Chapitre :
La complexité

Outils mathématiques : analyse élémentaire

- $\lfloor x \rfloor$ partie entière inférieure (ou plancher) du réel x : le plus grand entier ≤ x
  - En Python fonction `floor()`
- $\lceil x \rceil$ partie entière supérieure (ou plafond) du réel x : le plus petit entier ≥ x
  - En Python fonction `ceil()`

Outils mathématiques : analyse élémentaire

- $\lfloor x \rfloor = n \quad \Leftrightarrow n \leq x < n + 1$
- $\lceil x \rceil = n \quad \Leftrightarrow n - 1 < x \leq n$
- $\lfloor x + n \rfloor = \lfloor x \rfloor + n$
- $\lceil x + n \rceil = \lceil x \rceil + n$
- Pour tout entier n :
  - $n = \lfloor n/2 \rfloor + \lceil n/2 \rceil$

Outils mathématiques : analyse élémentaire

Pour tout réel x, pour tout entier n :

- $\lfloor x \rfloor < n \iff x < n$;
- $\lceil x \rceil \leq n \iff x \leq n$;
- $n < \lceil x \rceil \iff n < x$;
- $n \leq \lfloor x \rfloor \iff n \leq x$.

Pour tous réels x et y :

- $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1$;
- $\lceil x \rceil + \lceil y \rceil - 1 \leq \lceil x + y \rceil \leq \lceil x \rceil + \lceil y \rceil$.

Outils mathématiques : analyse élémentaire

$$\exp(a)\exp(b) = \exp(a+b) \qquad \exp(-a) = \frac{1}{\exp(a)}$$

$$\exp(x) = y \quad \Longleftrightarrow \quad x = \ln(y) \ (\text{pour } y > 0)$$

$$\exp(\ln(y)) = y \qquad \ln(\exp(x)) = x$$

$$\ln(uv) = \ln(u) + ln(v) \qquad \ln\left(\frac{1}{u}\right) = -\ln(u)$$

On en déduit (au moins pour n entier) :

$$a^n = \exp(\ln(a))^n = \exp(n\ln(a))$$

On défini donc, pour $x > 0$ et a quelconque

$$x^a := \exp(x\ln(a))\,.$$

27

Outils mathématiques : analyse élémentaire

ln logarithme népérien (ou naturel), de base e

$\log_a$ logarithme de base a : $\log_a(x) = \dfrac{\ln x}{\ln a}$

log fonction logarithme sans base précise, *à une constante multiplicative près*

$\log_2$ logarithme binaire, de base 2 : $\log_2(x) = \dfrac{\ln x}{\ln 2}$

$$a^x = y \quad \Longleftrightarrow \quad x = \log_a(y)$$

$$2^x = y \quad \Longleftrightarrow \quad x = \log_2(y)$$

28

Temps de calcul

- Sur les machines actuelles, difficile de prévoir le temps car dépend:
  - traduction (interprétation, compilation) code de haut niveau vers code de bas niveau, microcode.
  - l'environement (mémoire, système d'exploitation, multi-threading, hyper-threading...) ;
  - nombreuses optimisations qui dépendent de l'historique (caches, prédictions de branches...).
  - → utilisation d'un modèle de machine simplifiée

29

Complexités

- Définitions (complexités temporelle et spatiale)
  - complexité **temporelle** (ou en temps) : temps de calcul ;
  - complexité **spatiale** (ou en espace) : l'espace mémoire requis par le calcul.

30

Complexités

- Définitions (complexités pratique et théorique)
  - La complexité pratique est une mesure précise des complexités temporelles et spatiales pour un modèle de machine donné.
  - La complexité (théorique) est un ordre de grandeur de ces couts, exprimé de manière la plus indépendante possible des conditions pratiques d'exécution.

Discussion sur un exemple

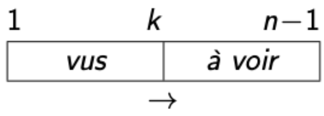
- Problème (plus grand diviseur)
  - Décrire une méthode de calcul du plus grand diviseur autre que lui-même d'un entier $n \geq 2$.

- Notons *pgd(n)* le plus grand diviseur :
  - $1 \leq pgd(n) \leq n - 1$;
  - $pgd(n) = 1 \Leftrightarrow$ n est premier.

Discussion sur un exemple

- Algorithme 1 :

On parcours les nombres de 2 à $n-1$ et l'on note le dernier diviseur que l'on a trouvé :



33

Discussion sur un exemple

- Algorithme 1 :

```
Entrée : un entier n
Sortie : pgd(n)
res ← 1
Pour k de 2 à n - 1 faire
     si k divise n alors
          res ← k
retourner res
```


34

Discussion sur un exemple

- Algorithme 2 :

Puisqu'il s'agit de trouver le plus grand diviseur, on peut procéder en décroissant sur les diviseurs possibles :

1	k	$n-1$
à voir		*vus*
	$\leftarrow$	

35

Discussion sur un exemple

- Algorithme 2 :

```
Entrée : un entier n
Sortie : pgd(n)
k ← n - 1
tant que k>0 et n mod k ≠ 0 faire
    k ← k - 1
retourner k
```


36

Discussion sur un exemple

- Algorithme 3 :

Remarque : le résultat cherché est $n \div p$, où p est le *plus petit* diviseur supérieur ou égal à 2 de n.

Notons $ppd(n)$ le plus petit diviseur en question.

2	k	n
vus	*à voir*	

$\rightarrow$

Si *ppd(n)* est trouvé alors on peut en déduire le *pgd(n)* par la formule :

pgd(n) = n / ppd(n)

Discussion sur un exemple

- Algorithme 3 :

```
Entrée : un entier n
Sortie : pgd(n)
k ← 2
tant que k<n et n mod k ≠ 0 faire
     k ← k + 1
retourner n/k
```

Discussion sur un exemple

- Algorithme 4 :
  - On peut maintenant tenir compte de ce que :
    - n non premier $\Longrightarrow 2 \leq ppd(n) \leq pgd(n) \leq n-1.$
  - D'où il vient que :
    - n non premier $(ppd(n))^2 \leq n$
  - Proposition
    - Si n ne possède pas de diviseur compris entre 2 et $\lfloor\sqrt{n}\rfloor$ c'est qu'il est premier ;
  - Permet d'améliorer le temps de calcul pour les nombres premiers : inutile de chercher en croissant entre $\lfloor\sqrt{n}\rfloor$ + 1 et n.

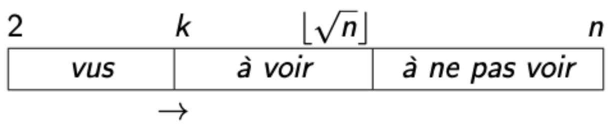
Licence Informatique 2ème année - Ph. Hunel 2024-2025

39

Discussion sur un exemple

- Algorithme 4 :

En procédant en croissant sur les diviseurs possibles :



Licence Informatique 2ème année - Ph. Hunel 2024-2025

40

Discussion sur un exemple

- Algorithme 4 :

```
Entrée : un entier n
Sortie : pgd(n)
k ← 2
tant que k ≤ ⌊√n⌋ et n mod k ≠0 faire
      k ← k + 1
si k > ⌊√n⌋
      retourner 1
sinon
      retourner n/k
```