Algorithmes et données.

Algorithme est une procédure de calcul bien définie qui prend en entrée une valeur, ou un ensemble de valeurs, et qui donne en sortie une valeur, ou un ensemble de valeurs. Un algorithme est donc une séquence d´étapes de calcul qui transforment l'entrée en sortie.

Algorithme est un outil permettant de résoudre un **problème de calcul** bien spécifié. L'énoncé du problème spécifie, en termes généraux, la relation désirée entre l'entrée et la sortie. L'algorithme décrit une procédure de calcul spécifique permettant d'obtenir cette relation entrée/sortie.

Exemple: **problème de tri**

Entrée: suite de n nombres $<a_1, a_2, \ldots, a_n>$.
Sortie: permutation (réorganisation) $<a'_1, a'_2, \ldots, a'_n>$ de la suite donnée, de façon que $a'_1 <= a'_2 <= \ldots <= a'_n$.

Une **instance d'un problème** consiste en l'entrée (satisfaisant aux contraintes, quelles qu'elles soient, imposées dans l'énoncé du problème) requise par le calcul d'une solution au problème.

Un algorithme et dit **correct** si, pour chaque instance en entrée, il se termine en produisant la bonne sortie. L'on dit qu'un algorithme correct **résout** le problème donné. L'algorithme incorrect risque de ne pas se terminer pour certaines instances en entrée, voire de se terminer sur une réponse autre que celle désirée.

Exemples des types de problème susceptibles d'être résolus par des algorithmes.

- le tri et la recherche

- le projet du génome humain

- le problèmes de recherche reliés à l'internet

- recherche du chemin le plus court

Techniques du développement d'algorithmes.

Problèmes et les algorithmes efficaçes.

- problèmes indécidables

- problèmes à solution exponentielle

- problémes NP-complets

- algorithmes d'approximation pour les problèmes NP-complets

- algorithmes polynomiaux

- algorithmes de la compléxité O(n log n)

- algorithmes linéaires

Importance de l'algorithmique vu les technologies modernes.

Efficaçité des algorithmes.

Exemple 1: le tri

- tri par insertion avec un ordinateur rapide A: $c_1 n^2$
- tri par fusion avec un ordinateur lent B: $c_2 n \log_2 n$.

A: un milliard d'instructions par seconde
B: 10 millions d'instructions par seconde

A: programmeur excellent, en langage machine: $c_1 = 2$,
B: programmeur médiocre, langage de haut niveau avec un compilateur peu performant: $c_2 = 50$.

Pour trier un million de nombres

A:

$$\frac{2 \cdot (10^6)^2 \text{ instructions}}{10^9 \frac{\text{instructions}}{\text{seconde}}} = 2000 \text{secondes}$$

B:

$$\frac{50 \cdot (10^6) \cdot \log 10^6 \text{instructions}}{10^7 \frac{\text{instructions}}{\text{seconde}}} \approx 100 \text{secondes}$$

Pour 10 millions de nombres: le tri par fusion - 20 minutes, le tri par insertion 2.3 jours.

Exemple 2: minimisation des automates finis.

Avant: 99% du programme en temps $c_1 n \log_2 n$ + 1% du programme en temps $c_1 n^2$.
Après: 100% du programme en temps $c_2 n \log_2 n$.
$c_1 = c_2 = 10{,}000$. 1 milliard d'instructions par seconde.

Avant:

$$\frac{10000(0.99(10^6)\log 10^6 + 0.01(10^6)^2)\text{instructions}}{10^9 \frac{\text{instructions}}{\text{seconde}}} \approx 28\text{heures}$$

Après:

$$\frac{10000(10^6)\log 10^6 \text{instructions}}{10^9 \frac{\text{instructions}}{\text{seconde}}} = 200\text{secondes}$$

Données.

Attributes fondamentaux de données:

- valeur

- nom (identificateur, méthôde d'accès)

- structure

- réprésentation dans la mémoire machine (addresse, taille de bloc)

- vie
    - naissance-activité-mort
    - naissance-activité-hibernation(inactivité)-activité-. . .-mort

Exemple:

int n;
float A[100];

noms: A, n, A[2], A[n+1], A[150], 3, 2.5, "bonjour", false

Attributes fondamentaux de données versus temps de compilation/exécution de programme

Critères de classification de données

- possèdent la valeur ou non (variables définies, indéfinies)

- changent la valeur ou non (variables, constantes)

- possèdent le nom ou non (constantes nommées, littéraux)

- type de vie (globales, locales, paramètres)

- structurées ou non (scalaires)

- changent la structure ou non (statiques, dynamiques)

Structures de données

Une **structure de données** est un moyen de stocker et organiser des données pour faciliter l'accès à ces données et leur modification. Il n'y a aucune structure de données qui réponde à tous les besoins. de sorte qu'il importe de connaître les forces et limitations de plusieurs de ces structures.

Analyse des algorithmes

- temps de calcul (possible autres ressources: la mémoire, la largeur de la bande de communication, le processeur, etc.)
- machine à accès aléatoire (RAM) à processeur unique
- instructions
    - arithmétiques (addition, soustraction, multiplication, division, modulo, partie entière, partie entière supérieure)
    - transfert de données (lecture stockage, copie)
    - instructions de contrôle (branchement conditionnel et inconditionnel, appel de sous-routine, sortie de sous-routine)

- types de données entier et virgule flottante; taille du mot de données bornée (par ex. entiers de taille n sont réprésentés par $c \lg n$ bits, avec $c >= 1$)
- autres instructions
    - exponentiation
- modèle RAM et hiérarchisation de la mémoire

Taille de l'entrée

- nombre d'éléments constituant l'entrée (par ex. tri)
- nombre total de bits (par ex. multiplication)
- graphe: nombre de sommets et le nombre d'arcs

Temps d'exécution

- nombre d'opérations élémentaires ("étapes") exécutées
- normalement chaque ligne de pseudo code - temps constant d'exécution

Exemple: Analyse du tri par insertion

Trier un tableau A de n éléments.

t_j - le nombre de fois que le test de boucle **tant que**, en ligne 5, est exécuté pour cette valeur de j (quand une boucle pour ou tant que se termine normalement le test est effectué une fois de plus que le corps de boucle).

	TRI-INSERTION (A)	coût	fois
1	**pour** $j \leftarrow 2$ à *longueur*[A]	c_1	n
2	**faire** *clé* $\leftarrow A[j]$	c_2	n-1
3	{ Insère $A[j]$ dans la suite triée $A[1 \,..\, j\text{-}1]$ }	0	n-1
4	$i \leftarrow j$-1	c_4	n-1
5	**tant que** $i > 0$ et A[i] > *clé*	c_5	$\sum_{j=2}^{n} t_j$
6	**faire** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i \leftarrow i$-1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] \leftarrow$ *clé*	c_8	n-1

Le temps d'exécution de TRI-INSERTION est

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Pour le tableau déjà trié (le cas le plus favorable) la valeur de t_j est toujours égale à 1 (pour j=2,3,...,n), donc on obtient

$$T(n) = (c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1))$$
$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

Pour le tableau trié en ordre décroissant (le cas le plus défavorable) la valeur de t_j est toujours égale à j (pour j=2,3,...,n), on a

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \qquad \text{et} \qquad \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

donc on obtient

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n-1)}{2} - 1\right) + c_6\frac{n(n-1)}{2} + c_7\frac{n(n-1)}{2} + c_8(n-1)$$
$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

Le tri par insertion a dans le cas le plus défavorable le temps d'exécution $\Theta(n^2)$ ("théta de *n*-deux")

Complexité algorithmique

Notation asymptotique

Notation Θ

$$\Theta(g(n)) = \{f(n) | \text{ il existe des constantes positives } c_1, c_2 \text{ et } n_0 \text{ telles que } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ pour tout } n \ge n_0\}$$

Pour tout $n \ge n_0$ la fonction f(n) est égale à g(n) à un facteur constant près. On dit aussi que g(n) est une **borne assymptotiquement approchée** de f(n).

Chaque membre $f(n) \in \Theta(g(n))$ doit être asymptotiquement positif.

Exemple:

$$f(n) = \frac{1}{2}n^2 - 3n$$

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

En choisissant $c_1 = 1/14$, $c_2 = 1/2$ et $n_0 \ge 7$ on peut vérifier que

$$\frac{1}{2}n^2 - 3n = \Theta(n^2)$$

Notez que $n^3 \ne \Theta(n^2)$

Notation O

$$O(g(n)) = \{f(n) | \text{ il existe des constantes positives } c \text{ et } n_0 \text{ telles que } 0 \le f(n) \le cg(n) \text{ pour tout } n \ge n_0\}$$

$O(g(n))$ - "grand *O* de *g* de *n*"

$f(n) = \Theta(g(n))$ implique $f(n) = O(g(n))$

Exemples:

$n = O(n^2)$

$n^2 \ne O(n)$

La borne $O(n^2)$ concernant le cas le plus défavorable du tri par insertion s'applique à toutes les entrées possibles. On l'appelle la **complexité maximale**. En revanche, la borne concernant le temps d'exécution du tri par insertion dans le cas le plus défavorable n'implique pas que, pour chaque entrée, le tri par insertion ait un temps d'exécution borné par $\Theta(n^2)$. Par exemple, si l'entrée était déjà triée, le tri par insertion s'exécutait avec un temps en $\Theta(n)$.

Les compromis espace- temps

On est souvent obligé de choisir entre une amélioration de la complexité de mémoire et une amélioration de la complexité de temps d'exécution, l'une se faisant au détriment de l'autre.