

Programmation Orientée Objet

Cours n°9 : Algorithmes et complexité

Alain Giorgetti

giorgett@lifc.univ-fcomte.fr

<http://lib.univ-fcomte.fr/~giorgett/>

Laboratoire d'Informatique
de l'Université de Franche-Comté

Notion d'algorithme

- Algorithme : ensemble d'actions visant un objectif simple et bien défini de traitement d'information
 - ◆ correspond souvent à un verbe d'action
 - échanger, recevoir, ranger, compter, construire ...
 - ◆ agit sur des données initiales variables ("entrées")
 - ◆ produit des résultats ("sorties") ou des effets
 - ◆ pour un même besoin, plusieurs algorithmes possibles
- Programmation d'un algorithme
 - ◆ expression dans un langage de programmation
 - ◆ un seul algorithme, plusieurs programmes (variantes)
 - ◆ styles de programmation (itérative, récursive ...)

Traitement des collections

- Origine des collections
 - ◆ 1, 2, ... plusieurs
 - monôme, binôme, ... polynôme
 - point, segment, triangle, ... polygone
 - ◆ regroupe plusieurs données de même nature
- Traitements
 - ◆ systématiques, itératifs ou récursifs
 - affichage
 - saisie
 - recherche d'éléments
 - ◆ Si ordre sur les éléments : classement, tri
 - ◆ Efficaces jusqu'à quelles tailles de collections ?

Structures collectives étudiées

- Tables
 - ◆ tableaux (**array** en Pascal) améliorés
 - ◆ tri par insertion : nombreux décalages
 - ◆ si le tableau est trié, recherche efficace (dichotomie)
- Listes chaînées
 - ◆ pour des insertions et des suppressions efficaces
 - ◆ recherche dans une liste : peu efficace
 - ◆ tri d'une liste par fusion
- Arbres binaires de recherche (ABR)
 - ◆ pour une recherche efficace
 - ◆ structure maintenue triée sur ajouts et suppressions

Complexité d'un algorithme

- Pour mesurer l'efficacité d'un algorithme, pour comparer des algorithmes
 - ◆ coût en nombre d'opérations, en fonction du nombre n de données (mots mémoire ou bits) à traiter
 - ◆ algorithme polynomial : $O(n^2)$, $O(n^3)$, ... opérations
 - parcours d'une matrice carrée $n \times n$, produit de 2 matrices, ...
 - ◆ algorithme exponentiel : $O(e^n)$
 - inutilisable en pratique si $n > N$
- Coût moyen : pas facile à calculer
 - hypothèses probabilistes sur la répartition des données
- Complexité : coût maximal, pire des cas

Complexité : premiers exemples

- Recherche dans un tableau **non trié** de n éléments
 - ◆ pire des cas : élément cherché absent du tableau
 - ◆ complexité linéaire : $O(n)$ tests
- Tri naïf d'un tableau
 - ◆ algorithme : pour chaque rang i (≥ 1) dans le tableau,
 - rechercher le maximum des éléments non triés (de i à n)
 - échanger ce maximum avec le premier élément non trié (en i)
 - ◆ complexité
 - pire des cas : tableau trié à l'envers
 - pour le rang i , la recherche coûte au plus $n - i + 1$ tests
 - au total, $n + (n - 1) + (n - 2) + \dots + 1 = n(n + 1)/2$ tests
 - complexité quadratique : $O(n^2)$, car n négligeable devant n^2
 - ◆ naïf car il existe des tris plus efficaces

Tri d'un tableau par insertion

- Données à trier

$$T[1], T[2], \dots, T[n]$$

- Algorithme

Pour chaque rang i , avec $2 \leq i \leq n$, insérer $T[i]$ dans $T[1], \dots, T[i-1]$ supposés triés.

L'insertion s'effectue par décalages successifs.

- Complexité

- pire des cas : tableau trié à l'envers
- pour le rang i , l'insertion nécessite au pire $i - 1$ décalages (et une affectation)
- au total, $2 + 3 + \dots + (n - 1) + n$
- complexité quadratique : $O(n^2)$

- Il existe des tris plus efficaces

Recherche par dichotomie

- Recherche de l'indice d'un élément x dans un tableau trié par ordre croissant

$$T[1] \leq T[2] \leq \dots \leq T[n]$$

- Algorithme récursif
 - ◆ forme générale : recherche entre $T[i]$ et $T[j]$
 - arrêt si $i > j$
 - ◆ choisir un rang m entre i et j
 - plus efficace si m est le milieu de $[i, j]$
 - ◆ comparer x avec $T[m]$
 - retourner m si $x = T[m]$
 - recherche entre $T[i]$ et $T[m]$ si $x < T[m]$
 - recherche entre $T[m+1]$ et $T[j]$ si $x > T[m]$

Complexité d'une dichotomie

- Soit $C(n)$ la complexité d'une recherche par dichotomie sur n données
 - ◆ on suppose que $n = 2^k - 1$
 - ◆ pour $n = 1$, une seule comparaison donc $C(1) = 1$
 - ◆ pour $n > 1$, on choisit m égal à $(n + 1) / 2 = 2^{k-1}$
 - une comparaison entre x et $T[m]$, qui échoue (pire des cas)
 - une recherche entre $T[1]$ et $T[m - 1]$, coût $C((n-1)/2) < C(n/2)$
 - **ou** une recherche entre $T[m + 1]$ et $T[n]$, coût $< C(n/2)$
 - ainsi, $C(n) = 1 + C(n/2)$ ou $C(2^k - 1) = 1 + C(2^{k-1} - 1)$
 - la suite $u_k = C(2^k - 1)$ est arithmétique de raison 1 et $u_1 = 1$
 - ◆ On vérifie que $C(n) = k = \log_2(n + 1)$
- Complexité logarithmique : $O(\log_2 n)$

Algorithmes de tri efficaces

- Tri par insertion, tri à bulle, ... : $O(n^2)$
- Tri rapide (*quicksort*)
 - ◆ coût moyen $O(n \log n)$
 - ◆ mais complexité (pire des cas) en $O(n^2)$
- Tris de complexité $O(n \log n)$
 - ◆ tri par tas (*heapsort*)
 - tas :
 - (1) arbre binaire équilibré presque complet
 - (2) chaque nœud est supérieur ou égal à ses fils
 - tri de meilleure complexité dans le pire des cas
 - mais moins bon en moyenne que le tri rapide

Tri par arbre binaire de recherche

- Principe

- ◆ insérer toutes les données à trier dans un ABR
- ◆ parcourir l'ABR en ordre infixé

- Complexité

- ◆ insertion de n données une à une dans un ABR
 - hypothèse : l'insertion ne déséquilibre pas l'arbre
 - insertion aux feuilles de l'arbre
 - nombre de nœuds parcourus :

$$\log_2 1 + \log_2 2 + \log_2 3 + \dots + \log_2 (n-1) < n \log_2 n$$

- ◆ parcours récursif de l'ABR en $O(n)$ opérations
- ◆ total : $O(n \log_2 n)$ si on maintient l'équilibre de l'arbre
 - conduit à la notion de tas et au tri par tas (*heapsort*)