Module: Structures de données en C
Licence Fondamentale IA, S3

Pr. Abdelkaher AIT ABDELOUAHAD

Département d'Informatique
Faculté des Sciences, Université Chouaib Doukkali
El Jadida

a.abdelkaher@gmail.com

19 Septembre 2024

Pré requis

- Algorithmique 1 et 2

- Programmation en langage C 1 et 2

Objectifs du cours

- Étudier les structures de données les plus utilisées en programmation
- Être capable de choisir la structure adéquate à chaque problème
- Être familiarisé à la résolution des problèmes algorithmiques
- Pouvoir implémenter les structures de données en C

Contenu

- Généralités
- Type abstrait de données
- Liste chaînée
- Pile
- File
- Arbre
  - Arbre binaire de recherche
  - Arbre de décision
  - Arbre équilibré
- Les graphes

Partie 1 : Généralités

Algorithmes

- Un algorithme est une description complète et détaillée des actions à effectuer et de leur séquencement pour arriver à un résultat donné
  - Intérêt : séparation analyse/codage (pas de préoccupation de syntaxe)
  - Qualités : exact (fournit le résultat souhaité), efficace (temps d'exécution, mémoire occupée), clair (compréhensible), général (traite le plus grand nombre de cas possibles),...
- L'algorithmique désigne aussi la discipline qui étudie les algorithmes et leurs applications en Informatique
- Une bonne connaissance de l'algorithmique permet d'écrire des algorithmes exacts et efficaces
- Exemples :
  - Une recette de cuisine est un algorithme, l'entrée étant les ingrédients et la sortie le plat cuisiné.
  - L'algorithme d'Euclide : c'est un algorithme permettant de déterminer le plus grand commun diviseur (P.G.C.D.) de deux entiers

Représentations des algorithmes

Historiquement, deux façons pour représenter un algorithme :

- L'organigramme : représentation graphique avec des symboles (carrés, losanges, etc.)
  - offre une vue d'ensemble de l'algorithme
  - représentation quasiment abandonnée aujourd'hui
- Le pseudo-code : représentation textuelle avec une série de conventions ressemblant à un langage de programmation (sans les problèmes de syntaxe)
  - plus pratique pour écrire un algorithme
  - représentation largement utilisée

Pseudo langage

- Structures élémentaires d'un pseudo langage :
  1. Entrées/Sorties : Lire, Écrire
  2. Affectation : X $\leftarrow$Y
  3. Instructions conditionnelles :
     - **Si** condition **Alors** instructions **FinSi**
     - **Si** condition **Alors** instructions1 **Sinon** instructions2 **FinSi**
  4. Instructions répétitives (les boucles)
     - **TantQue** condition **Faire** instructions **FinTantQue**
     - **Faire** instructions **TantQue** condition
     - **Pour** i=0 à n **Faire** instructions **FinPour**

Calculer la factorielle de N

Ecrire("saisissez le nombre :")
Lire (N)
F$\leftarrow$1
Pour I=1 à N Faire F$\leftarrow F \times I$
FinPour
Ecrire(F)

Remarque

Il n'existe pas de formalisme universel pour l'écriture d'un pseudo programme.

Codage d'un algorithme

- Traduire l'algorithme dans un langage de programmation pour pouvoir l'exécuter sur un ordinateur.
- Chaque instruction élémentaire sera traduite dans le langage de programmation.
- Il faut choisir la façon de représenter les données (structures de données).
- Pour résoudre un problème : nombreux algorithmes possibles.
- Pour chaque algorithme : plusieurs choix de structures de données.

Opérations élémentaires

- étant donné un algorithme, nous appelons opérations élémentaires :

  1. un accès en mémoire pour lire ou écrire la valeur d'une variable ;
  2. une opération arithmétique entre deux variables : +,- ...etc ;
  3. une comparaison entre deux variables.

- Considérons par exemple l'instruction c $\leftarrow$ a + b. Elle fait appel à quatre opérations élémentaires :

  1. l'accès en mémoire pour lire la valeur de a,
  2. l'accès en mémoire pour lire la valeur de b,
  3. l'addition de a et b,
  4. l'accès en mémoire pour écrire la nouvelle valeur de c.

Notions de complexité

- Plusieurs algorithmes permettent de résoudre un même problème.
- Exemple : Pour trier les éléments d'un tableau il y a différents algorithmes :
  1. tri par sélection,
  2. tri par insertion,
  3. tri rapide,
  4. tri par fusion,
  5. ....
- Comment évaluer les performances d'un algorithme ?
- Sur quel critère il faut se baser pour choisir le meilleur algorithme ?

Notions de complexité

- La différence entre les algorithmes peut ne pas être visible si la taille des données est petite.
- Cette différence augmente proportionnellement à la quantité des données.
- La complexité a été développée pour mesurer le degré de difficulté d'un algorithme (le coût de l'algorithme).
- La complexité permet de comparer l'efficacité des algorithmes.

Notions de complexité

- La complexité d'un algorithme est une évaluation du coût de l'algorithme en termes de :
  1. temps d'exécution (complexité temporelle) ou
  2. d'espace mémoire (complexité spatiale, encombrement en mémoire des données)

- On va traiter dans la suite la complexité temporelle. Les mêmes notions permettent de traiter la complexité spatiale.
  $\Rightarrow$ Le temps est beaucoup plus important que l'espace.

- La complexité permet de déterminer si un algorithme A et meilleur qu'un algorithme B.

Complexité en temps

- Bonne maîtrise de la complexité se traduit par :
  1. temps de calcul des applications prévisible,
  2. mémoire occupée par l'application est contrôlée.
- Mauvaise compréhension de la complexité débouche sur :
  1. des latences importantes dans les temps de calcul,
  2. des débordements mémoire,
  3. conséquence : planter la machine.

Complexité en temps

L'évaluation exacte du temps de calcul dépend de nombreux paramètres :

1. le langage utilisé pour coder l'algorithme (compilé ou interprété).
2. le compilateur utilisé.
3. l'ordinateur sur lequel va tourner le programme (sa rapidité).
4. taille et structure de données.
5. ....

Complexité en temps

- Pour évaluer un algorithme, une relation, entre la taille des données n et le temps t nécessaire pour leur traitement, est utilisée.

- Pour une relation linéaire entre n et t tel que $t_1 = cn_1$
  $n_2 = 5n_1 \rightarrow t_2 = 5t_1$

  Multiplication des données par 5 $\Longrightarrow$ Multiplication du temps par 5.

- Pour une relation non linéaire entre n et t : $t_1 = log_2 n_1$
  $n_2 = 2n_1 \rightarrow t_2 = log_2(2n_1) = t_1 + 1$

  Multiplication des données par 2 $\Longrightarrow$ Augmentation du temps par une unité (1).

Complexité en temps

- La fonction d'efficacité f liant n et t est importante pour une taille énorme de données.
- Les termes qui influencent faiblement sur l'amplitude de f pour une taille importante des données sont à éliminer.
  - On garde uniquement une approximation de f.
  - Cette approximation est plus proche à la fonction originale lorsqu'on traite une grande quantité de données.
  - Nous sommes plus intéressés par un comportement asymptotique : que se passe-t'il quand la taille des données tend vers l'infini ?
  - Cette mesure d'efficacité est appelée : Complexité asymptotique

Complexité en temps

- Exemple illustratif : $f(n) = n^2 + 100n + log_{10}n + 1000$
  - La fonction f est composée de 4 termes : n^2, $100n$, $log_{10}n$ et 1000.
  - Analyser la contribution de chaque terme en fonction de n (taille de données)

n	f(n)	n^2		100n		$\log_{10}$n		1000	
	Value	Value	%	Value	%	Value	%	Value	%
1	1,101	1	0.1	100	9.1	0	0.0	1000	90.82
10	2,101	100	4.76	1,000	47.6	1	0.05	1000	47.62
100	21002	10,000	47.6	10,000	47.6	2	0.991	1000	4.76
1,000	1,101,003	1,000,000	90.8	100,000	9.1	3	0.0003	1000	0.09
10,000	101,001,004	100,000,000	99.0	1,000,000	0.99	4	0.0	1000	0.001
100,000	10,010,001,005	10,000,000,000	99.9	10,000,000	0.099	5	0.0	1000	0.00

  - Pour des valeurs importantes de n la fonction f dépend principalement de son premier terme n^2. Les autres termes peuvent être ignorés.

Structures de données : introduction

- Programme = structures de données + algorithmes
- Une structure de données est une représentation des relations logiques qui existent entre les éléments individuels d'une catégorie de données.
- L'objectif des structures de données est le stockage des éléments auxquels ont souhaite accéder plus tard.
- On appel les différentes utilisation possible de la structure de données des opérations (lecture, insertion, suppression,...)
- Les structures de données sont définies indépendamment du langage de programmation.
- Bien connaître ses structures de données et savoir faire un choix joue donc un rôle très important pour le programmeur.

Structures de données : introduction

- Les opérations et le coût (la complexité) différent d'une structure à une autre
- Le choix d'une telle structure doit tenir compte de :
  - La place mémoire consommée par la structure.
  - La facilité qu'elle offre pour accéder à une certaine donnée.
- Attribution mémoire
  - Par le compilateur et ne peut pas être modifiée au cours du programme (variables statiques).
  - Effectuée pendant le déroulement du programme et peut donc varier pendant celui-ci (variables dynamiques).
- Organisation des données en mémoire
  - Contiguë : les éléments se trouvent en des adresses consécutives, l'un après l'autre.
  - Non contiguë : les éléments se trouvent en des adresses dispersées.

Objectifs

- Les structures de données peuvent être
  - Primitives (individuels) : entiers, réels, caractères, ...
  - Non primitives (composés) : tableaux, listes, fichiers,....
- Les structures de données peuvent être
  - Linéaires : une structure linéaire est un arrangement linéaire d'éléments liés par la relation successeur (exemples : Tableaux, Listes chaînées, Piles, Files).
  - Non linéaires
    - Arborescentes : Arbres, arbres binaires, ...
    - Relationnelles : Graphes
- Pour chaque structure, on présente
  - une définition abstraite ;
  - les différentes représentations en mémoire ;
  - une implémentation en langage C ;
  - quelques applications.

Types de données

- Avoir choisi les bons types de données permet d'avoir un programme :
  - plus lisible car auto documenté
  - plus facile à maintenir
  - souvent plus rapide, en tout cas plus facile à optimiser

" I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. " — Linus Torvalds

Bibliographie

- L'essentiel des structures de données en C. Ellis Horowitz, Sartaj Sahni. Dunod 1993
- Algorithmes et Structures De Données Génériques : Cours Et Exercices Corrigés en Langage C. Michel Divay, Dunod 2004
- Initiation à l'Algorithmique et à la Programmation en C. Rémy Malgouyres, Rita Zrour, Fabien Feschet. 2ème édition 2011.
- Programmer en C. Claude DELANNOY, 5ème édition 2009, Eyrolles.