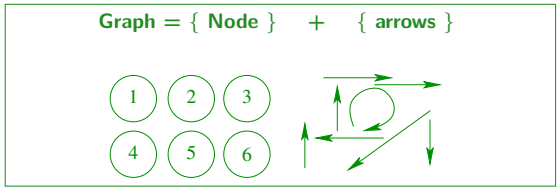
Plan

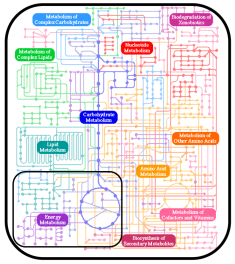
1. Introduction to algorithm complexity
2. Exact Pattern Matching
3. Graph algorithms
   - Introduction
   - Definitions
   - Representation by adjacency list
   - Representation by adjacency matrix
   - Breadth-first search
   - Depth-first search
   - Topological sorting of an acyclic directed graph
   - Strongly related components
   - Shortest path search
   - Dijkstra Algorithm
   - Bellman–Ford Algorithm

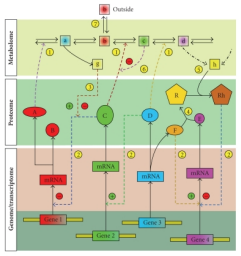
Graphs are everywhere...

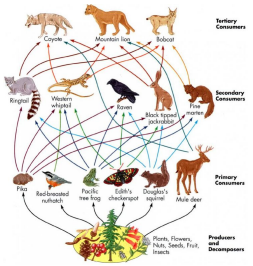


- Cartography : Road network, Internet network,
- Economics - Management : Supply planning, flow management, scheduling,
- Chemistry : Molecular modelling, DNA,
- Social sciences : Genealogy, mass phenomena, conflict,
- Linguistics : Grammar, Compilation,
- Artificial Intelligence : Behaviour
- Biology : phylogeny (trees ≡ a subclass of graphs), metabolic networks, genetic networks, prey-predator networks, molecular modelling...

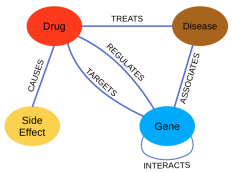
The key abstract idea to retain is : Interactions !



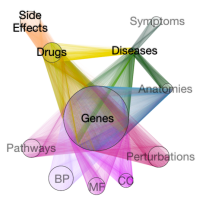


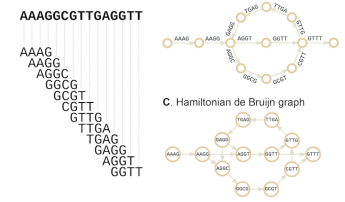


Model / Schema / Metagraph
(Subnetwork for GraphGist)

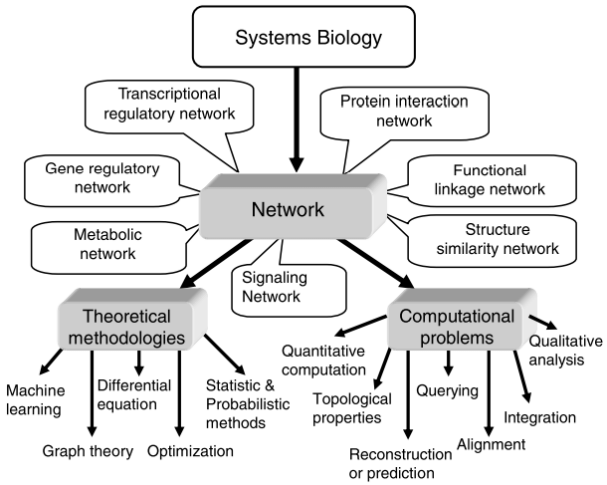


Entire Rebel Hetnet



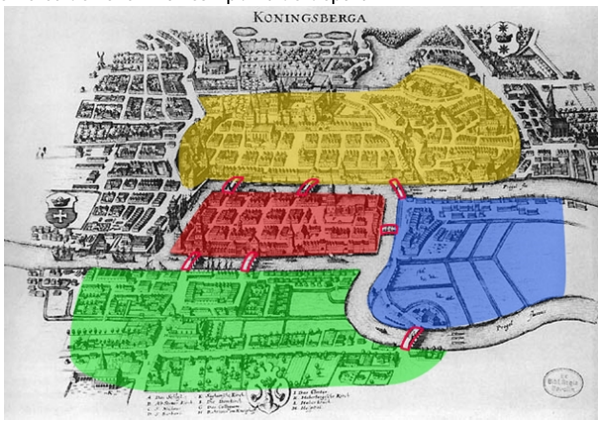


The key abstract idea to retain is : Interactions !



Le problème des ponts de Königsberg (18ème s.)

les habitants de Königsberg se demandaient s'il était possible, en partant d'un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ.

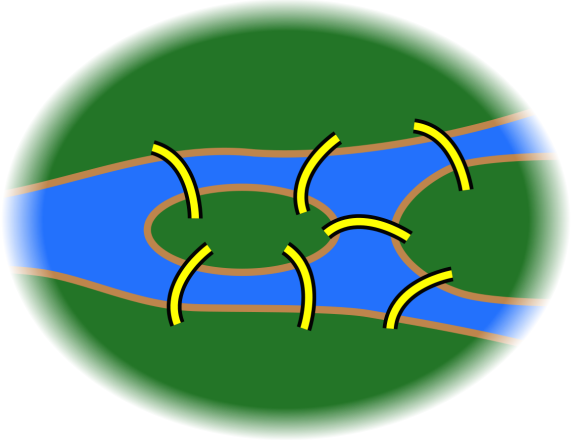


CÔTE D'AZUR

Le problème des ponts de Königsberg (18ème s.)

Algorithmics for Biology
Jean-Paul Comet
Complexity
Pat. Matching
Graphs
Intro
Definitions
Adjacency List
Adjacency Matrix
BFS
DFS
Topological sorting
strongly. connect.
Shortest path
Dijkstra
Bellman–Ford
Dyn.Prog.
Sequences

les habitants de Königsberg se demandaient s'il était possible, en partant d'un quartier quelconque de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir à leur point de départ.



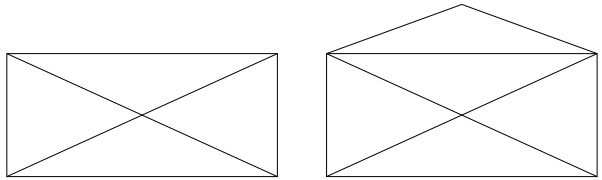
Une telle promenade n'existe pas, et c'est Euler qui donna la solution de ce problème en caractérisant les graphes que l'on appelle aujourd'hui « Eulériens » en référence à l'illustre mathématicien.

CÔTE D'AZUR

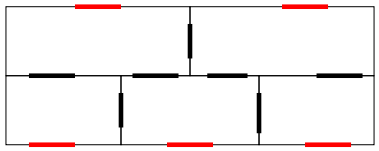
Autres exemples

Algorithmics for Biology
Jean-Paul Comet
Complexity
Pat. Matching
Graphs
Intro
Definitions
Adjacency List
Adjacency Matrix
BFS
DFS
Topological sorting
strongly. connect.
Shortest path
Dijkstra
Bellman–Ford
Dyn.Prog.
Sequences

1. Peut-on dessiner sans lever le crayon et en ne passant qu'une seule fois sur chaque arête les figures ci-contre ?



2. Peut-on "passer" d'une pièce à l'autre en franchissant une fois et une seule chacune des frontières ? On considérera d'abord le cas où aucune pièce n'a de porte vers l'extérieur, puis le cas où chaque pièce a une porte vers l'extérieur.



CÔTE D'AZUR

Définitions

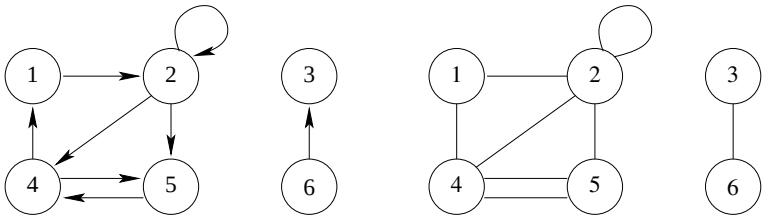
Algorithmics for Biology
Jean-Paul Comet
Complexity
Pat. Matching
Graphs
Intro
Definitions
Adjacency List
Adjacency Matrix
BFS
DFS
Topological sorting
strongly. connect.
Shortest path
Dijkstra
Bellman–Ford
Dyn.Prog.
Sequences

Un graphe orienté G est un couple (S, A) où :

- S : un ensemble fini dont les éléments sont appelés des *sommets*
- A : un sous-ensemble du produit cartésien $S \times S$. Les éléments de A sont appelés des *arcs*.

Un graphe non-orienté G est un couple (S, A) où :

- S : un ensemble fini dont les éléments sont appelés des *sommets*
- A : ensemble de paires non-orientées de sommets. une arête est un ensemble de 2 sommets ($a = \{u, v\}$) On note indifféremment $a = \{u, v\}$ ou $a = \{v, u\}$. Les éléments de A sont appelés des *arêtes*.



Degrés, chemin, chaîne, circuit, cycle

- Dans un graphe orienté, l'arc (u, v) **part** de u et **arrive** en v.
  Dans un graphe non orienté, l'arête (u, v) est **incident** à u et à v.
  Si $(u, v) \in A$, on dit que v est **adjacent** à u.
  Dans un graphe non orienté, la relation d'adjacence est *symétrique*.
- Le **degré** d'un sommet dans un graphe non orienté est le nombre d'arêtes qui lui sont incidentes. Pour les graphes orientés, on a :
  - le **degré sortant** : le nombre d'arcs qui partent du sommet,
  - le **degré entrant** : le nombre d'arcs qui arrivent au sommet.
- Un **chemin** dans un graphe orienté (resp. **chaîne** dans un graphe non orienté) de longueur k d'un sommet u vers un sommet u' est une séquence $< v_0, v_1, ..., v_k >$ de sommets tels que $u = v_0$, $u' = v_k$ et $(v_{i-1}, v_i) \in A$, pour $i = 2...k$.
  La **longueur** du chemin (resp. de la chaîne) est le nombre d'arcs de ce chemin (resp. le nombre d'arêtes de la chaîne).
- Un sommet u' est dit **accessible** à partir de u s'il existe un chemin de u vers u'. On note $u \rightsquigarrow u'$. Un chemin est **élémentaire** si tous ses sommets sont distincts.
- Un chemin d'un graphe orienté forme un **circuit** si le dernier sommet est égal au premier. Un circuit est dit **élémentaire** s'il ne passe pas deux fois par le même sommet. Une **boucle** est un circuit de longueur 1.
- Dans un graphe non-orienté, une **chaîne** $< v_0, v_1, ..., v_k >$ forme un **cycle** (**élémentaire**) si $k \geq 2$ et si $v_0 = v_k$ (et si les sommets sont distincts). Un graphe sans cycle est dit **acyclique**.

Connexité & co.

- Un graphe non-orienté est **connexe** si pour tout couple de sommets, les sommets sont reliés par une chaîne. Les **composantes connexes** sont les classes d'équivalence de sommets induites par la relation "*est accessible à partir de*".
- Un graphe orienté est **fortement connexe** si chaque sommet est accessible à partir de n'importe quel autre sommet. Les **composantes fortement connexes** sont constituées des classes d'équivalence de sommets induites par la relation "*sont accessibles mutuellement*".
- Un graphe est **complet** si tous les sommets sont adjacents à tous les autres.
- Un graphe connexe non orienté acyclique est un **arbre**.

Rappel : Une relation d'équivalence dans un ensemble E est une relation binaire qui est à la fois

- réflexive (pour tout élément x de E, on a $x \sim x$),
- symétrique (chaque fois que deux éléments x et y de E vérifient $x \sim y$, ils vérifient aussi $y \sim x$) et
- transitive (chaque fois que trois éléments x, y et z de E vérifient $x \sim y$ et $y \sim z$, ils vérifient aussi $x \sim z$).

On définit la classe d'équivalence $[x]$ d'un élément x de E comme l'ensemble des y de E tels que $x \sim y$:

$$y \in [x] \Leftrightarrow x \sim y.$$

On appelle représentant de [x] n'importe quel élément de [x].
L'ensemble des classes d'équivalence forme une partition de E.

Utilisation des graphes en bio-informatique

- L'alignement de séquences peut être vu comme un problème d'optimisation de chemins dans un graphe orienté.
- reconstruction des arbres phylogénétiques
- Les réseaux génétiques sont modélisés par des graphes d'interaction entre entités.
- La classification hiérarchique est basé sur un arbre.
- Réseaux complexes d interaction : réseaux de protéines

Représentation par liste d'adjacence

- La représentation par liste d'adjacence est utilisée pour les graphes peu denses : ceux pour qui $|A|$ est bien inférieur à $|S|^2$, souvent on note $|A| << |S|^2$
- Pour chaque nœud/sommet, la liste d'adjacence $Adj[u]$ est un liste chaînée des sommets v tels qu'il existe un arc (u, v).
  Pour un graphe orienté, la somme de toutes les longueurs des listes chaînées vaut $|A|$.
  Pour un graphe non orienté, la somme de toutes les longueurs des listes chaînées vaut $2 \times |A|$.
- Complexité mémoire : $\Theta(|S| + |A|)$
- Si on a un graphe pondéré : chaque arc possède un poids donné habituellement par une fonction de pondération,

$$\omega : A \longrightarrow \mathbb{R} \tag{2}$$

On note $\omega(u, v)$ le poids de l'arc (u, v).

Représentation par matrice d'adjacence

Pour cette représentation des graphes, on suppose que les sommets sont numérotés $1, 2, ..., |S|$.

- les arcs sont décrits par une matrice $|S| \times |S|$: $M = (a_{ij})$ telle que

$$a_{ij} = \begin{cases} 1 & \text{si} \quad (i,j) \in A \\ 0 & \text{sinon} \end{cases} \tag{3}$$

- Complexité mémoire : $\Theta(|S|^2)$.
  Dans un graphe non orienté, $M = {}^tM$ (transposée de la matrice initiale).
  On peut alors réduire d'un facteur 2 l'espace mémoire nécessaire, en ne mémorisant que la matrice triangulaire supérieure (avec la diagonale si on a des boucles).
- Si le graphe est pondéré :

$$a_{ij} = \begin{cases} \omega(u_i, u_j) & \text{si} \quad (i,j) \in A \\ 0 \text{ ou } \texttt{NIL} \text{ ou } \infty & \text{sinon} \end{cases} \tag{4}$$