Mathématiques appliquées à l'informatique

- BA2 Informatique

Johan Depréter – johan.depreter@heh.be

Informations pratiques

- Répartition des séances :
  - Théorie : +/- 16h
  - Pratique : +/- 8h

- Evaluations :
  - 20% de travail continu : Exercices à rendre en séances d'exercices + évaluation continue
  - 80% examen final : Examen oral avec préparation.

Conseils

- Dès qu'on rentrera dans le vif du sujet, essayez de travailler régulièrement.
- Préparez, et finissez, les séances de pratique.
- Les maths c'est pas toujours simple, mais il faut persévérer.

Pourquoi ?

Pourquoi ?

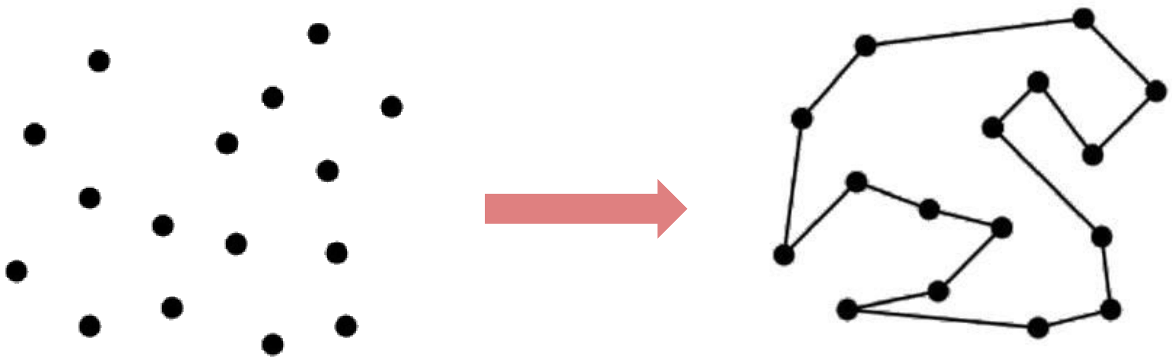
- Remember Algo
- Rigueur et esprit critique
- Socle nécessaire pour les domaines avancés de l'informatique : Cryptographie, Big Data, IA, Optimisation, etc

Rappel ?

- C’est quoi un algorithme ?
- Les preuves
- L’efficacité

Exemple

- On dispose d'un robot équipé d'un fer à souder.
- On doit programmer le robot pour qu'il passe par tous les points de contacts, dans un certain ordre.
- On cherche le chemin le plus optimisé, c'est-à-dire le chemin qui minimise la distance parcourue par le bras du robot



Tour du voisin le plus proche

Algorithme :

Input: Une collectio

Output: Un tour pa

Visiter un point *p*0 i

i ← 0

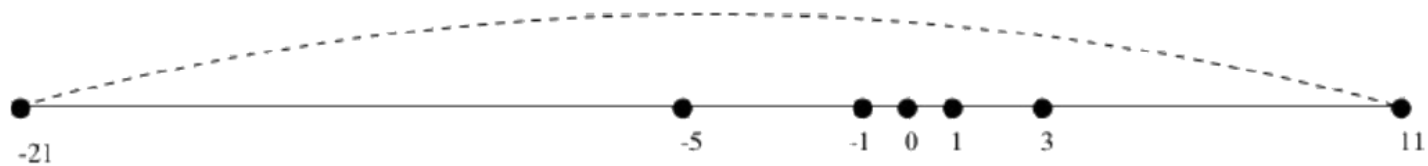
while *il y a des poin*

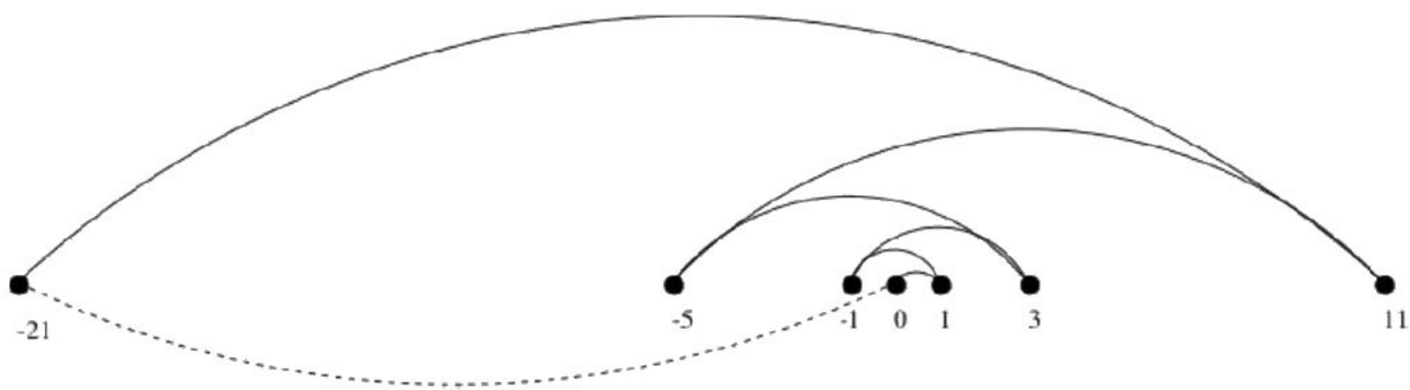
i ← *i* + 1

Visiter *pi*,

return *le tour selon*

nale


-21
-5
-1
0
1
3
11


-21
-5
-1
0
1
3
11

Autre solution

- On pourrait énumérer toutes les permutations possibles de *n* points.
  - Une permutation est un ordonnancement possible de *n* points.

- Pour chaque permutation, calculer la distance associée.

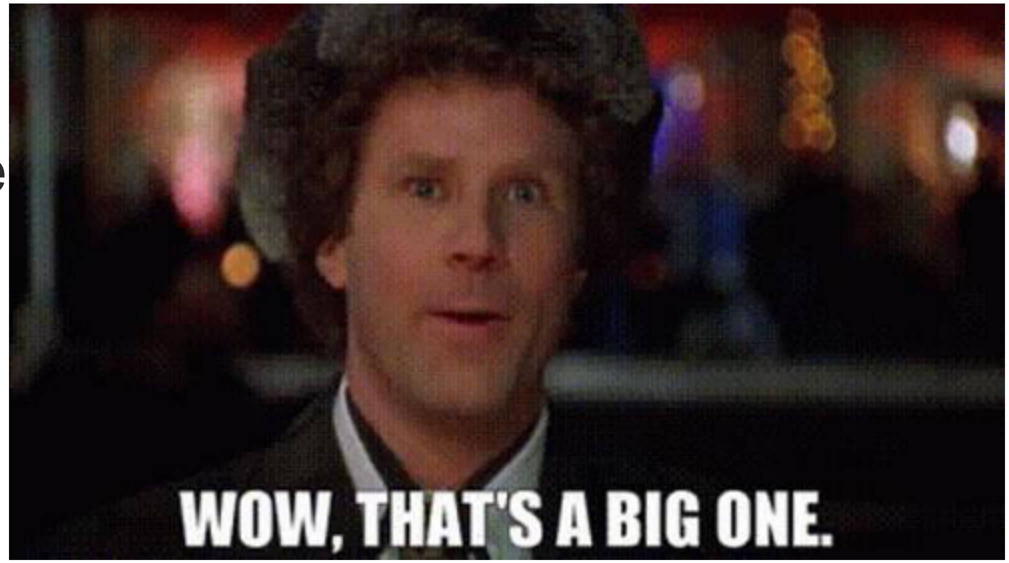
- Enregistrer le tour associé à la permutation de plus petite distance.

Recherche exhaustive

- Si on suit cette méthode, on est sur de trouver la solution optimale.

- L’algorithme est donc exact.

Recherche exhaustive

- Le nombre  n éléments.

- Exemples :

$$10! = 1 * 2 * 3 * \cdots * 10 = 3\,628\,800$$
$$20! = 1 * 2 * 3 * \cdots * 20 = 2\,432\,902\,008\,176\,640\,000 \approx 2 * 10^{18}$$
$$100! \approx 9 * 10^{157}$$
$$500! \approx 1 * 10^{1134}$$

Oui, mais ma machine…

- Frontier est le plus gros super calculateur au monde (mai 2023)

  1.194 exaflops/s

- Ca représente :

1 100 000 millions de millions d'opérations à la seconde.
$(1.194 * 10^{18})$

Frontier

- Déjà c’est un peu cher juste pour un robot à souder…
- Frontier aurait besoin de $2.6 * 10^{133}$ années pour calculer 100! Possibilités ($9 * 10^{157}$)

Age de l’univers	Calcul en années
13 800 000 000 années	26 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 années

Le voyageur de commerce

- Il n’existe pas d’algorithme exact et efficace pour résoudre ce problème.
- On va utiliser des algorithmes qui n’ont pas la garantie de donner une solution optimale (Heuristique)
- C’est ce qu’on appelle un problème NP-complet

On fait quoi avec tout ça ?

- Exactitude ≠ Optimalité

- Ce qu’on va chercher c’est l’exactitude même si on peut souvent reformuler des problèmes sous forme d’optimisation.

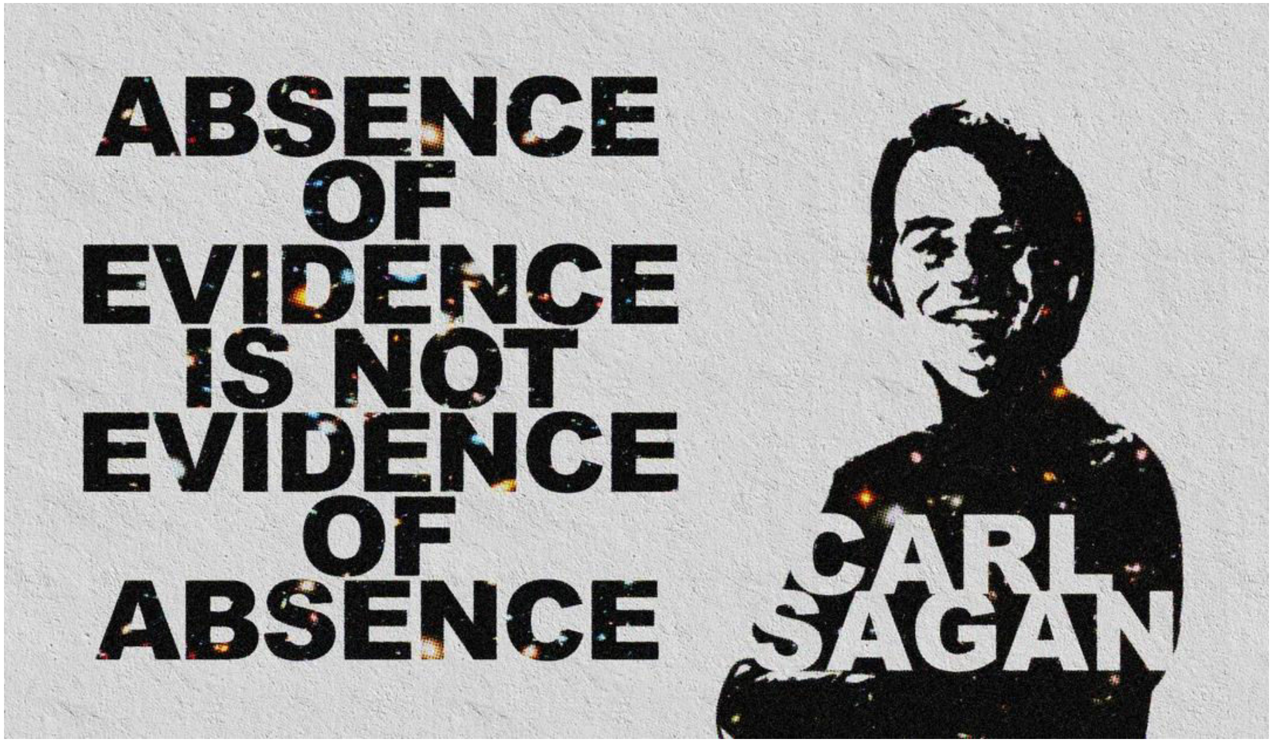
Prouver qu'un algorithme est inexact

Recherche d'un contre-exemple

- Un algorithme doit TOUJOURS fonctionner (selon sa spécification)
- Il suffit d'un contre-exemple pour prouver qu'il est inexact
  - Exemple de petite taille
  - Cas d'égalité dans la décision
  - Cas limites

Prouver qu'un algorithme est exact

⛔ Ne pas trouver de contre-exemple ne rend pas un ⛔ algorithme exact.



Prouver qu'un algorithme est exact

- Pas de

- Il faudrait aussi prouver l'efficacité…

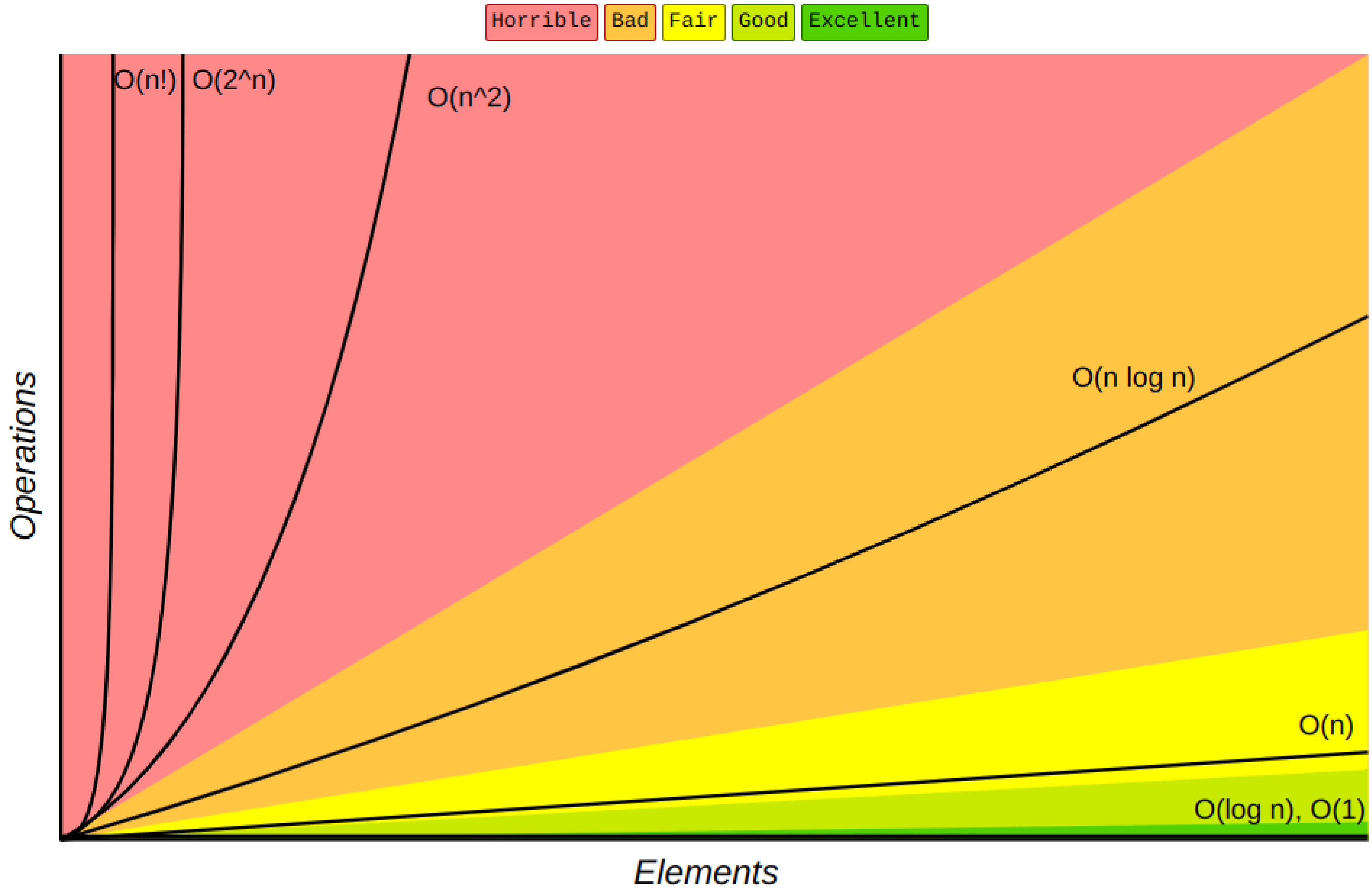
L'efficacité

- On va quantifier et caractériser l'efficacité selon deux grandeurs :
  - Le temps de calcul → La complexité temporelle
  - L'espace utilisé → La complexité spatiale
- Normalement c'est sensé vous rappeler quelque chose…

Complexité calculatoire

La complexité

Notation	Type de complexité
$O(1)$	complexité constante (indépendante de la taille de la donnée)
$O(\log(n))$	complexité logarithmique
$O(n)$	complexité linéaire
$O(nlog(n))$	complexité quasi linéaire
$O(n^2)$	complexité quadratique
$O(n^3)$	complexité cubique
$O(n^p)$	complexité polynomiale
$O(n^{\log(n)})$	complexité quasi polynomiale
$O(2^n)$	complexité exponentielle
$O(n!)$	complexité factorielle


Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Analyse asymptotique

- Objectif :

  Analyser le temps (ou l'espace) requis, en se concentrant sur l'algorithme et l'influence de la taille du problème, généralement dans le pire cas

- On va s'intéresser à l'évolution de la complexité lorsque la taille du problème augmente $(\lim_{n \to \infty})$

Comment le temps évolue ?

Par exemple, si la taille n du problème est multipliée par 10 comment évolue le temps $T = f(n)$?

Si $f(n) = c \Rightarrow f(10n) = c$	T
Si $f(n) = c.n \Rightarrow f(10n) = c * (10n) = 10f(n)$	$T \times 10$
Si $f(n) = c.n^2 \Rightarrow f(10n^2) = c * (10n)^2 = 100f(n)$	$T \times 100$

- La vitesse du processeur est un des facteurs qui conditionnent la valeur de la constante c
- Mais la vitesse du processeur ne change rien au rapport

$$f(10n) \big/ f(n)$$

En résumé

- Dans ce type de problème, la constante correspond à tout ce qui ne dépend pas de la taille du problème même si elle peut varier !

- On peut, dans notre cas, négliger les constantes !

Opération primitive

- C’est une instruction en langage de haut niveau
- Elle représente un nombre constant d’opérations élémentaires exécutées par le processeur
- Exemples :
  Une affectation, une comparaison, un embranchement, une opération arithmétique, un accès à un tableau, un return, un appel de fonction, …

La complexité temporelle

- Notation dite « Big O » → $O(f(n))$

- On ne retient que la grandeur :
  Exemple : $O(3n^2 + 5n + 7) \rightarrow O(n^2)$

- Indépendante :
  - du langage de programmation utilisé
  - du processeur de l'ordinateur sur lequel sera exécuté le code
  - de l'éventuel compilateur employé.

Comment on la calcule ?

- Compter le nombre d'opérations primitives

  $a = b * 3$ ➜ 1 multiplication + 1 affectation = 2 'unités'

- La taille des données, notée ***n***

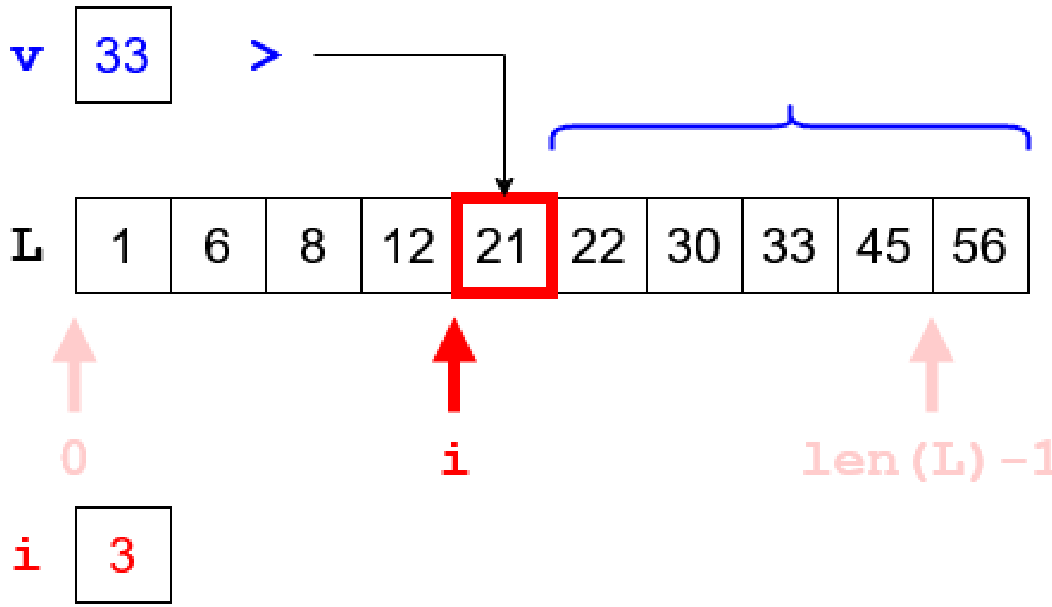
- La donnée en question

  Calcul dans le meilleur des cas

  Calcul dans le pire des cas

  Calcul dans le cas moyen

Cas pratique – Recherche dichotomique



Cas pratique – Recherche dichotomique

- Prenons un tableau de taille n
- Si on le coupe, on obtient : $n_1 = \frac{n}{2}, n_2 = \frac{n_1}{2} \dots$
- On va finir par avoir $n_i = 1$
- On obtient alors : $n_i = 1 = \frac{n}{2*2*2*2*2\dots*2} = \frac{n}{2^a} \rightarrow a = log_2(n)$
- La complexité temporelle est donc : $O(log_2(n))$

Mathématiques appliquées à l'informatique

- BA2 Informatique
  Johan Depréter – johan.depreter@heh.be

- Complexité calculatoire

Opération primitive

- C’est une instruction en langage de haut niveau

- Elle représente un nombre constant d’opérations élémentaires exécutées par le processeur

- Exemples :

  Une affectation, une comparaison, un embranchement, une opération arithmétique, un accès à un tableau, un return, un appel de fonction, …

Temps d'exécution

- On peut imaginer qu'une opération primitive se fait en 1 nanoseconde.

- Si n = 1000 combien de temps prend le programme pour s'exécuter en fonction de f(n)

F(n)	Temps
n	0.000 001 s
400 n	0.000 4 s
$2n^2$	0.002 s
n^5	≈ 11.5 jours

Notions de cas

- Prenons un exemple concret.

On a un algorithme suivant qui permet de renvoyer le premier entier négatif d'une liste de n entiers.

```
A = []

def FirstNega(A):
    for x in A:
        if x < 0:
            return x
```

Il se passe quoi si A vaut [-1]; [2,4,-3,6]; [5,9,4,7,-8]

Les instances

- Les instances sont importantes pour l'efficacité des algorithmes.
- Infinité d'instances
- On va classer selon si on est dans le meilleur, pire, moyen cas.

Identifier le pire cas

- Toujours considérer des problèmes d’une même taille
- Travailler avec des données spécifiques
- Chercher les limites :
  - Exemples de petites tailles ;
  - Cas d’égalité ;
  - Très grands, très petits, déjà trié en sens inverse, que des valeurs négatives, etc

Calculer le nombre d'opérations primitives

Algorithme : arrayMax

Entrée : A un tableau de n entiers ($n > 0$)

Sortie : La valeur maximale dans A

$currentMax \leftarrow A[0]$	// 2 opérations
for $i \leftarrow 1$ **to** $n - 1$ **do**	// $1 + 2.(n - 1) + 2.n$ op
if $currentMax < A[i]$ **then**	// 3 op., exécutées $n - 1$ fois
$currentMax \leftarrow A[i]$	// 2 op., exécutées au plus $n - 1$ fois
return $currentMax$	// 1 opération

Dans le pire des cas (A en ordre croissant) → 9n – 3
Dans le meilleur (A en ordre décroissant) → 7n - 1

Lequel choisir ?

- Si on devait d[…] le précédent on aurait 2 po[…]

  F(n) = 7n – 1[…]

  F(n) = 9n – 3[…]

- Borne supérie[…]

Notation ϑ

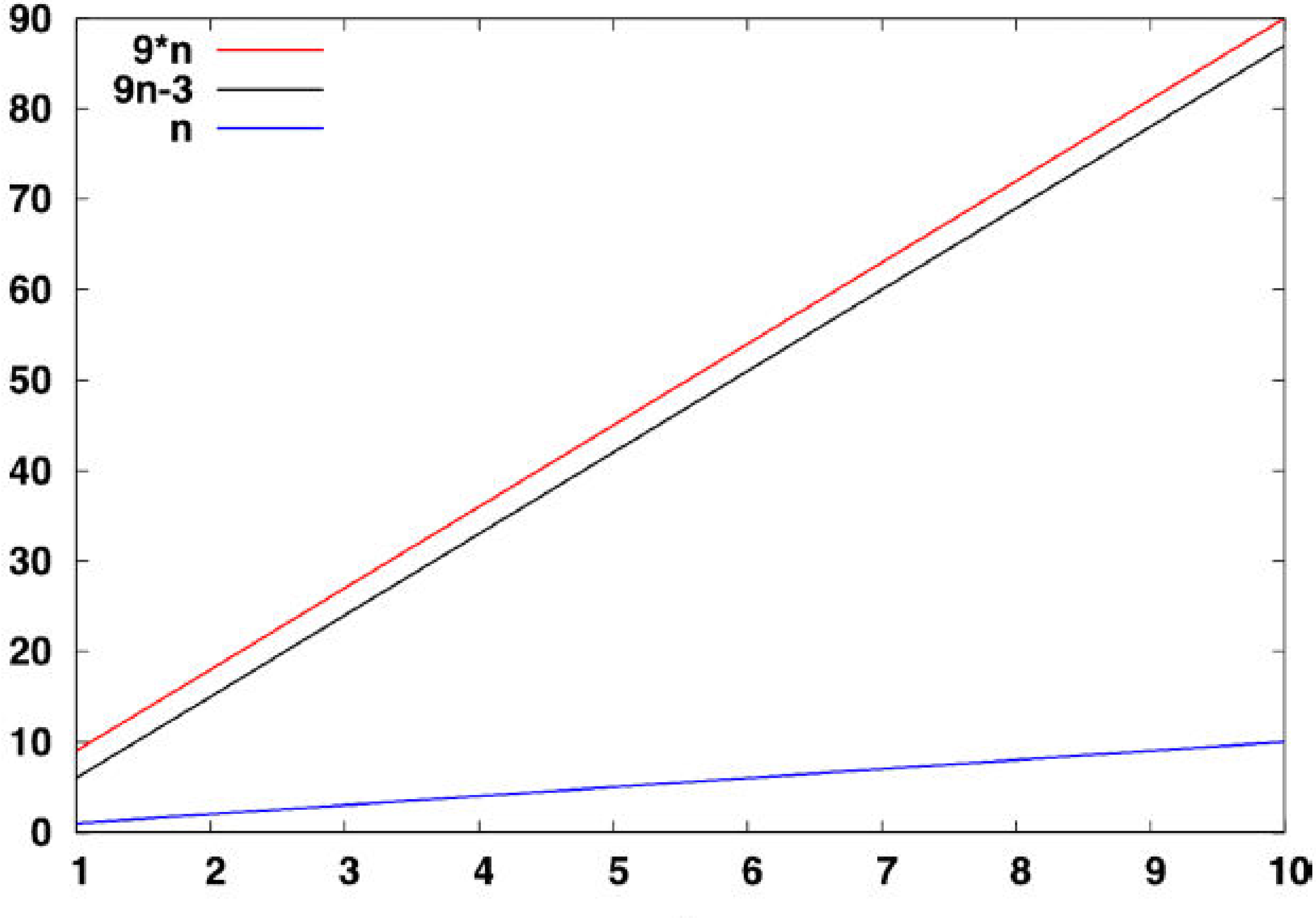
- « Grand O »

  $f(n) \in O(g(n))$ si $\exists\, c > 0$, $\exists\, n0 \geq 1$ tels que $f(n) \leq c.g(n)$, $\forall n \geq n0$

- Autrement dit :

  G(n) est une borne supérieure de f(n)

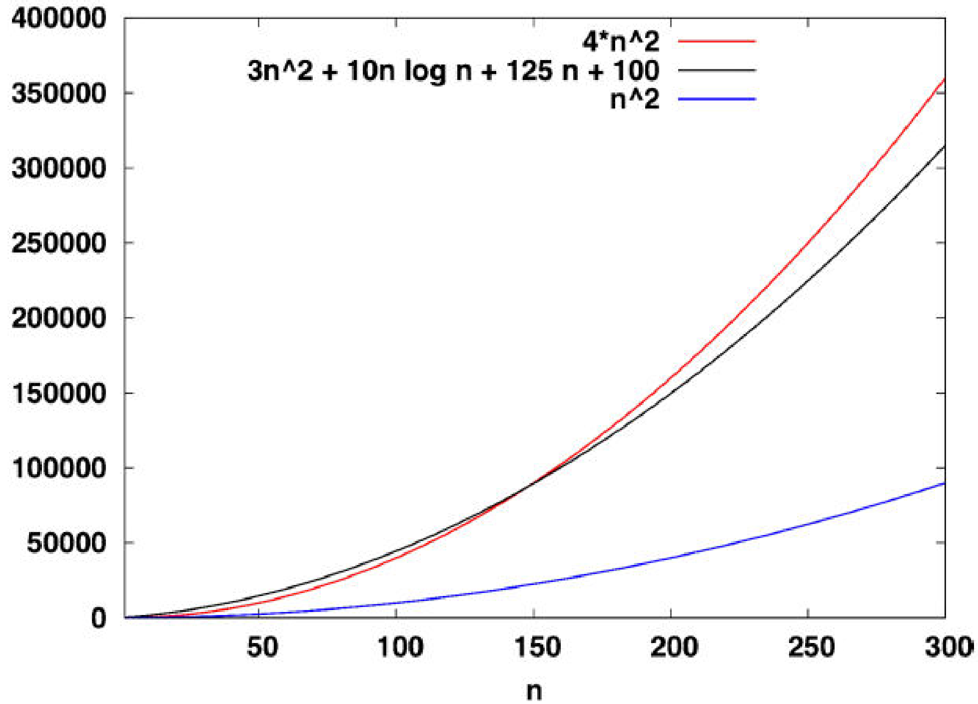
  Elle se « situe » à une constante multiplicative près


9*n
9n-3
n
0
10
20
30
40
50
60
70
80
90
1
2
3
4
5
6
7
8
9
10

Notation ϑ

- On va s'intéresser à la borne la plus simple et strict possible
- Dans l'exemple précédent, n est une borne supérieure de 9n – 3 à une constante multiplicative près
- On ne garde que les termes dominants et on supprime les constantes.

HEH.be Sciences et technologies



$$3n^2 + 10n\log_{10} n + 125n + 100 \in \mathcal{O}\left(n^2\right) \text{ car}$$
$$3n^2 + 10n\log_{10} n + 125n + 100 \leq 4.n^2 \text{ pour } n \geq 148$$

D’autres notations

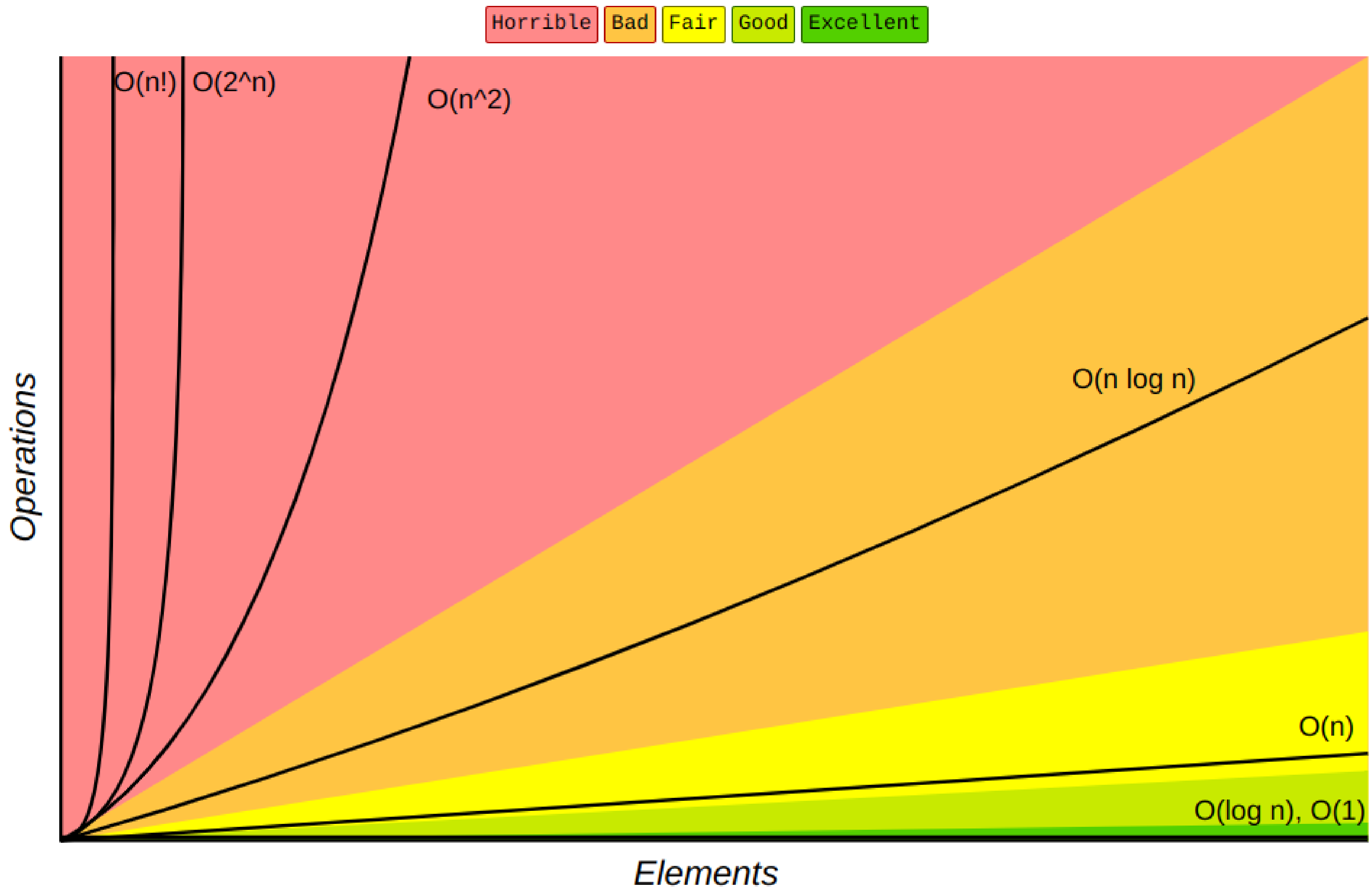
- Notation Ω
  - Borne inférieure
- Notation Θ
  - Borne exacte

Les boucles imbriquées

```python
n = 100

for i in range(0,n):
    pass
    for j in range(0, n):
        pass
```

≠

```python
n = 100

for i in range(0,n):
    pass

for j in range(0, n):
        pass
```


Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

La complexité

Notation	Type de complexité
$O(1)$	complexité constante (indépendante de la taille de la donnée)
$O(\log(n))$	complexité logarithmique
$O(n)$	complexité linéaire
$O(nlog(n))$	complexité quasi linéaire
$O(n^2)$	complexité quadratique
$O(n^3)$	complexité cubique
$O(n^p)$	complexité polynomiale
$O(n^{\log(n)})$	complexité quasi polynomiale
$O(2^n)$	complexité exponentielle
$O(n!)$	complexité factorielle

HEH.be Sciences et technologies

- Exercices

Tri à bulles

- Réaliser le calcul de complexité du tri à bulles en utilisant l'algorithme suivant :

```
tri_à_bulles(Tableau T)
   pour i allant de (taille de T)-1 à 1
       pour j allant de 0 à i-1
           si T[j+1] < T[j]
               (T[j+1], T[j]) ← (T[j], T[j+1])
```

Tri par insertion

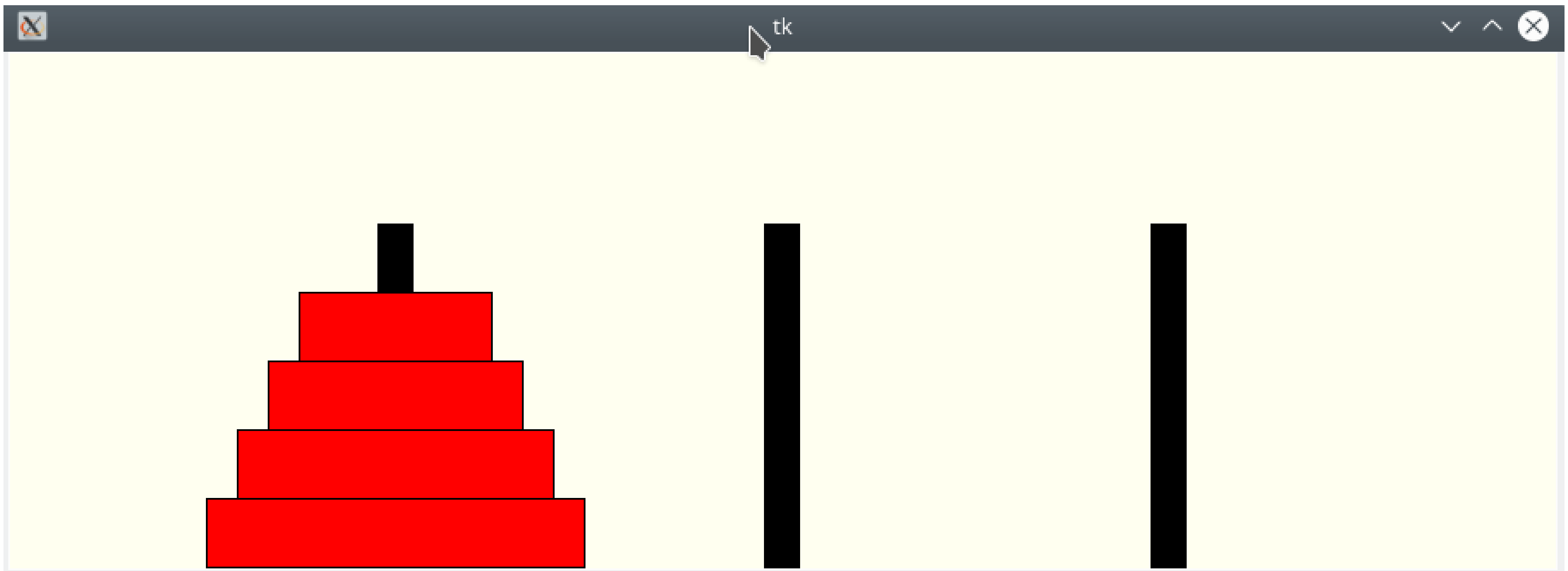
- Réaliser le calcul de complexité du tri par insertion en utilisant l'algorithme suivant :

```
procédure tri_insertion(tableau T)

    pour i de 1 à taille(T) - 1

        # mémoriser T[i] dans x
        x ← T[i]

        # décaler les éléments T[0]..T[i-1] qui sont plus grands que x, en partant de T[i-1]
        j ← i
        tant que j > 0 et T[j - 1] > x
                 T[j] ← T[j - 1]
                 j ← j - 1

        # placer x dans le "trou" laissé par le décalage
        T[j] ← x
```

HEH.be Sciences et technologies

- La récurrence

Les tours de Hanoï



Les tours de Hanoï

- On va imaginer le problème pour n disques.
- On aura T_n qui est le nombre minimum de coups pour transférer n disques.
- On peut assez facilement déterminer qu'alors :

$$T_1 = 1$$
$$T_2 = 3$$

Et plus grand ?

- A 3 disques, on va :
  1. Transférer les 2 premiers disques sur un piquet
  2. Déplacer le disque 3
  3. Retransférer les 2 premiers disques sur le piquet du disque 3

- Comment on fait pour n disques alors ?
  1. Transférer les n-1 disques sur un piquet (T_{n-1})
  2. Déplacer le plus grand disque (1 déplacement)
  3. Retransférer les n-1 disques sur le piquet (T_{n-1})

- On obtient donc : $2 * T_{n-1} + 1$

Pour n disques

- $T_n \leq 2 * T_{n-1} + 1 \; pour \; n > 0$

- On peut assez facilement montrer qu'il n'y a pas moins de mouvement possible

- On peut donc écrire : $T_n \geq 2 * T_{n-1} + 1 \; pour \; n > 0$

Récurrence

- On a donc :
  - $T_n \leq 2 * T_{n-1} + 1 \, pour \, n > 0$
  - $T_n \geq 2 * T_{n-1} + 1 \, pour \, n > 0$

- On peut aussi affirmer assez facilement que $T_0 = 0$

- On pourra alors écrire la récurrence suivante :

$$T_0 = 0$$
$$T_n = 2 * T_{n-1} + 1 \, pour \, n > 0$$

Application et vérifications

- On a le système d'équations de récurrence suivantes :

$$T_0 = 0$$
$$T_n = 2 * T_{n-1} + 1 \; pour \; n > 0$$

- $T_1 = 2 * T_0 + 1 = 1 \; (OK)$
- $T_2 = 2 * T_1 + 1 = 2 * 1 + 1 = 3 \; (OK)$
- $T_3 = 2 * T_2 + 1 = 2 * 3 + 1 = 7$
- …
- $T_{42} = 2 * T_{41} + 1 = ?$

Récurrence

- Le système d'équations de récurrence est indirecte et locale.
- On veut trouver une expression qui ne dépend de rien d'autre que T_n
- Une solution :
  - « Deviner la solution correcte »
  - Prouver que la supposition est correcte

Récurrence

- Ici, pour deviner on va de nouveau s'intéresser aux petits exemples :

$$T_2 = 3$$
$$T_3 = 2 * 3 + 1 = 7$$
$$T_4 = 2 * 7 + 1 = 15$$
$$T_5 = 2 * 15 + 1 = 31$$

- On dirait que $T_n = 2^n - 1 \; pour \; 0 < n \leq 6$

Induction

- C’est le fait de prouver une affirmation générale à partir de cas particuliers

- Très efficace pour prouver l’exactitude d’un algorithme récursif

- On va prouver que la première étape est correcte, et qu’a chaque étape on peut toujours passer à la suivante selon le même principe

HEH.be
Sciences
et technologies

Mathématiques appliquées à l'informatique

- BA2 Informatique
Johan Depréter – johan.depreter@heh.be

- Les preuves

La démonstration

Le but sera d'établir la véracité d'une proposition par une chaîne de déductions logiques à partir d'une liste d'axiomes

Une proposition c'est une assertion qui est soit vraie, soit fausse.

$$1 + 1 = 2 \; en \; base \; 10$$
$$\forall n \in \mathbb{N}: P(n) = n^2 + n + 41 \; est \; un \; nombre \; premier$$

P(n) dépend de plusieurs variable → Prédicat

D’autres exemples

$a^4 + b^4 + c^4 = d^4$ n’a pas de solutions dans les entiers positifs $\mathbb{N}_0 = \{1, 2, 3, 4, \dots\}$

$$Si\ a = 95800, b = 217519, c = 414560, et\ d = 422481$$

$$\exists a, b, c, d \in \mathbb{N}_0\ tel\ que\ a^4 + b^4 + c^4 = d^4$$

$313(x^3 + y^3) = z^3$ n’a pas de solutions dans $\mathbb{N}_0$

Conclusion

Se contenter de vérifier les premières valeurs de x ou des valeurs de x au hasard n'est pas suffisant pour affirmer que $\forall x : P(x)$

Les axiomes

Un axiome est une proposition qui est vraie par supposition.

Ex : Si $a = b$ et $b = c$ alors $a = c$

Les axiomes dépendent du contexte.

Il devraient être cohérents et complets, mais ça n’existe pas.

Preuve par induction

Utilisation d'un axiome d'induction :

$$Soit\ P(n), un\ prédicat\ avec\ n \in \mathbb{N}.$$
$$Si\ P(0) est\ vraie\ ET\ si$$
$$\forall n \in \mathbb{N}: \big(P(n) \Rightarrow P(n+1)\big)\ est\ vraie,$$
$$alors\ \forall n \in \mathbb{N}: P(n)\ est\ vraie.$$

Exemple

$$\forall n \in \mathbb{N}: \sum_{i=1}^{n} i = 1 + 2 + 3 + 4 + \cdots + n = \frac{n(n+1)}{2}$$

Cas de base :

$$n = 0 : 1 + 2 + 3 + \cdots + \mathrm{n} = 0 = \frac{0(0+1)}{2}$$

$$n = 1 : 1 + 2 + 3 + \cdots + \mathrm{n} = 1 = \frac{1(1+1)}{2}$$

Exemple

Intuition :

$$\begin{array}{ccccccccccc} 1 & + & 2 & + & 3 & + \cdots + & n & + \\ n & + & n-1 & + & \mathrm{n}-2 & + \cdots + & 1 & = \\ n+1 & + & n+1 & + & n+1 & + \cdots + & n+1 & \end{array}$$

Théorème :

$$\forall n \in \mathbb{N}: \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Induction

Démo : Par induction, soit

$$P(n): \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Cas de base: (n=0)

On a

$$\sum_{i=1}^{0} i = 0 = \frac{0(0+1)}{2}$$

P(0) est vraie.

Induction

Cas inductif: $\forall n \in \mathbb{N}$ on doit démontrer que $P(n) \Rightarrow P(n+1)$ est vraie

On va supposer que $P(n)$ est vraie et donc :

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

On va démontrer que :

$$\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$$

Induction

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \quad \cdots \quad \frac{n(n+1)}{\quad} + (n+1) = \frac{(n+1)(n+2)}{2}$$

Par induction … $\in \mathbb{N}$.

$2^{ème}$ exemple

Théorème : Tous les chats ont la même couleur

Démo : Par induction, soit

$P(n)$: Pour tout ensemble de n chats, les chats ont la même couleur.

Cas de base: (n=1) $P(1)$ est vraie

Cas inductif: $\forall n \in \mathbb{N}$ on doit démontrer que $P(n) \Rightarrow P(n+1)$ est vraie

On suppose que $P(n)$ est vraie et démontrons que $P(n+1)$ est vraie.

On a un ensemble de $n+1$ chats:

$$\{c_1, c_2, c_3, \ldots, c_{n-1}, c_n\}$$

Comme on a supposé que $P(n)$ était vraie

$\{c_1,$ … ont la même couleur

Et

$\{c_1,$ … eur

CQFD : Ils on

$$P(1) \nRightarrow P(2)$$

Tours d'Hanoï

On avait trouvé le système de récurrence suivant :

$$T_0 = 0$$
$$T_n = 2 * T_{n-1} + 1 \; pour \; n > 0$$

On avait trouvé $T_n = 2^n - 1 \; pour \; 0 < n \leq 6$

Prouvons que c'est vrai pour $n \; \geq \; 0$

Tours d'Hanoï

Cas de base: $(n = 0)$

$$T_0 = 2^0 - 1 = 1 - 1 = 0$$

Cas inductif:

$$T_{n+1} = 2 * T_n + 1$$

$$T_{n+1} = 2 * (2^n - 1) + 1$$

$$T_{n+1} = 2^{n+1} - 2 + 1$$

$$T_{n+1} = 2^{n+1} - 1$$

Preuve par l'absurde

Pour démontrer qu'une proposition p est vraie, on va supposer qu'elle est fausse (et donc que $\neg p$ est vraie) et, à partir de cette hypothèse, arriver à une assertion fausse.

$$Si\ (\neg p \Rightarrow F)\ est\ vraie\ alors\ p\ est\ vraie$$

Si on sait que $\vee$ = ou et $\wedge$ = et

Forme de P	**Forme de** $\neg P$
$\forall x : p(x)$	$\exists x : \neg p(x)$
$\exists x : p(x)$	$\forall x : \neg p(x)$
$p \wedge q$	$\neg p \vee \neg q$
$p \vee q$	$\neg p \wedge \neg q$
$p \Rightarrow q$	$p \wedge \neg q$

Exemple

Théorème : $\sqrt{2}$ est irrationnel

Démo : Par l'absurde, on va supposer que $\sqrt{2}$ est rationnel.

Ça veut dire que

$$\exists a \in \mathbb{Z} \; et \; \exists b \in \mathbb{N}_0 \; tel \; que$$

$$\sqrt{2} = \frac{a}{b} \; avec \; a \; et \; b \; premiers \; entre \; eux$$

$$\sqrt{2} = \frac{a}{b} \Rightarrow \mathrm{b}\sqrt{2} = a \Rightarrow b^2 * 2 = a^2 \Rightarrow a^2 \; est \; pair$$

Exemple

Si a^2 est pair, ça veut dire qu'il existe p tel que $a = 2p$

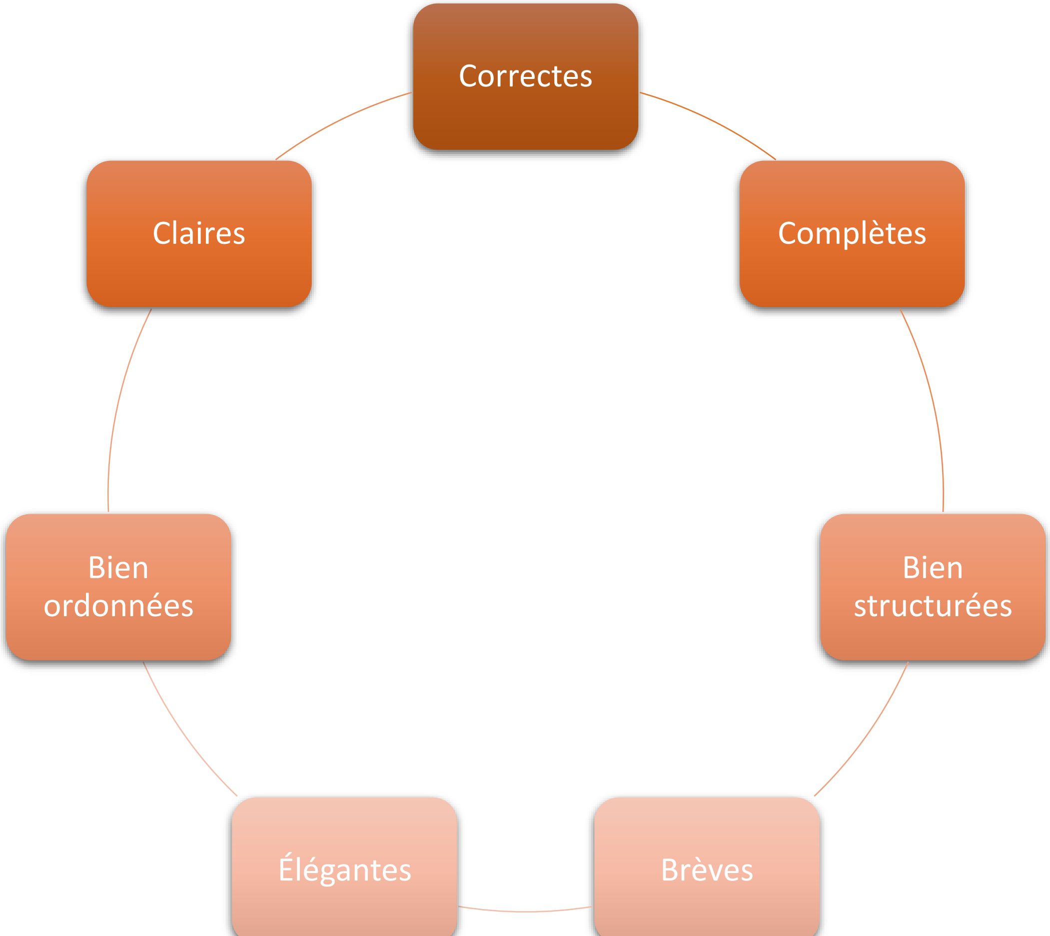
$$2b^2 = (2p)^2$$
$$2b^2 = 4p^2$$
$$b^2 = 2p^2$$

b^2 est donc pair lui aussi. On pourrait donc simplifier $\frac{a}{b}$ par 2…

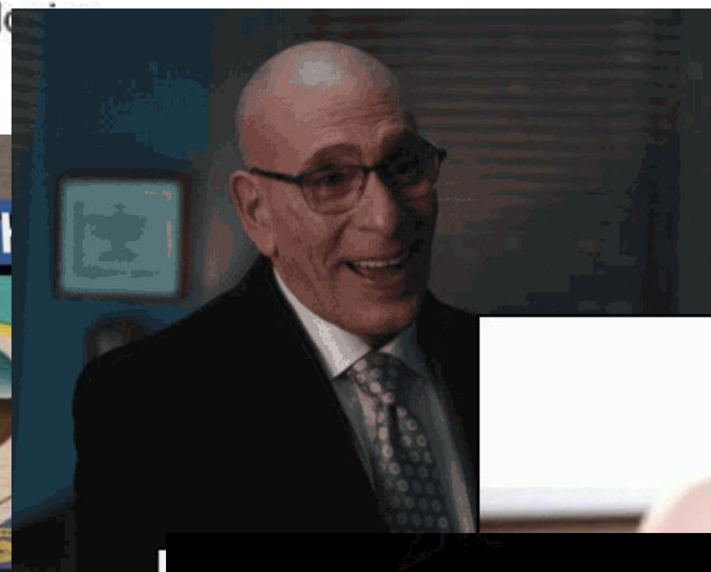
On avait dit qu'ils étaient premiers entre eux !!

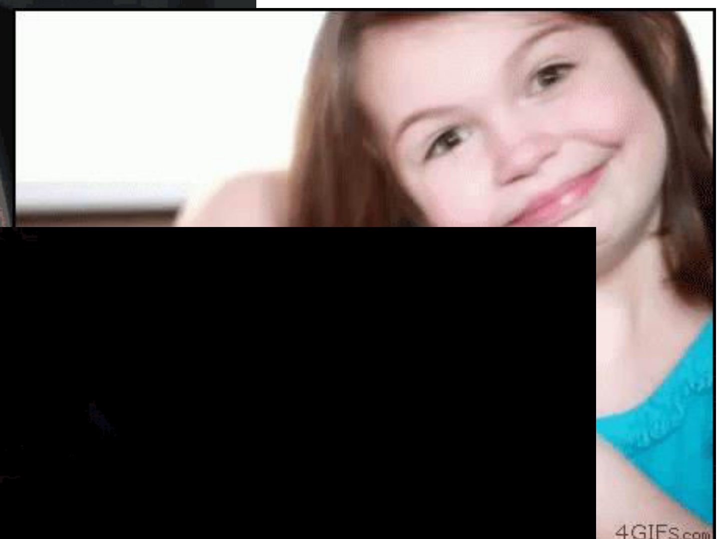
Les preuves

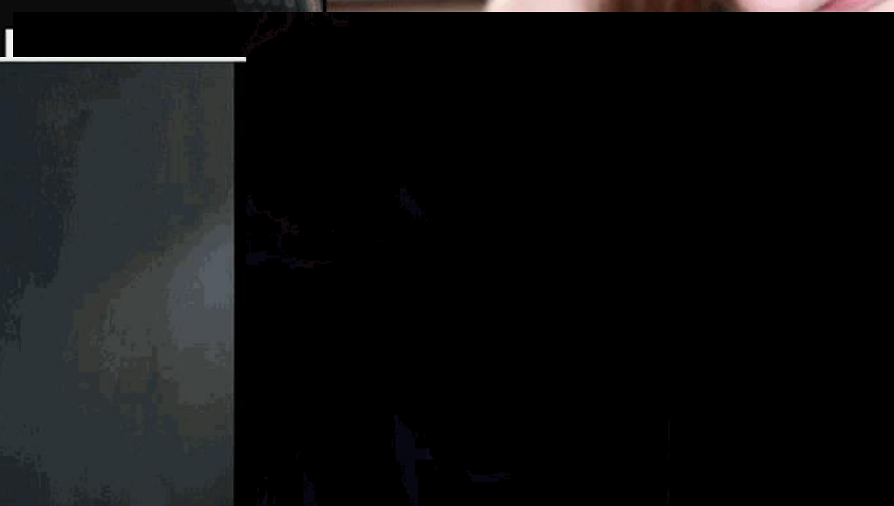
« Good proofs are like good code »



« Mauvaises preuves »







Exercices - Induction

❶ $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

❷ $\sum_{i=1}^{n} a^i = \frac{a(a^n - 1)}{a-1} \quad \forall 0 < a \neq 1$

❸ $\sum_{i=1}^{n} i^2 = \frac{n \times (n+1) \times (2n+1)}{6}$

Exercices - Absurde

- Démontrer que si vous rangez $(n+1)$ paires de chaussettes dans n tiroirs distincts, alors il y a au moins un tiroir contenant au moins 2 paires de chaussettes.

- Soit $a1, a2, \dots, a9$ des entiers naturels tels que $a1 + \cdots + a9 = 90$.Démontrer que parmi $a1, \dots, an$, il y a toujours 3 éléments dont la somme est supérieure ou égale à 30.

Prochaine séance

Séance du 26/02 est une séance d'exercices !

Sujets :

- Calculs de complexité
- Preuves par l'absurde
- Preuves par induction

Mathématiques appliquées à l'informatique

- BA2 Informatique
  Johan Depréter – johan.depreter@heh.be

Structures discrètes I :

- Ensembles, relations, logique de base

- Les ensembles

Définition

Un ensemble est une collection non ordonnée d'éléments distincts. L'ordre dans lequel les éléments sont listés n'a pas d'importance, et les doublons ne sont pas autorisés.

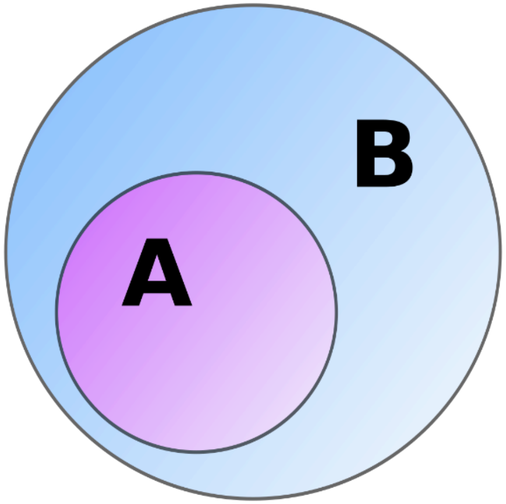
On utilise souvent la notation en extension (ex: A = {1, 2, 3}), en compréhension (ex: {x | x est un nombre pair}), ainsi que des symboles spéciaux comme $\mathbb{N}$ (entiers naturels), $\mathbb{Z}$ (entiers relatifs), $\mathbb{R}$ (réels), etc.

Ex: Soit A l'ensemble des nombres premiers inférieurs à 10: A = {2, 3, 5, 7}

Inclusion

On a un ensemble A et un ensemble B

$$A \subseteq B$$

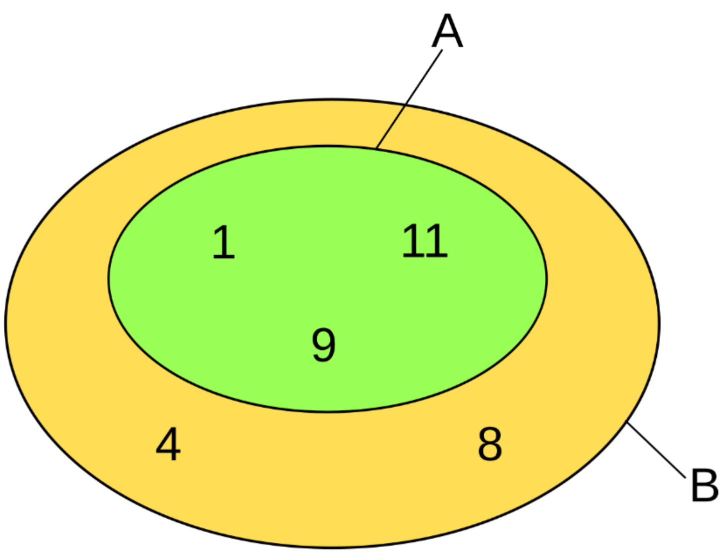


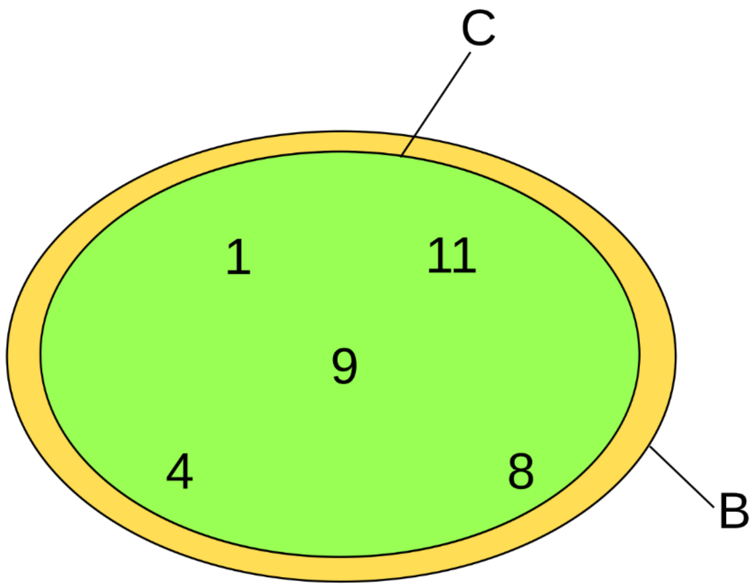
Exemple : $A = \{1,2\},\ B = \{1,2,3\} \Rightarrow A \subseteq B$

A = B ⇔ A ⊆ B ∧ B ⊆ A

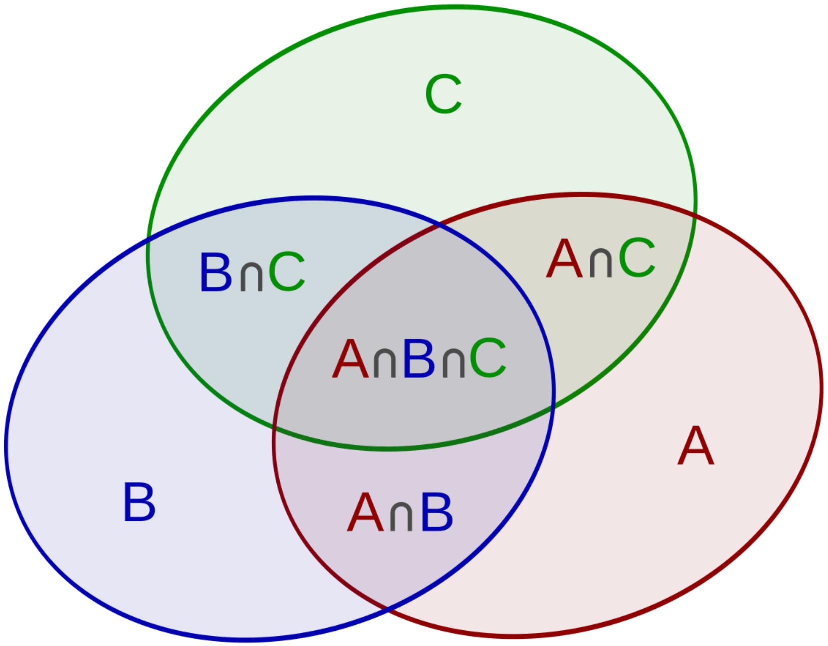
Inclusion

$\subset \neq \subseteq$





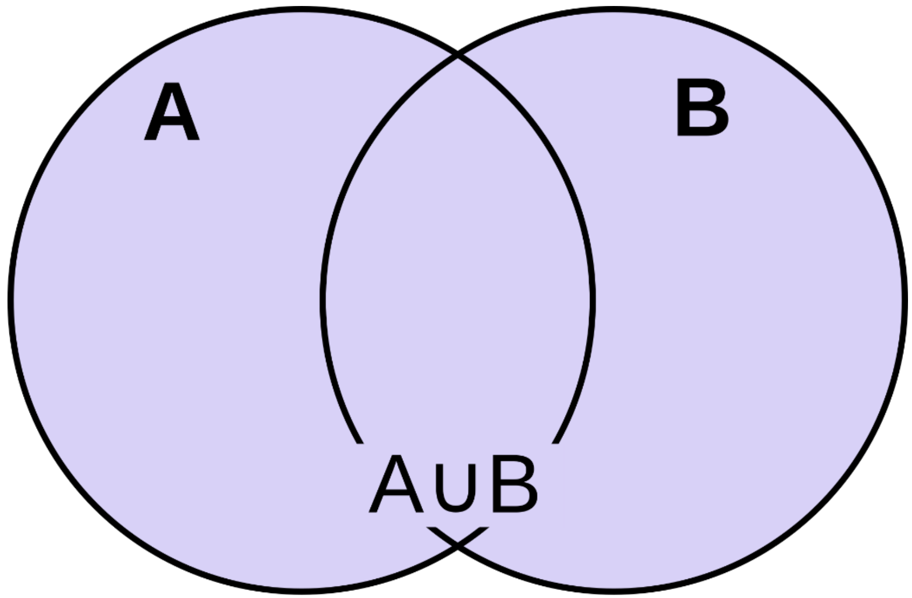
Intersection



$$A \cap B$$

$$Ex\colon \{1,2\} \cap \{2,3\} = \{2\}$$

Union



$$A \cup B$$

$$Ex: \{1,2\} \cup \{2,3\} = \{1,2,3\}$$

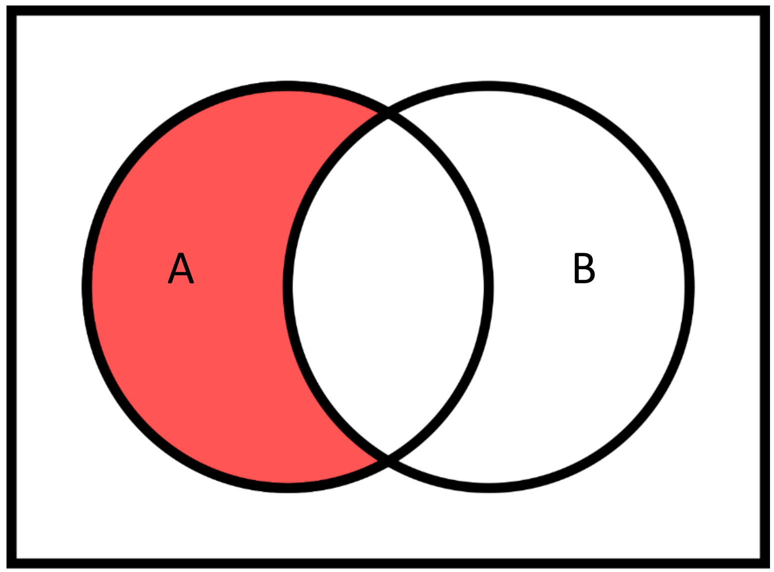
Exemple Python

```python
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.union(y)
a = x.intersection(y)

print(z)
print(a)
```

```
{'google', 'apple', 'microsoft', 'cherry', 'banana'}
{'apple'}
```

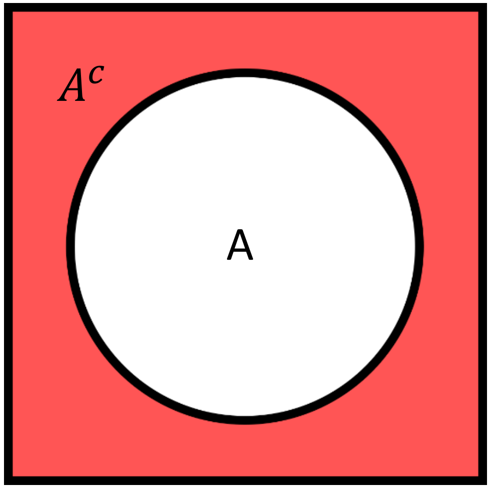
La différence

$$A \backslash B = \{x \in A | x \notin B\}$$
$$Ex\colon \{1,2,3\} \backslash \{2,3\} = \{1\}$$



Le complémenaire

$$A^c \, ou \, A' = U \setminus A \; (dans\ un\ univers\ U)$$
$$Si\ U = \{1,2,3,4\}, A = \{1,2\}, A^c = \{3,4\}$$



Produit cartésien

$$A \times B = \{(a,b) \mid a \in A, b \in B\}$$
$$\{1,2\} \times \{a,b\} = \{(1,a),(1,b),(2,a),(2,b)\}$$

Non-commutatif et non associatif

$$A \times B \neq B \times A$$
$$(A \times B) \times C \neq A \times (B \times C)$$

Ensembles finis et infinis

$$|A|$$

La cardinalité est le nombre d'éléments d'un ensemble fini

$$|\{a, b, c\}| = 3$$

$$\infty$$

Dénombrable ou indénombrable

$$\mathbb{N} \; et \; \mathbb{Z} \; sont \; dénombrables$$

$$\mathbb{R} \; est \; indénombrable$$

Récap’

Opérations clés :

$$\cup\ Union,$$
$$\cap\ Intersection,$$
$$\setminus\ Différence,$$
$$\times\ Produit\ cartésien$$

Importants pour les bases de données

Inclusion, cardinalité et complémentarité

Relations fortement basées sur le produit cartésien

- Les relations binaires

Notions de base

Une relation binaire R d'un ensemble A et d'un ensemble B est un sous-ensemble tel que :

$$R \subseteq A \times B$$

Exemple:

$$A = \{1, 2, 3\}\ et\ B = \{a, b\}$$
$$\{(0, a), (0, b), (1, a), (2, b)\}\ est\ une\ relation\ de\ A\ vers\ B$$

On peut faire la même chose avec A et A

$$A = \{1, 2, 3, 4\}\ si\ R = \{(a, b) | b\ divisible\ par\ a\}$$
$$R = \{(1,1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 4), (3, 3), (4, 4)\}$$

Exemples concrets

ID_FORMATION	NOM_FORMATION
1	SQL
2	Java
3	PHP

Table FORMATIONS

ID_FORMATEUR	NOM_FORMATEUR
101	M. Dupont
102	Mme. Lemoine
103	M. Durand

Table FORMATEURS

SELECT * FROM FORMATIONS, FORMATEURS;

ID_FORMATION	NOM_FORMATION	ID_FORMATEUR	NOM_FORMATEUR
1	SQL	101	M. Dupont
1	SQL	102	Mme. Lemoine
1	SQL	103	M. Durand
2	Java	101	M. Dupont
2	Java	102	Mme. Lemoine
2	Java	103	M. Durand
3	PHP	101	M. Dupont
3	PHP	102	Mme. Lemoine
3	PHP	103	M. Durand

Propriétés de base

Réflexive

$$\forall x, (x, x) \in \mathbb{R}$$

Chaque élément est en relation avec lui-même.

Exemple: "être égal à" - chaque nombre est égal à lui-même.

Symétrique

$$(x, y) \in \mathbb{R} \Longrightarrow (y, x) \in \mathbb{R}$$

Si x est lié à y, alors y est aussi lié à x.

Exemple: "être ami avec" - si A est ami de B, B est ami de A.

Propriétés de base

Antisymétrique

$$(x,y) \in \mathbb{R} \wedge (y,x) \in \mathbb{R} \Longrightarrow x = y$$

Si x est lié à y et inversement, alors x et y sont identiques.

Exemple: "≤" - si a≤b et b≤a, alors nécessairement a=b.

Transitive

$$(x,y) \in \mathbb{R} \wedge (y,z) \in \mathbb{R} \Longrightarrow (x,z) \in \mathbb{R}$$

Si x est lié à y et y à z, alors x est lié à z.

Exemple: "plus grand que" - si a>b et b>c, alors a>c.

Equivalence

Une relation est dite **équivalente** si elle est :

- réflexive
- symétrique
- transitive

Exemples :

- l'égalité dans $\mathbb{N}$, dans $\mathbb{Z}$
- avoir le même âge

Relation d’ordre

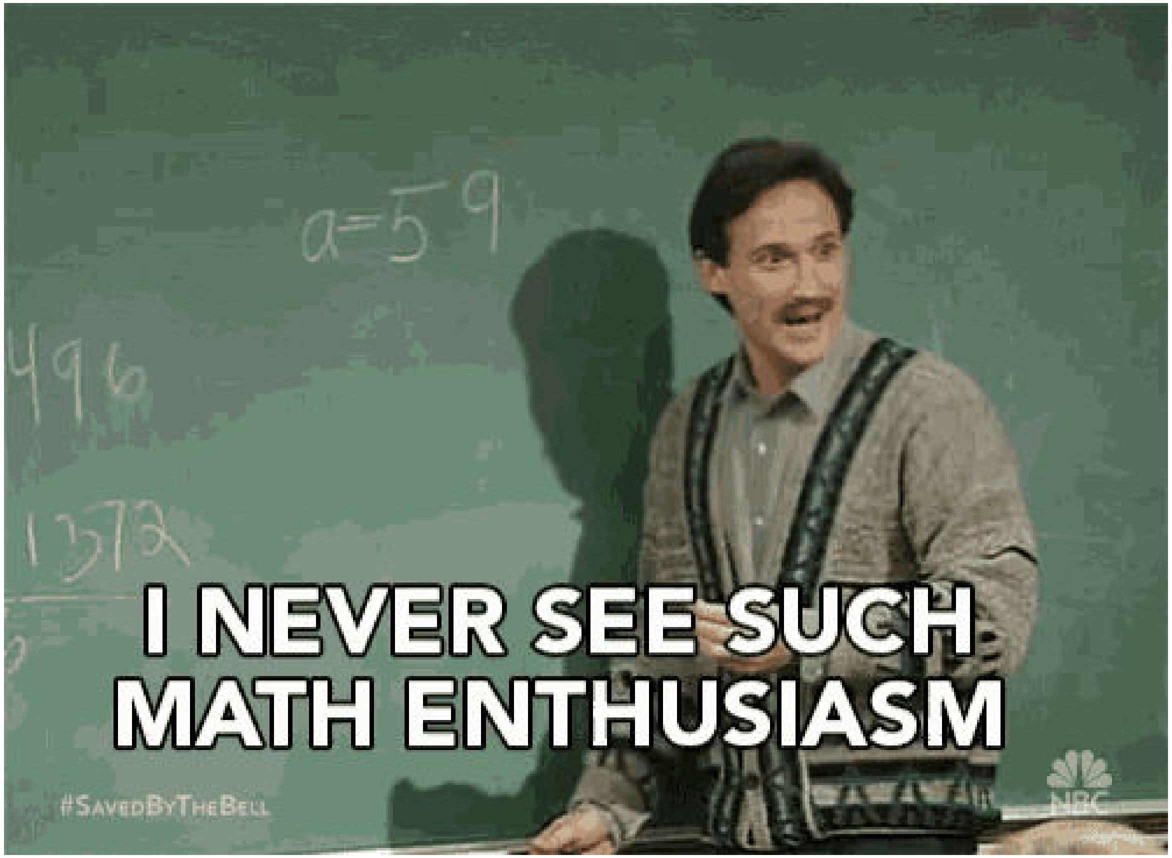
Une relation est dite **d’ordre partiel** si elle est :

- réflexive
- antisymétrique
- transitive

Exemples :

- ≤ sur les entiers
- l’inclusion dans un ensemble

Exercices


a=59
496
1372
I NEVER SEE SUCH
MATH ENTHUSIASM
#SavedByTheBell
NBC

Exercice 1

On a un univers $U = \{x \in \mathbb{Z} \mid -3 \leq x \leq 6\}$

Donc $U = \{-3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6\}$

On a 3 sous ensembles :

1. $A = \{x \in U \mid x \geq 0\ et\ x\ est\ pair\}$
2. $B = \{x \in U \mid |x| \leq 2\}$
3. $C = \{x \in U \mid x^2 \leq 9\}$

Exercice 1

1. Lister explicitement A, B, C.

2. Calculer:

$$A \cup B, A \cap B, (A \cup B) \cap C, C \setminus B.$$

3. Vérifier les relations d’inclusion :

$B \subseteq C$?

$A \subseteq C$?

4. Discuter : $(B \cup C) = C$?

Exercice 2

On considère l'alphabet binaire $\Sigma = \{0,1\}$.

On définit :

$$A = \{w \in \Sigma \ \mid |w| \leq 2\}.$$
$$B = \{w \in \Sigma \ \mid w \ commence \ par \ 0\}$$
$$C = \{w \in \Sigma \ \mid w \ se \ termine \ par \ 1\}$$

Exercice 2

1. Lister (explicitement) tous les mots de A. Puis déterminer lesquels appartiennent aussi à B, et lesquels à C.

2. Calculer $A \cap B$ et $A \cap C$.

3. Décrire $B \cup C$: que signifie “commencer par 0 ou finir par 1” ?

4. Qu’est-ce que $B \cap C$? Donner un exemple de mot qui y appartient et un exemple qui n’y appartient pas.

Exercice 3

Sur l'ensemble $X = \{1,2,3,4\}$, on définit la relation $\sim$ par :

$$a \sim b \Leftrightarrow (a\ mod\ 2) = (b\ mod\ 2).$$

1. Que signifie concrètement $(a\ mod\ 2) = (b\ mod\ 2)$.
2. Vérifier si $\sim$ est réflexive, symétrique, transitive.
3. Conclure si $\sim$ définit une équivalence, un ordre partiel, ou aucun des deux.

Exercice 4

Soit $X = \{1,2,3,4,6\}$. On définit la relation R par :

$$aRb \Leftrightarrow a \; divise \; b.$$

1. Écrire les couples (a, b) qui appartiennent à R.
2. Vérifier la réflexivité, antisymétrie, transitivité, symétrie.
3. Conclusion : Est-ce une relation d'équivalence ou un ordre partiel ? Justifier.


REALLY LONG
TIME AGO

Les graphes

Graphe orienté est représenté par $(V, E), avec\ E \subseteq V \times V$

$$arc(x, y)\ dans\ le\ graphe\ E\ \leftrightarrow couple(x, y)\ dans\ la\ relation\ R$$

On peut vérifier les propriétés des relations dans les graphes

Exemples pratiques :

- Réseau social
- Accès à l'information

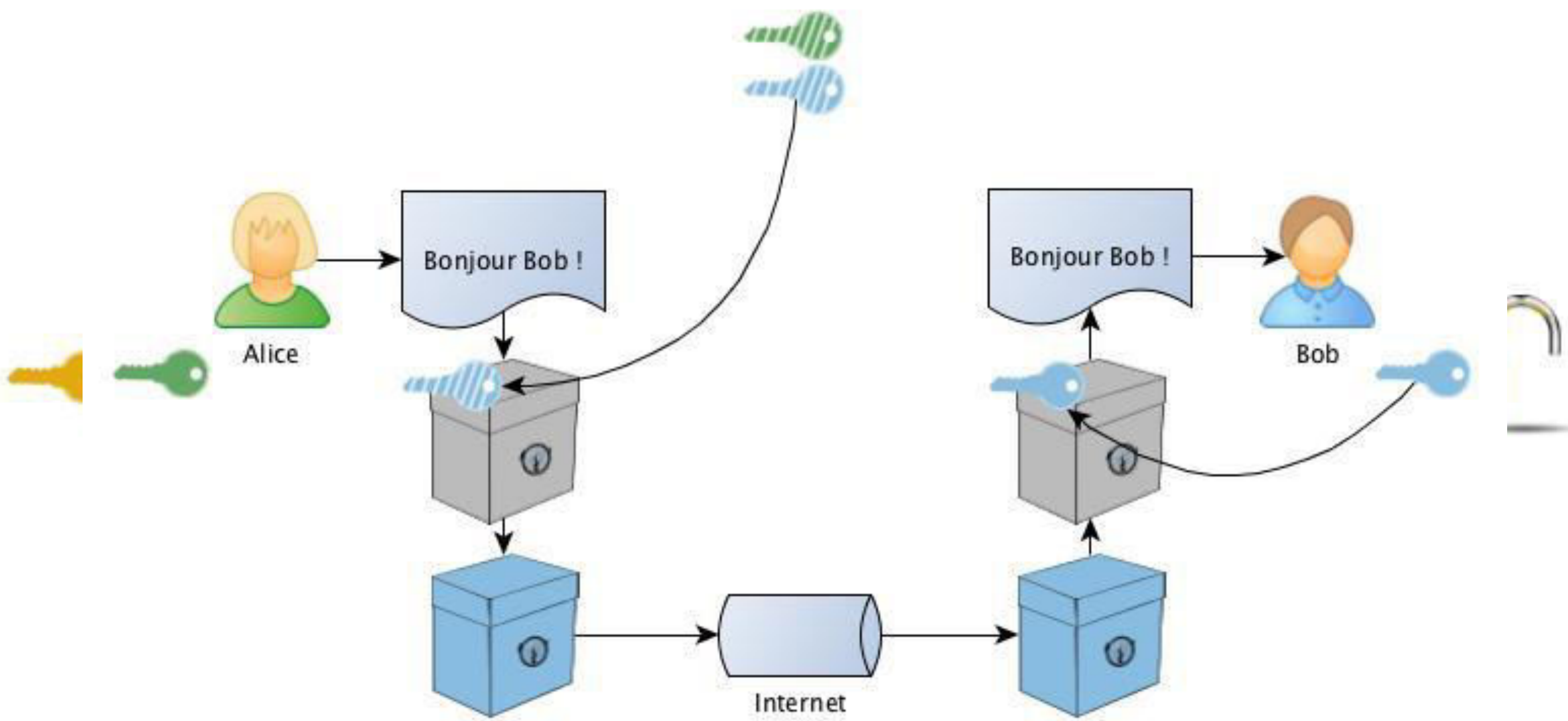
Mathématiques appliquées à l'informatique

- BA2 Informatique
Johan Depréter – johan.depreter@heh.be

- Théorie des nombres - RSA

Rappel

Chiffrement symétrique – Chiffrement asymétrique



Rappel

Symétrique	Asymétrique
1 seule clé	2 clés (publique et privée)
Rapide	Plus lent
Problème de partage de clé	Résout le problème de partage de clés

Chiffrement asymétrique

Un chiffrement asymétrique va utiliser un couple de clés :

- Une clé publique *p*
- Une clé privée *s*

La clé publique va permettre de chiffrer le message alors que la clé privée permettra de le déchiffrer

Il y a donc une fonction de chiffrement f et une fonction de déchiffrement g :

$$g(f(message, p), s) = message$$

RSA

Rivest Sham ifficulté de
factoriser de premier.

Pour génére

- On choisi
- On calcul
- On calcul $q - 1)$
- On choisi avec $\varphi(n)$
- On calcul

Factorisation en nombres premiers

Théorème : Tout entier positif $n > 1$ s'écrit comme produit de nombres premiers et ce produit est unique.

Case de base : $n = 2$

$$2 = 2 \text{ (qui est un nombre premier)}$$

Hypothèse de récurrence :

On a un entier $k \geq 2$

On suppose que pour tout entier m tel que $2 \leq m < k$, on peut écrire m comme un produit de nombres premiers

Factorisation en nombres premiers

Cas 1 : k est premier

$$k = k\ (produit\ d'un\ nombre\ premier)$$

Cas 2 : k est composite

k n'est pas premier donc :

$$\exists d\ avec\ 1 < d < k\ tel\ que\ d|k$$

$$k = d * \frac{k}{d}\ \left(avec\ 1 < \frac{k}{d} < k\right)$$

Par l'hypothèse de récurrence, on va pouvoir dire :

$$d = p_1 * p_2 * \cdots * p_r\ (p_i\ étant\ premiers)$$

$$\frac{k}{d} = q_1 * q_2 * \cdots * q_r\ (q_i\ étant\ premiers)$$

Factorisation en nombres premiers

Donc :

$$k = d * \frac{k}{d} = \quad \ldots \quad q_2 * \cdots * q_r)$$

Factorisation en nombres premiers

Reste à prouver l'unicité à l'ordre près des facteurs

$$Si\ n = p_1 * p_2 * \cdots * p_k\ et\ si\ n = q_1 * q_2 * \cdots * q_l$$

Alors il faut montrer que $\{p_1, p_2, \dots, p_k\}$ et $\{q_1, q_2, \dots, q_l\}$ sont les mêmes ensembles de nombres premiers à l'ordre près.

On peut aussi écrire :

$$n = p_1 * p_2 * \cdots * p_k = q_1 * q_2 * \cdots * q_l$$

Factorisation en nombres premiers

$$p_2 * \cdots * p_k = \frac{q_1 * q_2 * \cdots * q_l}{p_1}$$

Ce qui veut dire que : $p_1 | \ (q_1 * q_2 * \cdots * q_l)$

Utilisation du lemme d'Euclide : si un nombre premier p divise un produit ab, alors il divise a ou il divise b.

Ici, on peut donc en déduire que p_1 divise un q_j

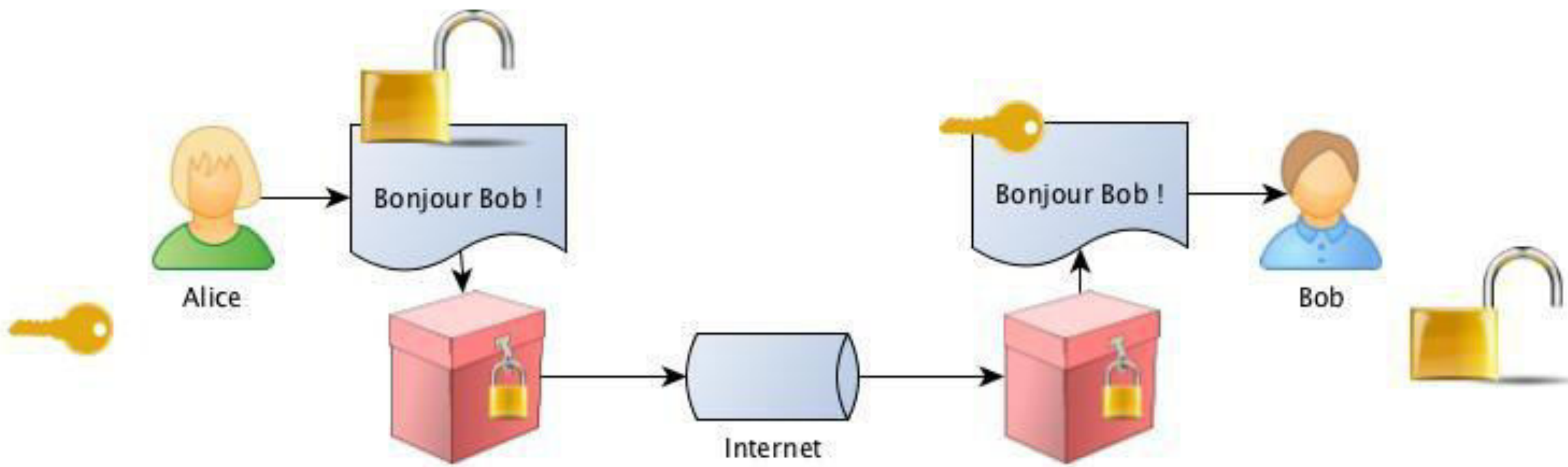
Or, on sait que q_j est premier donc $p_1 = q_j$

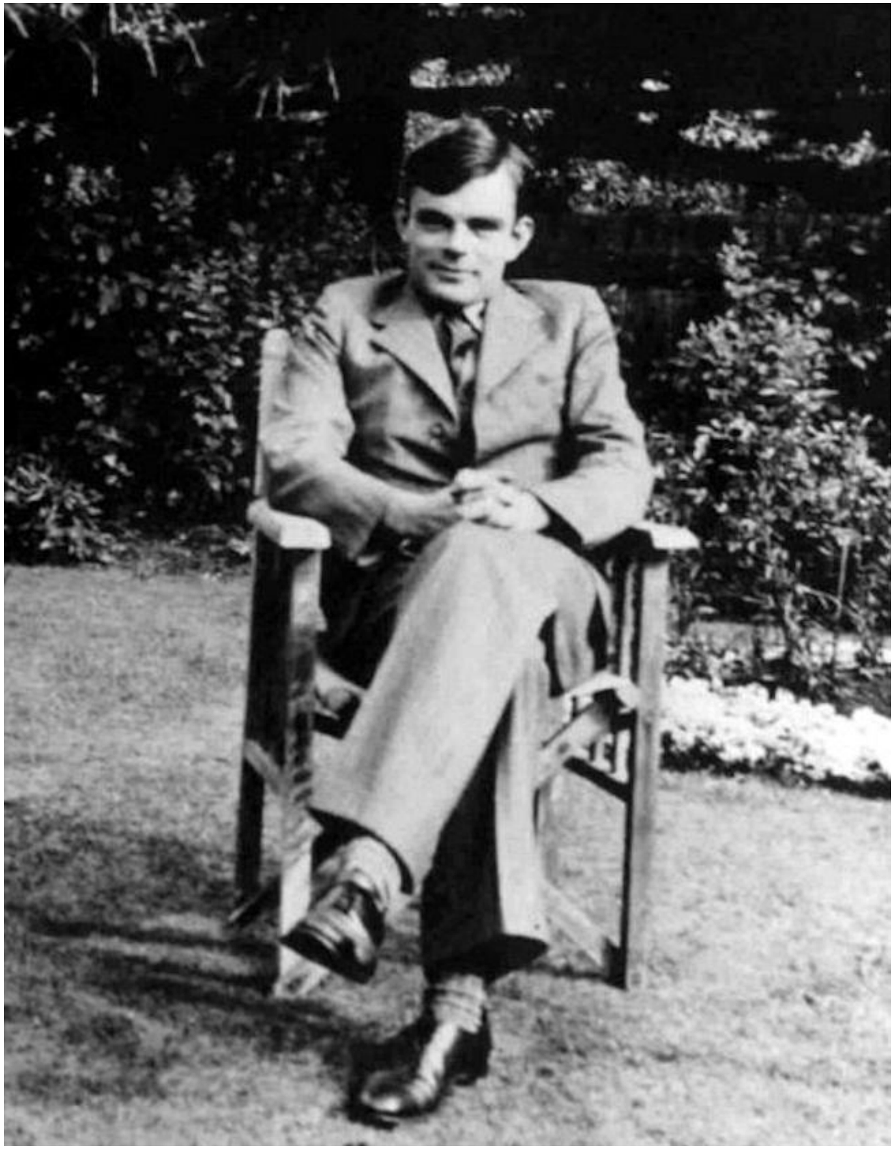
Factorisation en nombres premiers

On peut alors simplifier l'égalité en supprimant le facteur commun.

On peut reproduire l'opération avec $p_2, p_{3,\ldots}p_k$ et donc on montre que tous les p_i sont égaux à des q_j

$$CQFD$$


Alice
Bonjour Bob !
Internet
Bonjour Bob !
Bob



Code de Turing v1.0

Message :

v	i	c	t	o	r	y

↓

22	09	03	20	15	18	25	13

$$m = 2209032015182513 \text{ (premier)}$$

Clé : k un nombre premier

Message chiffré :

$$m' = m * k$$

Code de Turing v1.0

Si on choisis $k = 22801763489$ comme clé avec le message $m = 220903201518251\,3$

$$
\begin{aligned}
m' &= m * k \\
&= 2209032015182513 * 22801763489 \\
&= 50369825549820718594667857
\end{aligned}
$$

Si on connaît la clé, alors c'est simple :

$$\frac{m'}{k} = \frac{m * k}{k} = m$$

Code de Turing v1.0

Par contre, si c

Vraiment ?

Supposons qu s :

Alors :

Code de Turing v2.0

Clé secrète : k (nombre premier)

Clé publique : p (nombre premier)

Message : $m \in \{0,1,\dots,p-1\}$

Message chiffré : $m' = reste(mk,p)$

Autrement dit : $m' \equiv mk \pmod p$

Et pour déchiffrer ?

Arithmétique modulaire

Définition :

$$Soient\ x, y \in \mathbb{Z}\ et\ n \in \mathbb{N}_0$$
$$x \equiv y\ \ (mod\ n)\ si\ n|(x-y), càd\ si\ x-y \in n\mathbb{Z}$$

On va dire que x est congru à y modulo n

Exemple :

$$31 \equiv 16\ (mod\ 5)\ car\ 31-16 = 15 \in 5\mathbb{Z}$$

Inverse multiplicatif

Définition :

Si $x * y \equiv 1 \pmod{n}$, on appelle y l'inverse multiplicatif de x modulo n, et on note $y \equiv x^{-1} \pmod{n}$

Rappel : $\forall a, b \in \mathbb{N}_0$

$$pgcd(a,b) = \min\{x \in \mathbb{N}_0 | \exists s,t \in \mathbb{Z} : x = s * a + t * b\}$$

Si a et b sont premiers entre eux, $pgcd(a,b) = 1$

Inverse multiplicatif

Donc :

$$1 = s * a + t * b \Rightarrow s * a = 1 - t * b$$

$$\Rightarrow s * a \equiv 1 \ (mod\ b)$$

$$\Rightarrow s \equiv a^{-1} \ (mod\ b)$$

Inverse multiplicatif

Théorème : $x \in \mathbb{Z}$ possède un inverse multiplicatif modulo $n \in \mathbb{N}_0$ ssi $pgcd(x,n) = 1$

Démo : On vient de vérifier que un des sens, vérifions l'autre.

$$\exists y \in \mathbb{Z} : x * y \equiv 1\ (mod\ n)$$

$$\Leftrightarrow$$

$$\exists y \in \mathbb{Z}, \exists k \in \mathbb{Z} : x * y = 1 + k * n$$

$$\Leftrightarrow$$

$$\exists y \in \mathbb{Z}, \exists k \in \mathbb{Z} : x * y - k * n = 1$$

$$\Leftrightarrow$$

$$pgcd(x,n) = 1$$

Code de Turing v2.0

Clé secrète : k

Clé publique : p

Message : $m \in \{0,1,\ldots,p-1\}$

Message chiffré : $m' = reste(mk,p)$

Déchiffrement : $pgcd(k,p) = 1$ donc $\exists k^{-1}\ (mod\ p)$

$$m' \equiv m * k\ (mod\ p)$$

$$\Rightarrow m' * k^{-1} \equiv m\ (mod\ p)$$

$$\Rightarrow m \equiv m' * k^{-1} (mod\ p)$$

$$\Rightarrow m = reste(m'k^{-1}, p)$$

Fonction φ d'Euler

Définition : $Pour\ n \in \mathbb{N}_0,$

$$\varphi(n) \coloneqq \#\ entiers\ dans\ \{1, 2, 3, \dots, n-1\}$$
$$relativement\ premier\ avec\ n$$

Exemples :

$$\varphi(12) = 4 \quad \{1,2,3,4,5,6,7,8,9,10,11\}$$
$$\varphi(15) = 8 \quad \{1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$$
$$\varphi(p) = p - 1 \quad si\ p\ est\ premier$$

Théorème d'Euler

Soient n un entier positif et a un entier tel que $pgcd(a, n) = 1$.
Alors $a^{\varphi(n)} \equiv 1\ (mod\ n)$

Démonstration :

$$S = \{x \in \{1,2, \dots, n-1\} \mid pgcd(x, n) = 1\}.$$

La cardinalité de S vaut par définition $\varphi(n)$

On va considérer la fonction suivante :

$$f : S \to S, f(x) = (a * x)\ mod\ n$$

On prouve assez facilement que c'est une injection et une bijection

Théorème d'Euler

Injection :

Si $f(x1) = f(x2)$, cela signifie $(a \cdot x1) mod n \; = \; (a \cdot x2) mod n$.

Autrement dit, $ax1 \equiv ax2 (mod n)$

Or $pgcd(a, n) = 1$ implique que l'on peut "diviser" par a modulo n.
Donc $x1 \equiv x2 (mod n)$

Comme x1et x2 sont dans l'intervalle [1,n−1], on conclut x1=x2

Donc f est **injective**.

Théorème d'Euler

Il s'agit donc d'une permutation

Soit $S = \{x_1, x_2, x_3, \dots, x_{\varphi(n)}\}$, on considère le produit de tous les éléments :

$$P = x_1 * x_2 * x_3 * \cdots * x_{\varphi(n)}$$

Comme il s'agit d'une permutation, $f(x_i) = (a * x_i)\ mod\ n$ est le même ensemble

$$P' = (a * x_1) * (a * x_2) * (a * x_3) * \cdots * (a * x_{\varphi(n)})$$

$$P' \equiv \left(a^{\varphi(n)}\right) * \left(x_1 * x_2 * x_3 * \cdots * x_{\varphi(n)}\right) (mod\ \ n)$$

$$\Leftrightarrow$$

$$P' \equiv a^{\varphi(n)} * P\ \ (mod\ n)$$

Théorème d'Euler

Comme c'es … e :

Si P est prem … ns $mod\ n$

On peut aus…

Petit théorème de Fermat

Cas pa

Si n = p

RSA - Génération

On choisit 2 nombres premiers distincts p et q

On calcule le module de chiffrement $n = p * q$

On calcule la fonction d'Euler : $\varphi(n) = (p-1)(q-1)$

On choisit un nombre premier e premier avec $\varphi(n)$ donc $pgcd\big(e, \varphi(n)\big) = 1$

On détermine d tel que :

$$e * d \equiv 1 \ (mod\ \varphi(n))$$

Autrement dit :

$$d \equiv e^{-1} \ (mod\ \varphi(n))$$

On le calcule via Euclide étendu

RSA - Suite

On obtient les deux clés :

Clé publique : (n, e)

Clé privée : (n, d)

Pour chiffrer un message m (un entier $< n$) :

$$c = m^e \ (mod\ n)$$

Pour déchiffrer, en connaissant d :

$$m = c^d \ (mod\ n)$$

Pourquoi ça marche ?

On va montrer que $(m^e)^d \equiv m \ (mod\ n)$

On sait que :

$$e * d \equiv 1 \left(mod\ \varphi(n)\right)$$
$$e * d = 1 + k * \varphi(n)$$

On a alors :

$$(m^e)^d = m^{e*d} = m^{1+k*\varphi(n)} = m^1 * m^{k*\varphi(n)}$$

Pourquoi ça marche ?

On uti

Si pgc

Donc

$$CQFD$$

Sciences et technologies

Mathématiques appliquées à l'informatique

- BA2 Informatique
  Johan Depréter – johan.depreter@heh.be

- Hachage et implications

Définition d’une fonction de hachage

C’est une application :

$$h : D \rightarrow H$$

Où D est une donnée quelconque et H un espace fini.

Utilisations :

- Vérification d’intégrité
- Mots de passe
- Signatures numériques
- Tables de hachage

Sciences et technologies

Principe du Dirichlet

Exemples de Hash

MD5 – 128 bits (obsolète)

SHA-1 – 160 bits (cassé)

SHA-256 – 256 bits

SHA-3, BLAKE3 – 256 bits

Rappel des propriétés

Déterministe

« à une même entrée, la fonction produit toujours la même sortie. »

Résistance à la pré-image

« il est **incomputable (ou extrêmement difficile)** de retrouver le message d'origine à partir d'un hash. »

Résistance à la seconde pré-image

« étant donné un message x, il est difficile de trouver un autre message $x' \neq x$ tel que $h(x) = h(x')$. »

Rappel des propriétés

Résistance aux collisions

« c'est difficile de trouver **2 messages distincts** $x \neq x'$ tels que $h(x) = h(x')$ »

Diffusion

« Une **petite modification** de l'entrée provoque une **modification radicale** du hash. »

Uniformité

« La sortie du hash doit être **uniformément répartie** dans l'espace de sortie H »

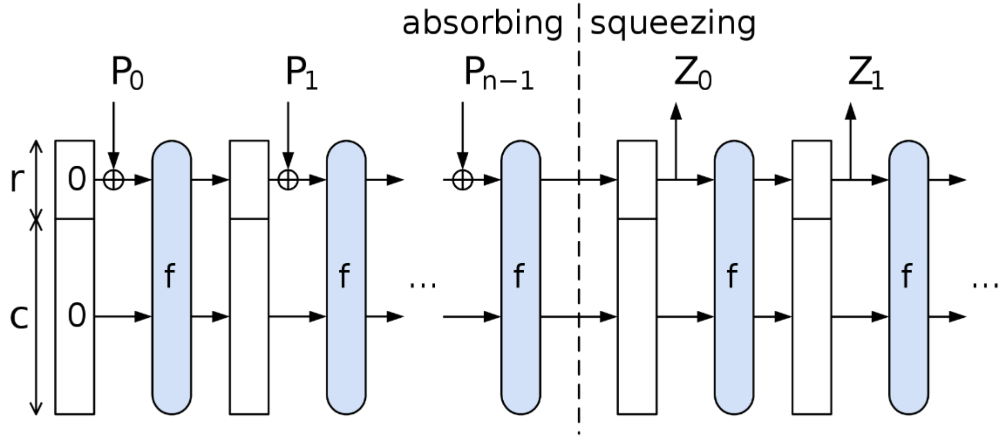
HEH.be Sciences et technologies

Récapitulatif

Propriété	But principal
Déterminisme	Fiabilité du hachage
Résistance à la préimage	Mot de passe
Résistance à la deuxième préimage	Protection contre faux
Résistance aux collisions	Intégrité, crypto
Diffusion	Obfuscation, sécurité
Uniformité	Moins de collisions
Efficacité	Applicabilité

Résistance à la préimage

Utilisation de « Construction en éponge »



Nombre importants de permutation qui empêche d'inverse directement la fonction

Résistance à la seconde préimage

Encodage de la longueur du message (Merkle-Damgård)

- On va ajouter la longueur du message à la fin du message

- Un peu comme une signature et une date à une lettre identique

Diffusion

Utilisations de XOR, de changement de bits et de rotation de bits

Exemples :

$MD5(« bonjour ») = f02368945726d5fc2a14eb576f7276c0$

$MD5(« Bonjour ») = ebc58ab2cb4848d04ec23d83f7ddf985$

Modification de l’état interne entre chaque itération

Uniformité

Tests statistiques Diehard et les STS du NIST

Batterie de tests qui permettent de mesurer la qualité de l'aléatoire

Se compose, entre autre, de :

- Tests de distances minimales,
- Tests de compression,
- Tests de flux binaire,
- …

HEH.be Sciences et technologies

Paradoxe des anniversaires

Dans un groupe de n personnes, quelle est la probabilité que deux d’entre elle partagent le même anniversaire ?

Hypothèses :
- 365 jours
- Distribution uniforme

Paradoxe des anniversaires

Etape 1 – Événement complémentaire

« La probabilité que toutes les personnes aient des anniversaires différents »

$$P(Collision) = 1 - P(Pas\ de\ collision)$$

Paradoxe des anniversaires

Etape 2 – Calcul

$$P(n) = \frac{365}{365} * \frac{364}{365} * \frac{363}{365} * \cdots * \frac{365 - n + 1}{365}$$

$$P(n) = \prod_{i=0}^{n-1} (1 - \frac{i}{365})$$

Donc, la probabilité qu'il y ait au moins une collision :

$$P_{Collision}(n) = 1 - \prod_{i=0}^{n-1} (1 - \frac{i}{365})$$

Paradoxe des anniversaires

Etape 3 – Application numérique

$$P(2) = 1 - \frac{365 * 364}{365^2} \approx 0.0027$$

$$P(23) \approx 0.5073$$

À 23 personnes, il y a +/- 50% de chances que deux d'entre elles aient le même anniversaire.

Paradoxe des anniversaire

Intuition :

«On a 365 jours, donc tant qu’on n’a pas 365 personnes, il y a peu de chances de collision.»

Ce serait vrai si on cherchait une date particulière.

Sauf qu’on veut juste savoir si deux personnes quelconques ont le même jour, peu importe lequel.

HEH.be Sciences et technologies

Paradoxe des anniversaires

Ce qu'on fait réellement :

On compare chaque paire possible de personne dans un groupe

$$Nombre\ de\ comparaison = \binom{n}{2} = \frac{n * (n+1)}{2}$$

Par exemple, si n = 23, on fait 253 comparaisons !

Approximation pour des grandes valeurs

$$P(n) \approx e^{-n*(n-1)/2*k}$$

Et donc :

$$P_{Collision}(n) \approx 1 - e^{-n*(n-1)/2*k}$$

On peut faire une estimation rapide du seuil 50% :

$$e^{-n^2/2*k} \approx 0.5 \Longrightarrow \frac{n^2}{2*k} \approx \ln(2) \Longrightarrow n \approx \sqrt{2k*\ln(2)} \approx 1.774\sqrt{k}$$

$$Si\ k = 365 \Longrightarrow n \approx 23$$
$$Si\ k = 2^{32} \Longrightarrow n \approx 77163$$
$$Si\ k = 2^{128} \Longrightarrow n \approx 2^{64} \approx 1.8*10^{19}$$

Attaque par collision

Trouver deux messages différents $x \neq x'$ tels que :

$$h(x) = h(x')$$

Grâce au paradoxe des anniversaires, on sait que la complexité d'à peu près $\sqrt{k}$ si le hash à k valeurs possibles.

Utilisations :

- Falsifier des documents
- Casser des signatures
- Tromper les systèmes de vérification d'intégrité

Collisions réelles

MD5 (2004)

On est capable de générer des collisions en quelques secondes (FastColl, etc)

Il est encore utilisé dans certaines vérification d’intégrité de fichier pdf, etc

En 2008, création de faux certificats SSL

SHAttered

SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

CWI

Marc Stevens
Pierre Karpman

Google

Elie Bursztein
Ange Albertini
Yarik Markov

SHAttered

$$SHA1 = 38762cf7f55934b34d179ae6a4c80cadccbb7f0a$$

Ressources utilisées :

- 9 223 372 036 854 775 808 SHA-1 calculés (~2^{63})
- Environ 110 années CPU cumulées (GPU + CPU)
- Environ 110 000 $ de coût estimé

SHAttered

Conséquences :

Navigateur Chrome (Google) a désactivé SHA-1 pour les certificats HTTPS en 2017

Microsoft, Mozilla, Apple ont suivi peu après

SHA-1 est déconseillé pour toute nouvelle application

SHA-256, SHA-3 et BLAKE3 sont les alternatives recommandées

Conclusions

Les collisions arrivent vite

Probabilités liées à n^2/k

Toujours garder en mémoire les attaques par collision

Fonction de hachage avec un k assez grand