

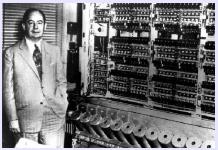

Information, Calcul et Communication

Compléments de cours

J.-C. Chappelier

Leçon I.1b (algorithmes, complexité) – Points clés

2 grandes parties :

① comment écrire un algorithme

② complexité d'un algorithme

Leçon I.1b (algorithmes, complexité) – Points clés ①

① **comment écrire un algorithme :**

- « méthodologie » :
  1. **comprendre** le problème : question ? entrée(s) ? sortie(s) ?
     (écriture de l'algorithme : ne pas oublier de bien spécifier l'/les entrée(s) et la/les sortie(s))
  2. trouver *un* algorithme correct (vérifier qu'il est bien correct ; penser à vérifier les cas limites)
  3. trouver un algorithme *efficace*
     ☞ savoir calculer la complexité d'un algorithme
- « boite à outils » :
  - affectation ($x \longleftarrow 18$) et autres opérations usuelles
  - 3 structures de contrôle : branchements, boucles, itérations
  - **Sortir :**
  - *appel* d'autres (sous-)algorithmes
- trois problèmes « clés » : recherche, tri, plus court chemin

Leçon I.1b (algorithmes, complexité) – Points clés ②

② **complexité d'un algorithme :**

- est une fonction de la taille de l'entrée
- compte le nombre d'opérations élémentaires nécessaires
- dans le pire des cas
- on s'intéresse à son comportement à l'infini (c.-à.d. quand la taille de l'entrée augmente) :
  on utilise donc une notation d'ensemble : $\Theta(f(n))$:
  ensemble des fonctions qui « croissent à l'infini comme $f(n)$ »
- meilleure complexité des algorithmes résolvant les trois problèmes « clés » :
  - recherche : liste ordonnée : $\Theta(\log n)$; liste quelconque : $\Theta(n)$
  - tri : $\Theta(n \log n)$
  - plus court chemin : ? (semaine prochaine)

Leçon I.1b (algorithmes, complexité) – Question(s)

QUESTIONS ?

Rappel semaine passée :

1. Quelle est la meilleure complexité ?
   A. $\Theta(\log(n))$ B. $\Theta(n)$ C. $\Theta(n \cdot \log(n))$ D. $\Theta(n^2)$

La *meilleure* complexité est la plus petite :
on préfère un algorithme qui prend *moins* de temps

Trouver tous les éléments maximaux dans une liste

Eléments maximaux

entrée : *L une liste (non vide ? ?) de nombres*
sortie : *(liste des positions des) éléments maximaux de la liste*

$n \longleftarrow$ ***taille***(L)
$L \longleftarrow$ ***tri***(L)
$L' \longleftarrow (n)$ *// liste à 1 seul élément*
Tant que $n > 1$ ***et*** $L(n-1) = L(n)$
  $n \longleftarrow n-1$
  $L' \longleftarrow L' \oplus (n)$ *// ajout de la valeur n à la liste*
Sortir : L'

Complexité ?

☞ $\Theta(n \log n)$, mais...
...il est FAUX ! !

Peut-on faire mieux ?

☞ **oui**
non seulement *juste*
mais aussi
à moindre coût

Trouver tous les éléments maximaux dans une liste

Eléments maximaux v2

entrée : *L une liste (non vide ? ?) de nombres*
sortie : *(liste des positions des) éléments maximaux de la liste*

$n \longleftarrow$ ***taille***(L)
$L' \longleftarrow ()$ *// liste vide*
$x_{\max} \longleftarrow -\infty$
Pour *i allant de* 1 *à n*
  Si $L(i) > x_{\max}$
    $x_{\max} \longleftarrow L(i)$
    $L' \longleftarrow (i)$ *// liste à 1 seul élément*
  Sinon, si $L(i) = x_{\max}$
    $L' \longleftarrow L' \oplus (i)$ *// ajout de la valeur i à la liste*
Sortir : L'

Complexité ?
☞ $\Theta(n)$

Peut-on faire mieux ?
☞ **non**

Leçon I.1b (algorithmes, complexité) – Recherche

Recherche dans une liste : qu'est-ce qui est le mieux ?

A] faire une recherche linéaire

B] commencer par trier la liste, puis faire une recherche par dichotomie

Cela dépend de ce que l'on veut :

soit K le nombre de fois que l'on fait la recherche,
et soit n taille de la liste ;

A : $K \cdot f(n)$, avec $f \in \Theta(n)$
B : $g(n) + K \cdot h(n)$, avec $g \in \Theta(n \log n)$ et $h \in \Theta(\log n)$