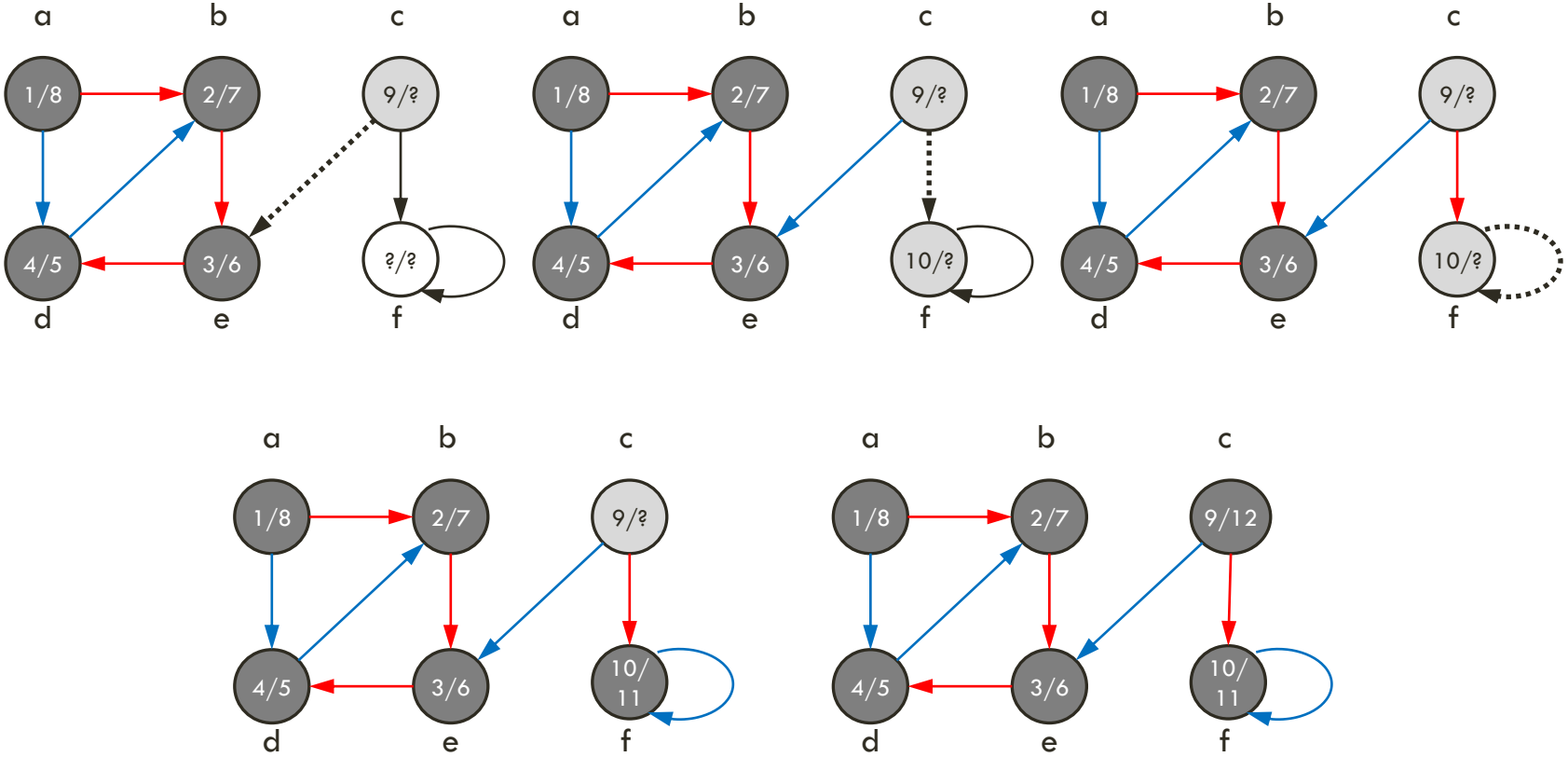
EXEMPLE COMPLET



EXEMPLE COMPLET



EXEMPLE COMPLET



COMPLEXITÉ DE L'ALGORITHME

Que faut-il stocker pour que cet algorithme s'exécute correctement ?

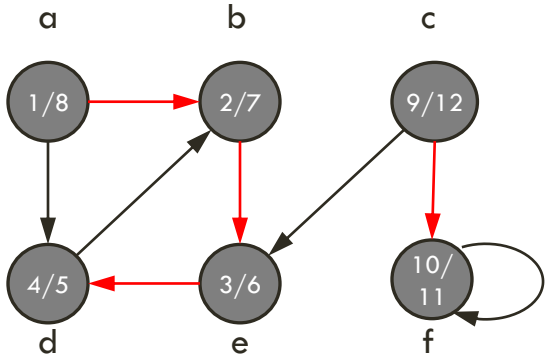
- Complexité spatiale (mémoire nécessaire)

Combien d'opérations faut-il faire au total ?

- Complexité temporelle (ou juste complexité)

Quel est l'arbre (la forêt) retourné par l'algorithme (variable π) ?

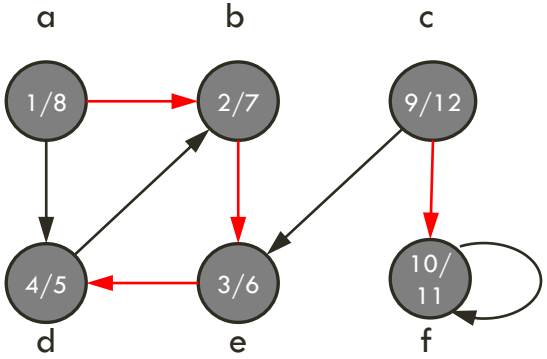
```
DFS-VISIT(u)
1  color[u] ← GRAY          ▷ White vertex u has just been discovered.
2  time ← time + 1
3  d[u] ← time
4  for each v ∈ Adj[u]      ▷ Explore edge (u, v).
5      do if color[v] = WHITE
6          then π[v] ← u
7               DFS-VISIT(v)
8  color[u] ← BLACK         ▷ Blacken u; it is finished.
9  f[u] ← time ← time + 1
```



PARCOURS EN PROFONDEUR - ARCS

Le parcours en profondeur permet d'étiqueter les arcs en 4 familles

- Les arcs de liaison (ceux de la forêt utilisés durant le parcours)
- Les arcs arrières (Back) qui ramènent vers un ancêtre (découvert avant dans l'arbre) : d vers b, f vers f
  - Comment identifie-t-on un ancêtre ?
- Les arcs avants (Forward) qui amènent à un descendant (découvert après dans l'arbre) : a vers d
  - Comment identifie-t-on un descendant ?
- Les arcs transverses (Cross), tous les autres : c vers e



APPLICATIONS DU DFS

Comment savoir si un graphe a un cycle ?

- Dans le cas d'un graphe non orienté
- Dans le cas d'un graphe orienté

Comment calculer les composantes connexes d'un graphe ?

- Dans le cas d'un graphe non orienté
- Dans le cas d'un graphe orienté

Cf TD 1

PARCOURS EN LARGEUR

L'idée est similaire mais ici on visite un sommet puis tous ses voisins, puis tous les voisins des voisins, etc.

- On découvre donc tous les sommets à distance k avant de découvrir eux à distance k+1
- Parcours en largeur = Breadth-first search = BFS

On part du même principe que pour le DFS, en stockant :

- La distance à la source
- L'arbre de visite des sommets

PARCOURS EN LARGEUR - ALGORITHME

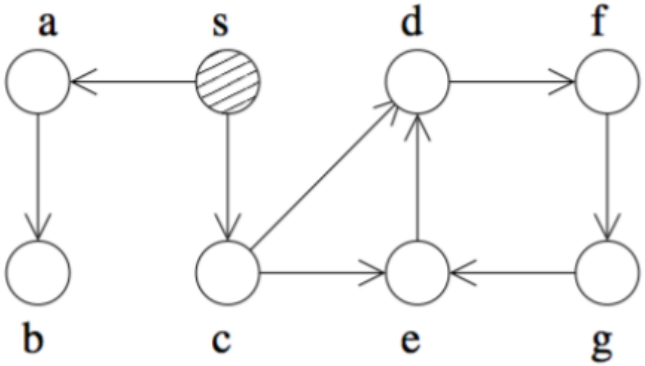
```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

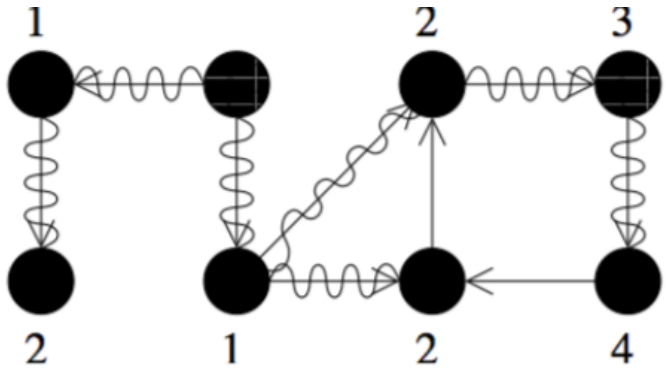
RÉSULTAT

Pour chaque sommet on sait à quelle distance (en nombre d'arcs) il est de la source (preuve ?)

Il faut aussi gérer les cas des graphes non connexes

- donc rajouter une fonction similaire à DFS





TRI TOPOLOGIQUE

Dans un graphe orienté on peut considérer que s'il y a un arc (u,v) alors u doit être exécuté avant v

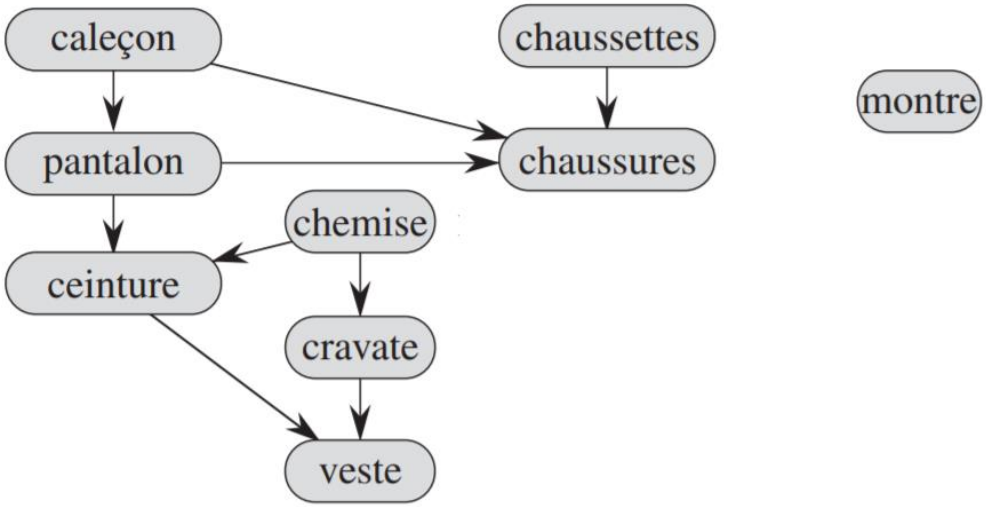
- Par exemple le graphe peut représenter l'ensemble des actions à réaliser pour développer un logiciel
- Make utilise un tri topologique pour savoir dans quel ordre compiler les fichiers

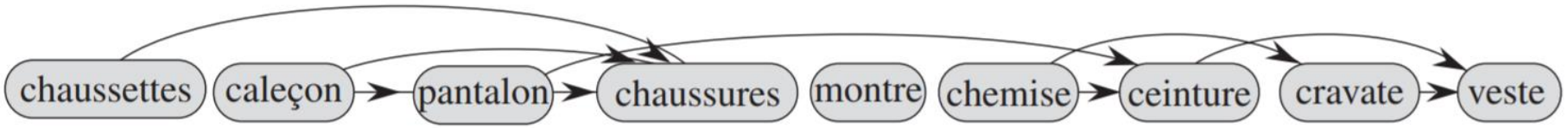
Un tri topologique permet, à partir d'un graphe orienté, de donner un ordre d'exécution sur les sommets respectant les arcs

EXEMPLE

Ordre pour s'habiller

- On veut que tous les arcs aillent de la gauche vers la droite

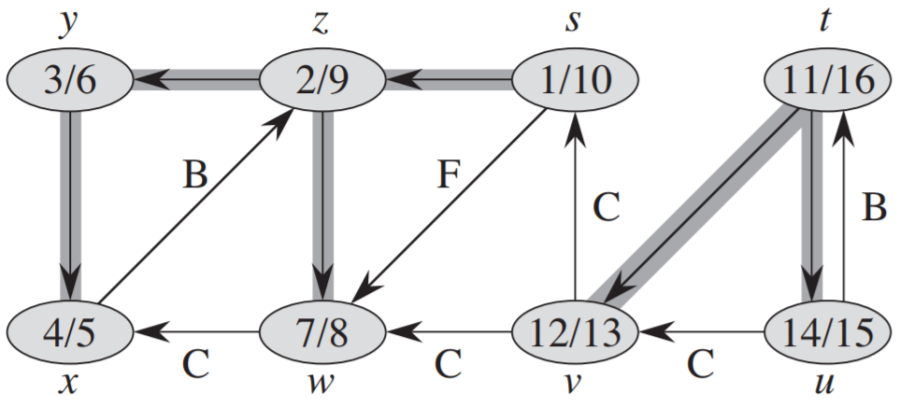




ALGORITHME

Peut-on toujours faire un tri topologique d'un graphe orienté ?

- Non, par exemple…



Comment caractériser les graphes qui peuvent en admettre un ?

- DAG = Directed Acyclic Graph

ALGORITHME

Théorème : Lister les sommets d'un DAG par ordre décroissant de temps de fin de visite donne un tri topologique.

Autre version : S'il y a un arc (u,v) alors f[u] > f[v] (et donc u sera avant v dans le tri)

Preuve : supposons un arc (u,v)

Cas 1 : DFS visite u en premier. Alors v sera visité et terminé avant que u ne le soit donc f[u] > f[v].

Cas 2 : DFS visite v en premier. Il ne peut pas y avoir de chemin du v vers u (car le graphe est un DAG), donc v sera terminé avant de découvrir u, donc f[u] > f[v].

TRI TOPOLOGIQUE - ALGORITHME

Le théorème n'est valable que pour les DAGs. Un algorithme simple consiste donc à :

- Calculer un parcours en profondeur du graphe pour calculer les dates de fin de traitement
- Ordonner les sommets par date décroissante

On peut aussi modifier le parcours en profondeur pour qu'il ajoute tout sommet terminé en tête d'une liste, ça évite de trier a posteriori…

Ok, mais comment savoir si un graphe est un DAG ?

EXERCICES

Caractériser les graphes contenant une chaîne Eulerienne

Idem pour un cycle Eulerien

Montrer que les propriétés suivantes caractérisent un arbre

- $|E| = |V| - 1$
- Deux sommets quelconques sont connectés par une et une seule chaîne
- Supprimer un lien déconnecte un arbre
- Ajouter une arête crée un cycle