TCA101: Initiation à l'Informatique

Robert Strandh

TCA 101, Informatique

Initiation à l'informatique

Responsable et cours : Cyril Gavoille & Robert Strandh

Planning :

6 séances de cours

6 séances de TD

6 séances de TP

+ environ 3h de travail individuel par semaine

Web : http ://dept-info.labri.u-bordeaux.fr/tca101/

Support de cours

Polycopié : *Initiation à l'Informatique*
par Robert Strandh et Irène Durand

Transparents

Objectif et contenu

Objectif : Donner une idée fidèle du contenu des études supérieures en informatique

Thème : Étude d’un objet appelé *graphe*

Contenu : Théorie des graphes (cours)

Algorithmique des graphes (TD)

Programmation des algorithmes de graphes (TP)

Il faut activer les comptes

Vous avez reçu un compte sur l'ensemble des ordinateurs de l'université.
Il faut suivre les instructions pour activer le compte au moins 24 heures (de préférence plus) avant le premier TP.

C'est quoi l'informatique ?

- dans la vie quotidienne : ordinateur avec logiciels
- à l'université : une discipline universitaire

L'informatique n'est pas une science (expérimentale).
L'informatique est similaire aux mathématiques (étude d'objets abstraits).
Les objets en mathématiques : nombres, relations, fonction, transformations, *etc*.
Les objets en informatique : algorithmes, programmes, preuves, systèmes de réécriture, images numériques, graphes, *etc*.

Domaines en informatique fondamentale

Exemples de domaines :

- Algorithmique. Les méthodes les plus efficaces pour traiter un problème donné.
- Structures de données. La meilleure façon d'organiser un ensemble de données dans le but d'y accéder rapidement.
- Complexité. Une façon d'exprimer l'efficacité d'un algorithme, indépendamment d'un ordinateur ou d'un langage de programmation particulier.

Domaines de l'informatique fondamentale (suite)

Exemples de domaines plus théoriques :

- Théorie des langages. Différentes façons de produire et reconnaître des suites de symboles.
- Calculabilité. Déterminer pour quels problèmes il est théoriquement possible/impossible d'écrire un programme.
- Logique. La puissance d'expression de différents types de logique.

Domaines de l'informatique pratique

Exemples de domaines :

- Programmation. Techniques pour organiser un programme de façon qu'il soit facilement modifiable.
- Génie Logiciel. Méthodes pour organiser le développement d'un logiciel de grande taille.
- Informatique multimédia. Méthodes d'analyse, modification et synthèse d'images et de sons.
- Systèmes d'exploitation. Techniques pour réaliser un système qui assure intégrité, sécurité et performance.
- Compilation. Techniques pour traduire un programme en code machine efficace.

Pourquoi étudier l'informatique

Deux sous-questions :

- pourquoi choisir une carrière en informatique ?
- pourquoi étudier l'informatique alors qu'on a choisi une carrière différente (physique, chimie, mathématique, *etc.*) ?

Pourquoi une carrière en informatique ?

Raisons techniques :

- garantie d'embauche,
- les produits de haute technologie contiennent de plus en plus de logiciels,
- la complexité des logiciels augmente,
- la compétition est de plus en plus sophistiquée.

Raisons non techniques :

- contacts (souvent internationaux),
- voyages,
- mobilité (même internationale).

Pourquoi l'informatique pour les non informaticiens

- Le travail d'un scientifique ou d'un ingénieur nécessite de plus en plus la manipulation de logiciels,
- Ces logiciels sont de plus en plus sophistiqués,
- Souvent, ces logiciels nécessitent de la programmation,
- Pour programmer de manière efficace, il faut des connaissances en informatique (algorithmique, programmation).
- C'est surtout nécessaire pour produire des programmes maintenables.

Un mot sur l'importance de l'algorithmique

Il est facile de se tromper d'algorithme.
Une telle erreur peut faire la différence entre plusieurs années et quelques minutes de calculs sur une même machine.
C'est souvent une question d'utilisation de structures de données ou d'algorithmes connus dans la littérature.

Un mot sur la programmation

Il ne suffit pas de construire un programme qui marche.
L'essence de la programmation est l'organisation pour faciliter la maintenance (qui représente environ 80% du coût d'un logiciel).
Cela nécessite la construction d'*abstractions* (sous-programmes, modules, classes, extensions syntaxiques, fonctions de première classes, *etc.*).
Plusieurs styles de programmation adaptés aux types différents de problèmes : programmation impérative, fonctionnelle, orientée-objets, logique. Chaque type a ses idiomes de programmation qu'il faut apprendre.

Prérequis pour études supérieures en informatique

Prérequis :

- Il faut être bien organisé (ça s'apprend),
- Il faut avoir une curiosité intellectuelle, car l'informatique nécessite un apprentissage permanent,

Non prérequis :

- Connaissance préalable d'un langage ou d'un système d'exploitation,
- Connaissance de la programmation (c'est souvent un handicap),
- Connaissance de logiciels destinés au grand public.

Choix d’un langage de programmation

Paramètres (langage ou implémentation du langage) :

- facilité d’apprentissage, facilité d’utilisation,
- rapidité d’exécution, rapidité de compilation,
- absence de défauts dans le compilateur,
- pérennité (fabricant, langage, implémentation),
- disponibilité de programmeurs,
- expressivité du langage (structuration, styles),
- normalisation, conformité des implémentations.

Choix d’un langage pour l’enseignement

- facilité d’apprentissage (moins important dans l’industrie),
- utilité plus tard,
- facilité de programmer de façon propre et modulaire.

Nous avons choisi le langage Python.

Caractéristiques de Python

- implémentation libre et gratuite existe,
- orienté-objets,
- facilité de manipulation de listes,
- grand nombre de bibliothèques,
- efficacité moyenne du code,
- structure de bloc indiquée par l'indentation (unique pour Python).

Le Graphe

C’est une *collection d’objets* munie d’une *relation binaire* entre ces objets.
Une relation binaire est un *ensemble de couples d’objets*.
En mathématiques, la collection est souvent infinie et *non dénombrable* (les réels par exemple), alors qu’en informatique, elle est souvent dénombrable et parfois finie.
En informatique, les objets représentés sont souvent des objets plus concrets (molécules, composants électroniques, villes, réseaux de téléphones mobiles, personnes).

Exemple de graphe : repas de Noël

Ensemble : toutes les personnes assistant à un repas de Noël.
Relation : l'ensemble des couples de personnes (p_1, p_2) tels que p_1 est un ancêtre de p_2 (relation non symétrique).

Exemple de graphe : molécules

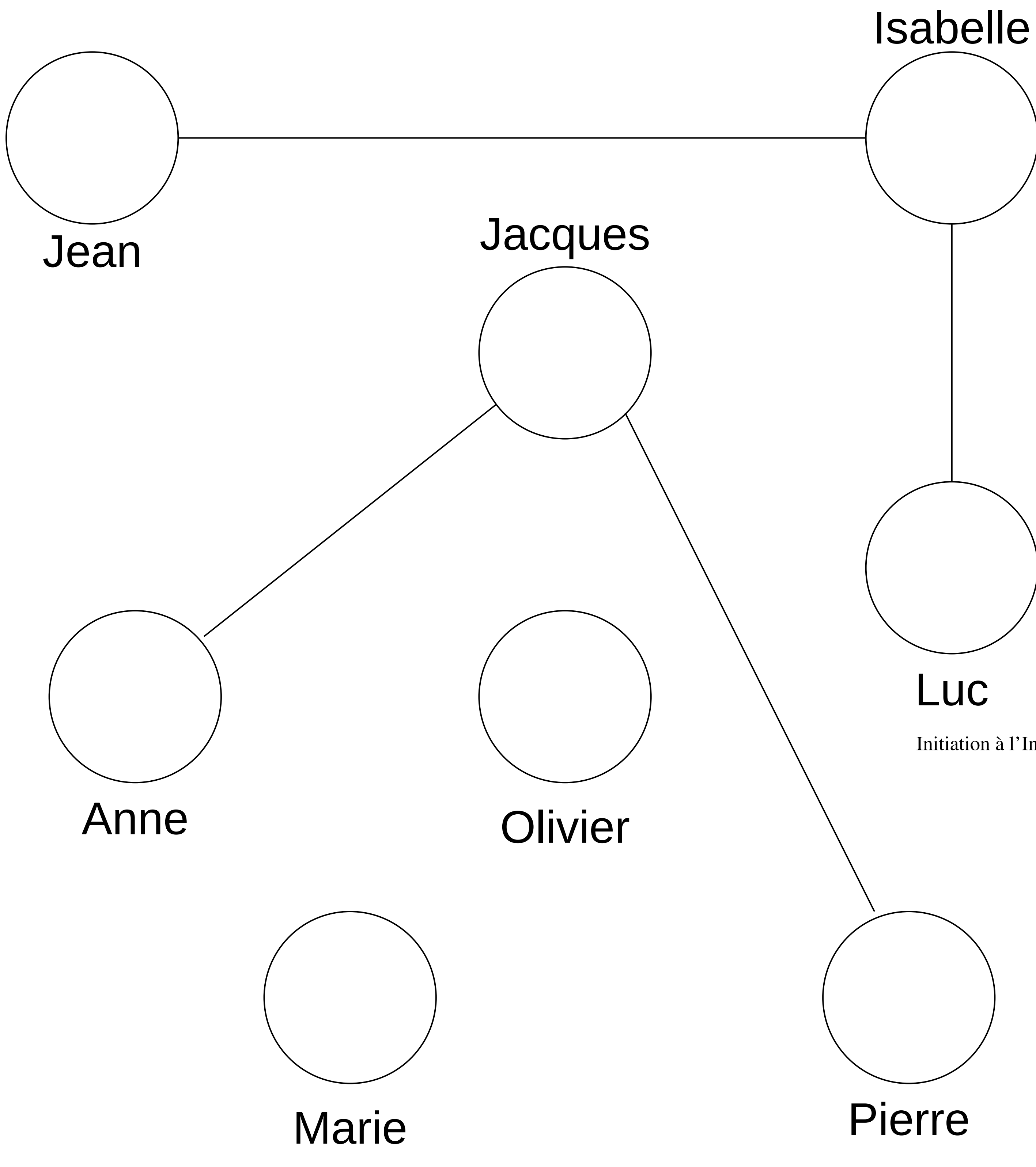
Ensemble : les atomes d'une molécule.
Relation : l'ensemble des couples d'atomes (a_1, a_2) tels que a_1 et a_2 partagent au moins un électron (liaison covalente, relation symétrique).

Exemple de graphe : réseau de stations-relais

Ensemble : les stations-relais d'un réseau de communication de téléphones mobiles.
Relation : l'ensemble des couples de stations (s_1, s_2) tels qu'il existe un lien direct de communication (par micro-ondes, câble, *etc.*) entre s_1 et s_2.

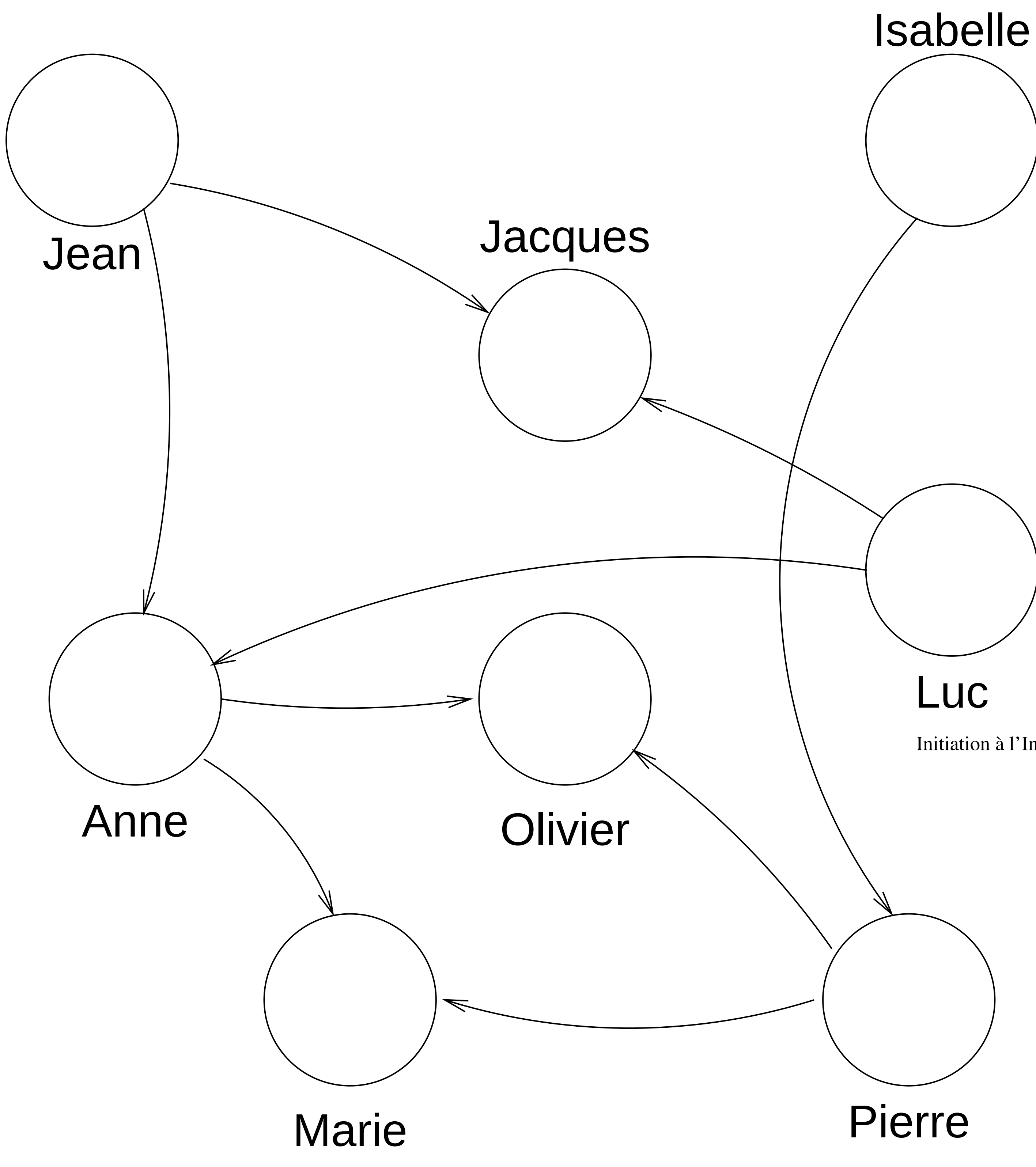
Représentation graphique d'un graphe

Les cousins (relation symétrique, graphe non orienté) :



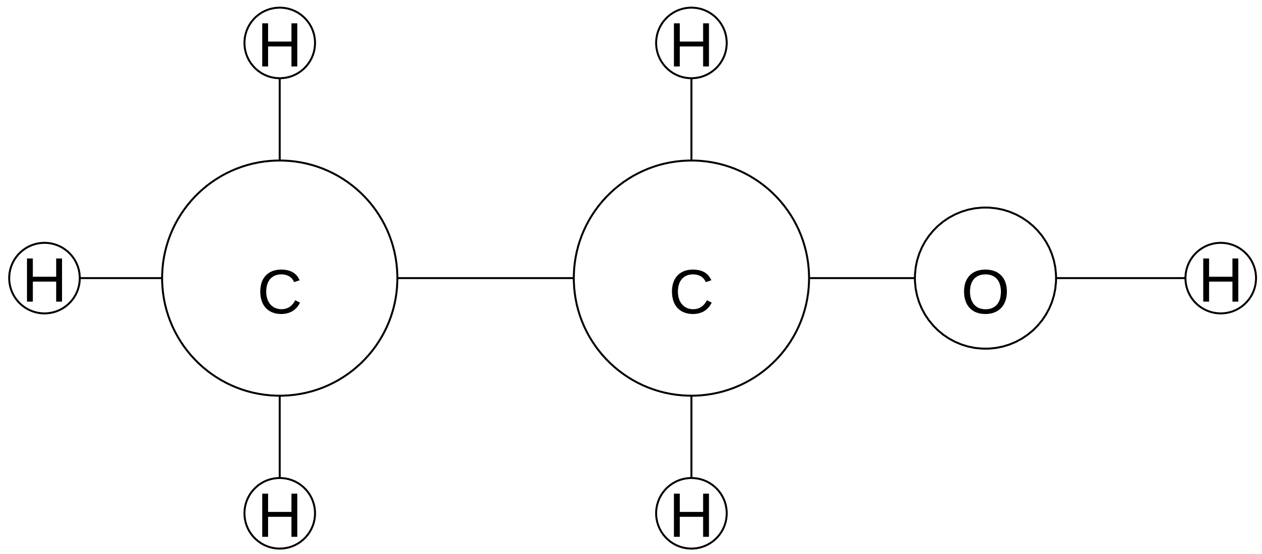
Représentation graphique d'un graphe

$x \longrightarrow y$ = "x a pour parent y"
(relation non symétrique, graphe orienté) :



Représentation graphique d'un graphe

Molécule (relation symétrique, graphe non orienté) :



Concepts et notation

Il s'agit de donner un *nom* et un *façon d'écrire* certaines notions fréquemment utilisées.
Raison pour introduire des concepts et des notations :

- évite la répétition de phrases compliquées,
- précision ; on évite l'ambiguïté,

Exemples connus : racine carrée.

Concepts et notations ensemblistes

Nous supposons que la notion d'*ensemble* est connue.
Par contre, en informatique, il faut souvent préciser la *fonction de comparaison* utilisée entre deux éléments de l'ensemble.
Exemple :
L'objet a est une Renault Clio immatriculée 1234AB33, l'objet b est une Renault Mégane immatriculée 1234AB24, et l'objet c une Renault Clio immatriculée 1234AB40.
Est-ce que c est élément de l'ensemble $\{a, b\}$?

Élément d'un ensemble

Nous utilisons la notation $x \in E$ pour dire que x est élément de l'ensemble E.
Cette notation ne précise pas le test d'égalité qu'il faut donc préciser séparément.

Sous-ensembles

Nous utilisons la notation $E \subseteq F$ pour dire que E est un sous-ensemble de F, à savoir que tout élément de E est aussi élément de F. Attention, il est possible que $E = F$. Sinon on écrit $E \subset F$
Pour l'ensemble des sous-ensembles d'un ensemble E (appelé les *parties* de E), nous utilisons la notation $\mathcal{P}(E)$.

Fonctions

Une fonction est un objet mathématique qui, à un objet d'un ensemble fait correspondre un objet d'un autre ensemble. Exemple : $y = sin\ \alpha$.
En informatique les ensembles sont plus souvent des personnes, des voitures, des graphes, des sommets ou des arêtes.
On dit qu'une fonction est *appliquée à un ou plusieurs arguments* et qu'elle *renvoie (ou retourne) une valeur*.
Ceci reflète l'aspect exécutable d'une fonction.

Domaine et image d'une fonction

L'ensemble de tous les arguments possibles d'une fonction ϕ est le *domaine* de la fonction : $dom(\phi)$.
L'ensemble de toutes les valeurs possibles d'une fonction est l'*image* de la fonction : $img(\phi)$.
Notation : $\phi : dom(\phi) \longrightarrow img(\phi)$.
Par exemple $sin : \mathbb{R} \longrightarrow [-1, 1]$.

Conditions nécessaires et suffisantes

Condition nécessaire : “A est une condition nécessaire pour B” est la même chose que “B implique A” ou $B \Rightarrow A$. Ici B est l’objectif.
Une autre façon de le dire : “B seulement si A”
Condition suffisante : “A est une condition suffisante pour B” est la même chose que “A implique B” ou $A \Rightarrow B$. B est encore l’objectif.
Une autre façon de le dire : “B si A”
Condition nécessaire et suffisante : “B si et seulement si A”. B est l’objectif et souvent un concept à définir.

Raisonnement par l'absurde

Nous avons besoin de calculer la négation d'une phrase.
La négation de "B seulement si A" est "non A mais B". (le "mais" est le "et" logique)
La négation de "B si A" est "A mais non B".
La négation de "$\forall a\ A$" est "$\exists a\ \neg A$."
La négation de "$\exists b\ B$" est "$\forall b\ \neg B$."

Complexité

Comment savoir si une méthode est *efficace* ?
C'est le problème du domaine de la *complexité asymptotique*, ou simplement la *complexité*.

Complexité (suite)

On suppose l'existence d'un ensemble d'*opérations* simples et rapides (ou opérations élémentaires).
Une opération est simple et rapide si un ordinateur peut l'exécuter avec un nombre faible d'instructions.
Exemple d'opérations élémentaires :

- additionner, soustraire, multiplier ou diviser deux nombres,
- tester si une valeur est égale à une autre valeur,
- affecter une valeur à une variable.

Complexité (suite)

Pour déterminer si une méthode est efficace, on compte d'abord le nombre d'opérations nécessaire à effectuer dans le pire des cas et *en fonction de la taille du problème*.
Par exemple, pendant un pot, on souhaite que chaque participant serre la main à chaque autre participant. L'opération élémentaire est "serrer la main". La taille du problème est le nombre de participants.

Complexité (suite)

En général, pour n personnes, il faut $n(n-1)/2$ opérations élémentaires.
Une autre façon d'écrire $n(n-1)/2$ est $\frac{1}{2}n^2 - \frac{1}{2}n$.

Complexité (suite)

Pour obtenir la complexité asymptotique, on remplace d'abord toute constante (de type $\frac{1}{2}$ ou 55) par 1. Cela nous donne $n^2 - n$.

Puis, on garde uniquement le terme le plus grand pour n grand. Cela donne n^2.

Finalement, on indique que ces approximations ont été effectuées en rajoutant $O()$ comme ceci : $O(n^2)$.

En réalité, on effectue les approximations avant de compter exactement.

Définition d'un graphe (1)

Une première tentative :
Un *graphe* est un couple (V, E), où V est un ensemble d'objets appelés les *sommets* du graphe (V pour l'anglais "vertex"), et $E \subseteq V mes V$ est une relation binaire sur $V mes V$. Les éléments de E sont appelés les *arêtes* du graphe (E pour l'anglais "edge").

Problème de la définition

Cette définition a plusieurs problèmes :

- On ne peut pas avoir deux arêtes différentes entre deux sommets (les arêtes n'ont pas d'identité propre),
- Un couple (s_1, s_2) n'est pas le même que (s_2, s_1).

En fait, la définition donne ce que l'on appelle un *graphe orienté simple*.
Ici *simple* signifie qu'il y a au plus une arête entre deux sommets.

Définition d'un graphe (2)

Deuxième tentative :
Un *graphe* est un triplet (V, E, ϕ), où V est un ensemble d'objets appelés les *sommets* du graphe, et E est un ensemble d'objets appelés les *arêtes* du graphe et ϕ est une fonction $\phi : E \longrightarrow \mathcal{P}(V)$ telle que $\forall e \in E$, $|\phi(e)| \in \{1, 2\}$.

Interprétation de la définition

Ici, les arêtes sont des objets à part.
La fonction ϕ prend comme argument une arête et renvoie un ensemble de sommets (les points extrêmes de l'arête).
Pour forcer une arête à avoir un ou deux points extrêmes, il faut une restriction sur la taille de l'ensemble renvoyé.
La façon d'exprimer cela est : $\phi : E \longrightarrow \mathcal{P}(V)$ telle que $\forall e \in E$, $|\phi(e)| \in \{1, 2\}$.

Le graphe en tant que type abstrait

Cette définition a des conséquences sur la programmation :

- à partir d'un objet de type *graphe*, récupérer l'ensemble des sommets du graphe,
- à partir d'un objet de type *graphe*, récupérer l'ensemble des arêtes du graphe,
- à partir d'une arête du graphe, récupérer le(s) sommet(s) extrémités de l'arête.

On appelle une collection d'opérations comme celle-ci un *type abstrait*.

La notion de type abstrait

C'est une notion centrale en programmation.
Cela permet de créer des programmes *modulaires* (i.e., contenant des parties relativement indépendantes) et donc *maintenables*.
Pour programmer une application, on se pose la question : "Quels sont les objets manipulés par le programme, et quelles sont les opérations sur ces objets ?"
La notion de type abstrait sera traitée en TD.

Définition d'un graphe (3)

Un *graphe* est un triplet (V, E, ψ), où V est un ensemble d'objets appelés les *sommets* du graphe, E est un ensemble d'objets appelés les *arêtes* du graphe et ψ est une fonction $\psi : V \longrightarrow \mathcal{P}(E)$ telle que $\forall e \in E, |\{s \in V,\ e \in \psi(s)\}| \in \{1, 2\}$.

Interprétation de la définition

Elle génère les mêmes objets que la précédente.
Le rôle de la fonction est totalement différente. Ici, elle est appliquée à un sommet et renvoie un ensemble d'arêtes.

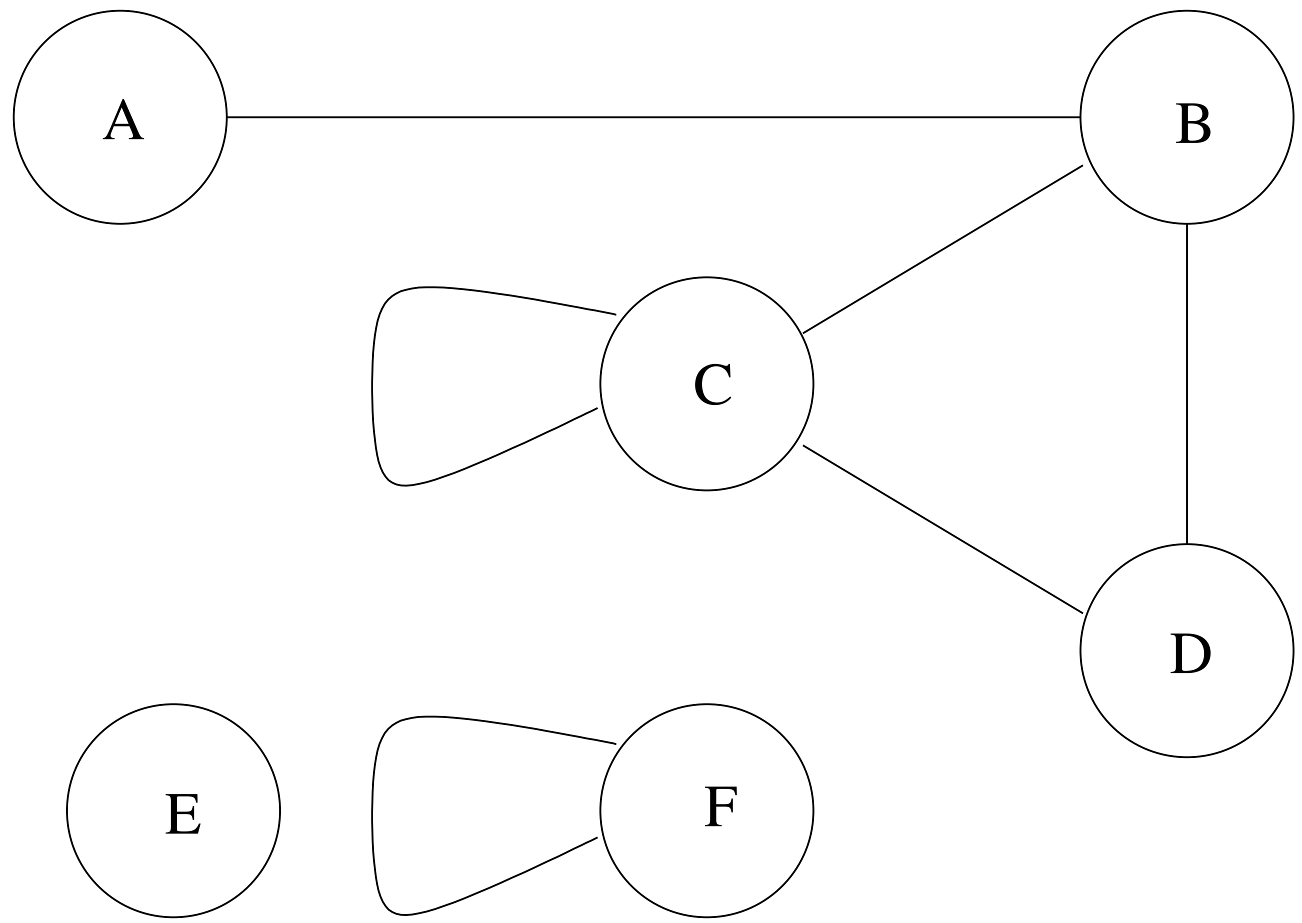
Quel type abstrait est le bon ?

Ça dépend de ce que vous voulez en faire (de l'application).
Certaines opérations sont plus rapides et/ou plus simples à programmer selon le type choisi.
Souvent, il n'est pas possible d'avoir que des opérations simples et rapides. Il faut donc choisir.

Degré d'un sommet

Le *degré* d'un sommet s, noté $d(s)$, est le nombre de brins d'arêtes ayant s comme extrémité.
Une boucle compte deux fois.

Exemple de degré



Ici $d(A) = 1$, $d(B) = 3$, $d(C) = 4$, $d(D) = 2$, $d(E) = 0$, $d(F) = 2$.

Un premier théorème

Pour un graphe G ayant au moins un sommet,

$$\sum_{s \in V(G)} d(s) \; = \; 2|E(G)|$$

Technique de preuve par induction

1. Vérifier que $\sum_{s \in V} d(s) = 2|E|$ pour un graphe sans arête,
2. Supposer que $\sum_{s \in V} d(s) = 2|E|$ pour n'importe quel graphe avec au plus k arêtes,
3. Prouver que si $\sum_{s \in V} d(s) = 2|E|$ est vrai pour n'importe quel graphe avec au plus k arêtes, c'est aussi vrai pour n'importe quel graphe avec $k+1$ arêtes.

La preuve du théorème (1/2)

Par induction sur le nombre d'arêtes dans le graphe.

- (cas de base) La propriété est trivialement vraie pour un graphe avec $|E| = 0$, car le degré de chaque sommet du graphe est 0.
- (hypothèse d'induction) On suppose que pour un graphe G avec au moins un sommet et au plus k arêtes, $\sum_{s \in V(G)} d(s) = 2|E(G)|$.

La preuve du théorème (2/2)

- (induction) Nous construisons un graphe G' en rajoutant une arête à G. Les sommets extrémités de la nouvelle arête, disons s_1 et s_2 (il est possible que s_1 et s_2 soient le même sommet) vont avoir leur degré augmenté de 1 chacun. Il est donc vrai que $\sum_{s \in V(G')} d(s) = \sum_{s \in V(G)} d(s) + 2$. De plus, nous avons rajouté une arête dans G'. Par conséquent $|E(G')| = |E(G)| + 1$. Donc, $\sum_{s \in V(G')} d(s) = \sum_{s \in V(G)} d(s) + 2 = 2|E(G)| + 2 = 2(|E(G)| + 1) = 2|E(G')|$. Nous avons donc $\sum_{s \in V(G')} d(s) = 2|E(G')|$

Nous avons donc prouvé la propriété par récurrence.

La notion de chaîne

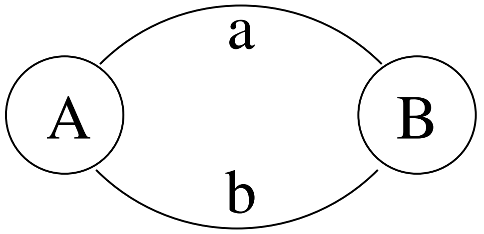
Il est souvent nécessaire de savoir si l'on peut aller d'un sommet à un autre en suivant des arêtes.
Exemple d'utilité : Est-ce possible de prendre le train pour aller de Bordeaux à Rome ? De Bordeaux à Oslo ? De Bordeaux à Reykjavik ?
La notion de chaîne exprime cette idée.

Définition de chaîne (1)

Première tentative :
(attention : cette définition n'est pas bonne).
Une *chaîne* dans un graphe est une suite
$C = s_1, s_2, \ldots, s_k$ de sommets du graphe, telle que
$\forall i$, $1 \leqslant i < k$, il y a une arête entre s_i et s_{i+1}.

Problème de la définition

S'il y a plusieurs arêtes entre deux sommets, on ne sait pas par laquelle il faut passer.
Exemple :

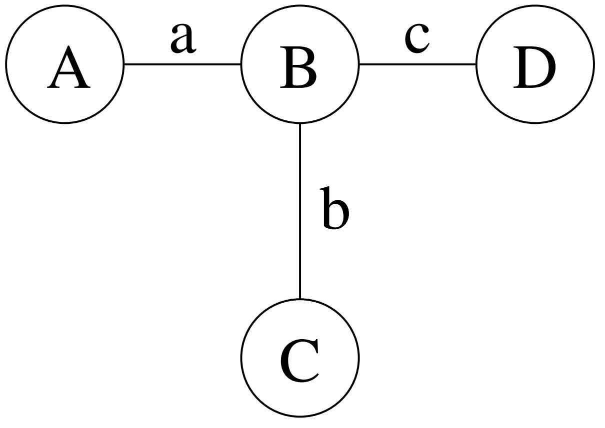


Définition de chaîne (2)

Deuxième tentative :
(attention : cette définition n'est pas bonne).
Une *chaîne* dans un graphe est une suite $C = e_1, e_2, \ldots, e_k$ d'arêtes du graphe, telle que $\forall i$, $1 \leqslant i < n$, e_i et e_{i+1} partagent un sommet.

Problème de la définition

La suite a, b, c dans le graphe suivant sera considérée comme une chaîne :



De plus, on ne sait pas par quel sommet la chaîne commence.

Définition de chaîne (3)

(la bonne)
Une *chaîne* dans un graphe est une suite

$$C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$$

de $k+1$ sommets et k arêtes en alternance, telle que $\forall i$, $1 \leqslant i \leqslant k$, les extrémités de e_i sont s_i et s_{i+1}. On dit alors que C *est une chaîne entre* s_1 *et* s_{k+1}.

Existence d'une chaîne

Vérifier s'il existe une chaîne entre un sommet s et un sommet t n'est pas forcément simple.
Pour y arriver, nous allons utiliser une technique pour marquer et démarquer les sommets.
Mais il nous faut d'abord un peu de théorie.

Chaîne simple

Une chaîne $C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$ est *simple* si et seulement si
$\forall i, j \in [1, k], s_i = s_j \Rightarrow i = j$ et
$\forall i, j \in [2, k+1], s_i = s_j \Rightarrow i = j$
Autrement dit “un sommet figure au plus une fois dans la chaîne”. [sauf pour le sommet de début et de fin. S’ils sont les mêmes, il s’agit d’un *cycle*]

Théorème

Dans un graphe G, s'il existe une chaîne entre $s \in V(G)$ et $t \in V(G)$, alors il existe une chaîne *simple* entre s et t.
La preuve est *constructive*. On prend une chaîne non simple et on supprime les cycles.

Preuve

Soit $C = s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}$ la chaîne.
Si C est simple, le travail est terminé.
Sinon, il y a deux sommets s_i et s_j tels que $i, j \in [1, k]$, pour lesquels $i \neq j$ et $s_i = s_j$ ou bien deux sommets s_i et s_j tels que $i, j \in [2, k+1]$, pour lesquels $i \neq j$ et $s_i = s_j$
En supposant que $i < j$ (sinon on renverse les rôles de i et j), on peut alors écrire :
$C = s_1, e_1, \ldots, s_i, e_i, \ldots, s_j, e_j, \ldots, s_k, e_k, s_{k+1}$.

Preuve (suite)

Construire C' plus courte en supprimant de C la partie $e_i, \ldots, s_j$.
On obtient alors :
$C' = s_1, e_1, \ldots s_i, e_j, \ldots, s_k, e_k, s_{k+1}$.
Mais comment pouvons-nous être sûr que C' est une chaîne ?
Chaque arête de la chaîne doit être entourée de ses deux extrémités.
C'est le cas dans C'.
Répéter tant que C' n'est pas simple.

Existence d'une chaîne

Voici la méthode (en TD, vous allez voir une méthode plus algorithmique et *récursive*) :

1. démarquer tous les sommets
2. marquer s
3. tant que t n'est pas marqué,
   (a) chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
   (b) si une telle arête n'existe pas, renvoyer la valeur "faux"
   (c) sinon marquer l'extrémité non encore marquée
4. renvoyer la valeur "vrai"

Existence d'une chaîne

Au pire, cet algorithme passe par chaque arête du graphe pour trouver la première arête dont un sommet extrémité est marqué et l'autre ne l'est pas, ce qui coûte $|E|$.
Ensuite, la deuxième arête est trouvée au pire en $|E| - 1$ étapes. Et ainsi de suite. L'algorithme s'arrête quand tous les sommets ont été marqué (il n'y a plus rien à faire !). Donc au pire, à l'étape $|V|$, tous les sommets ont été marqué.
La complexité de cet algorithme est donc au plus :

$$\sum_{i=0}^{|V|} (|E| - i) \ = \ O(|V| \cdot |E|) \ .$$

(En TD vous verrez un algorithme en $O(|V|)$)

Connexité

La notion de connexité exprime la possibilité d'aller de n'importe quel sommet du graphe à n'importe quel autre sommet du graphe.
Plus formellement :
Un graphe G est connexe si et seulement si $\forall s, t \in V(G)$, il existe une chaîne entre s et t.
Nous allons étudier des méthodes efficaces pour déterminer si un graphe est connexe.

Approche simple

La première idée est toujours d'appliquer la définition.
Donc, pour déterminer si un graphe est connexe, vérifier si pour chaque couple (s, t) de sommets, il existe une chaîne entre s et t. Il y a n^2 tels couples, si $n = |V|$.
Au total, la méthode simple a une complexité $O(n^3 \cdot |E|)$ (voir $O(n^3)$ si la version TD est utilisée).
Nous allons montrer que c'est possible de le faire en $O(n)$.

Différence entre $O(n^3)$ et $O(n)$

Pour apprécier la différence entre $O(n^3)$ et $O(n)$, imaginons un ordinateur capable d'exécuter une instruction élémentaire en 10ns (raisonnable).
Imaginons aussi un graphe de $100\,000$ sommets (un réseau routier par exemple).
Exécuter n instructions élémentaires nécessite 1ms, soit $0,001$s.
Exécuter n^3 instructions élémentaires nécessite 10^7s soit environ 4 mois.

Approche plus efficace

Il nous faut encore un théorème :
Un graphe est connexe si et seulement si à partir de n'importe quel sommet s du graphe, il existe une chaîne entre s et chacun des sommets du graphe.

Preuve (partie “si”)

La preuve est par contradiction.
Nous allons supposer qu’à partir de n’importe quel sommet s du graphe, il existe une chaîne entre s et chacun des sommets du graphe, mais, $\exists s_1, s_2 \in V(G)$, tels qu’il n’existe pas de chaîne entre s_1 et s_2, puis montrer que cela nous donne un résultat absurde.
Il suffit pour cela de construire les chaînes $C_1 = s, \ldots, s_1$ et $C_2 = s, \ldots, s_2$.
Puis, on regarde $\overline{C_1}C_2 = s_1, \ldots, s, \ldots, s_2$ qui est une chaîne entre s_1 et s_2.

Preuve (partie “seulement si”)

Il faut prouver que si un graphe est connexe, alors à partir de n’importe quel sommet s du graphe, il existe une chaîne entre s et chacun des sommets du graphe. C’est une conséquence immédiate de la définition de connexité.

Un algorithme plus efficace

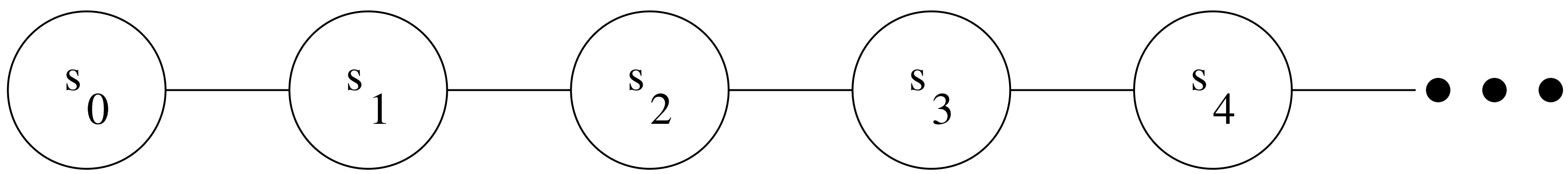
Pour déterminer si un graphe est connexe, on peut donc faire la chose suivante :
Choisir un sommet s arbitraire dans le graphe.
Vérifier si pour chaque sommet t du graphe, il existe une chaîne entre s et t.
Il faut donc faire $O(n)$ vérifications. Le coût de chaque vérification est $O(n)$ (algorithme version TD).
Au total, la complexité est $O(n^2)$.

Différence entre $O(n^3)$, $O(n^2)$ et $O(n)$

Imaginons encore un ordinateur capable d'exécuter une instruction élémentaire en 10ns et un graphe de $100\,000$ sommets.
Exécuter n instructions élémentaires nécessite 1ms, soit $0,001$s.
Exécuter n^2 instructions élémentaires nécessite 10^2s soit environ 2 minutes.
Exécuter n^3 instructions élémentaires nécessite 10^7s soit environ 4 mois.

Encore une amélioration

On observe que l'algorithme parcourt la même chaîne plusieurs fois.
Dans le graphe suivant :



il va parcourir entre s_0 et s_1, puis entre s_0 et s_2, refaisant alors le parcours entre s_0 et s_1, puis entre s_0 et s_3, refaisant alors le parcours entre s_0 et s_2, refaisant alors le parcours entre s_0 et s_1, *etc*.

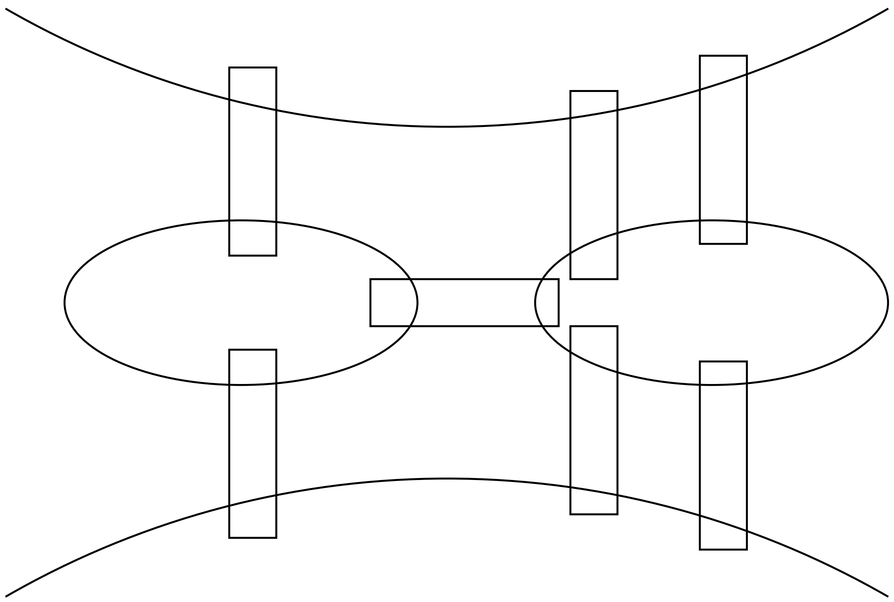
Méthode finale pour la connexité

Voici la méthode (en TD, vous allez voir une méthode plus algorithmique et *récursive*) :

1. démarquer tous les sommets
2. marquer un sommet arbitraire s
3. chercher une arête dont un sommet extrémité est marqué et l'autre ne l'est pas
4. tant qu'une telle arête existe :
   (a) marquer l'extrémité non encore marquée
5. si tout les sommets sont marqués, renvoyer “vrai”
6. sinon renvoyer “faux”

Graphes eulérien

Les ponts de la ville de Königsberg (maintenant Kalniningrad) :



Promenade

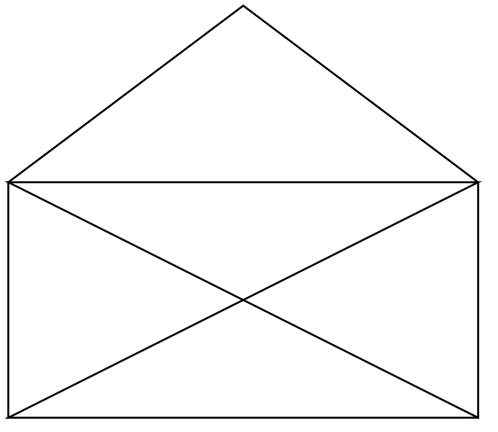
Peut-on commencer une promenade sur une île ou une rive, terminer la promenade sur n'importe quelle autre (ou la même) île ou rive en passant exactement une fois sur chacun des ponts ?
C'est le mathématicien Euler qui a trouvé la réponse (d'où le nom "eulérien").

C’est un problème de graphe

Dans un graphe G, est-il possible de trouver une chaîne $C = s_1, e_1, s_2, e_2, \ldots, s_m, e_m, s_{m+1}$ telle que $\{e_1, e_2, \ldots, e_m\} = E(G)$ et $|E(G)| = m$.

Dessiner une enveloppe

Est-ce possible de dessiner cette enveloppe sans lever le crayon :



Définition de graphe eulérien

Un graphe G est *eulérien* si et seulement si il est possible de trouver une chaîne $C = s_1, e_1, s_2, e_2, \ldots, s_m, e_m, s_{m+1}$ telle que $\{s_1, s_2, \ldots, s_m, s_{m+1}\} = V(G)$, $\{e_1, e_2, \ldots, e_m\} = E(G)$ et $|E(G)| = m$.

Conditions nécessaires

Théorème : Un graphe eulérien est forcement connexe.
D'autres façons de dire la même chose :

- Tout graphe eulérien est connexe,
- Si un graphe est eulérien, alors il est connexe,
- G eulérien $\Rightarrow$ G connexe,
- Si un graphe n'est pas connexe, alors il n'est pas eulérien,
- G est connexe est une condition nécessaire pour que G soit eulérien.

Preuve du théorème

Par contradiction. On suppose l'existence d'un graphe G eulérien non connexe.
Le graphe G est eulérien. Donc, il existe une chaîne $C = s_1, e_1, s_2, e_2, \ldots, s_m, e_m, s_{m+1}$ telle que $\{s_1, s_2, \ldots, s_m, s_{m+1}\} = V(G)$,
$\{e_1, e_2, \ldots, e_m\} = E(G)$ et $|E(G)| = m$.
Le graphe n'est pas connexe. Donc, $\exists s_1, s_2 \in V(G)$, tels qu'il n'existe aucune de chaîne entre s_1 et s_2.
Nous allons appeler ces sommets A et B.

Preuve (suite)

La chaîne C contient tous les sommets, y compris A et B.
Disons que $A = s_i$ et $B = s_j$.
On peut donc écrire la chaîne C comme ceci :
$C =$
$s_1, e_1, s_2, e_2, \ldots, A, e_i, \ldots, e_{j-1}, B, \ldots, s_m, e_m, s_{m+1}$.
Examinons la sous-chaîne : $C' = A, e_i, \ldots, e_{j-1}, B$ de C.
C' est bien une chaîne entre A et B. Nous avons une contradiction.

Parité des sommets

Définition : un sommet est *pair* si son degré est pair.
Un sommet est *impair* si son degré est impair.

Encore une condition nécessaire

Théorème : Dans un graphe eulérien, le nombre de sommets impairs est forcement 0 ou 2.
La ville de Königsberg n'est donc pas eulérien. Pour ce qui concerne l'enveloppe, le théorème ne nous donne pas le droit de constater qu'elle est eulérien.

Preuve

Par contradiction : On suppose l'existence d'un graphe eulérien dont le nombre de sommets impairs n'est ni 0 ni 2.
Le graphe est eulérien. Donc, il existe une chaîne $C = s_1, e_1, s_2, e_2, \ldots, s_m, e_m, s_{m+1}$ telle que $\{s_1, s_2, \ldots, s_m, s_{m+1}\} = V(G)$,
$\{e_1, e_2, \ldots, e_m\} = E(G)$ et $|E(G)| = m$.

Preuve (suite)

Examinons un sommet autre que s_1 et s_{m+1} dans cette chaîne. Un tel sommet s_i est entouré de e_{i-1} et e_i ce qui donne une contribution de 2 au degré de s_i. Le degré de s_i est donc pair.
Pour s_1 et s_{m+1} il existe deux cas : $s_1 \neq s_{m+1}$ et $s_1 = s_{m+1}$
Premier cas : s_1 et s_{m+1} sont impairs. Il y a donc 2 sommets impairs.
Deuxième cas : $s_1 = s_{m+1}$ est pair. Il y a donc 0 sommets impairs.

La preuve nous donne plus

Cette preuve nous indique comment trouver une chaîne eulérienne dans un graphe ayant deux sommets impairs : il faut commencer dans un sommet impair et terminer dans l'autre sommet impair.

Conditions suffisantes

Sans preuve :
Théorème : Un graphe G est eulérien si et seulement si G est connexe et le nombre de sommets pairs de G est 0 ou 2.