Algorithmes et Structures de Données

Fabrice Huet

Université de Nice-Sophia Antipolis

Université Nice SOPHIA ANTIPOLIS

Plan - Objectifs

- Notion d'Algorithme
- Complexité
- Algorithmes itératifs et récursifs
- Types de Données Abstraites
  - Structures linéaires
  - Structures arborescentes
  - Graphs
- Algorithmes de tri

Introduction

- Le mot « **algorithme** » vient du Mathématicien arabe du 9ème siècle Al al-Khwārizmī
- Un algorithme traite de l'information
  - L'information est codée par un nombre fini de symboles
- Un algorithme prend en entrée des données et fournit un résultat
- L'exécution de l'algorithme est automatique et consiste en une suite finie d'opérations sur des suites finies de symboles

Algorithme vs Programme

- Un programme est un algorithme
- Thèse de Turing-Church : les problèmes ayant une solution algorithmique sont ceux résolubles par une machine de Turing (théorie de la calculabilité)
- L'avènement de l'ordinateur à rendu possible l'utilisation d'algorithmes (programmes) pour résoudre des problèmes

Description informelle de 2 algorithmes

- Calcul du pgcd de deux entiers naturels (Euclide)
- Obtention des nombres premiers inférieurs à un entier N donné (crible d'Eratosthène)

PGCD de 2 entiers naturels

- Problème: étant donné deux entiers naturels n et m, calculer leur pgcd (plus grand commun diviseur)
- Euclide propose l'algorithme suivant:
  1. Calculez le reste r dans la division de m par n
  2. Si r est nul alors le pgcd est n
  3. Sinon recommencer l'étape 1 avec m = n et n = r

Exemple

- Exemple d'exécution : m = 32, n = 12
  - 32 = (2 x 12) + 8
  - 12 = (1 x 8) + 4
  - 8 = (2 x 4) + 0
- On a donc pgcd(32, 12) = 4

Crible d'Ératosthène

- Principe de l'algorithme :
  1. On barre les multiples de 2 (à partir de 4)
  2. On avance au premier nombre non barré (c'est un nombre premier)
  3. On barre ses multiples
  4. On retourne à l'étape 2 jusqu'à ce que le premier nombre non barré ait son carré supérieur à N
- Les nombres restants non barrés sont les nombres premiers inférieurs à N

Crible d'Ératosthène

- Pourquoi s'arrêter dès que le premier nombre non barré a son carré supérieur à N ?
- Si p est le premier entier après √N,
  - 2p est déjà barré (multiple de 2),
  - 3p aussi (multiple de 3), ..., (p-1)p aussi (multiple de p-1).
  - Et comme p.p > N, les multiples de p n'ont pas besoin d'être barrés et cela est encore vrai pour les multiples des nombres supérieurs à p

Exemple


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Notion de pseudo langage

- On a besoin d'un langage formel minimum pour décrire un algorithme
- Un langage de programmation (Java, C, Pascal, etc.) est trop contraignant
- Dans la littérature, les algorithmes sont décrits dans un **pseudo langage** qui ressemble à un langage de programmation (le pseudo langage utilisé dépend donc de l'auteur et peut être spécifié par celui-ci en début d'ouvrage)

Pseudo langage

- Tous les pseudo langage recouvrent les mêmes concepts de la programmation impérative :
- Variables, affectation
- Structures de contrôle : séquence, conditionnelle, itération
- Découpage de l'algorithme en fonctions, procédures
- Structures de données simples ou élaborées (tableaux, listes, dictionnaires, etc.)

Pseudo langage

```
// m et n sont supposés positif
    entier r;
    tant que (n ≠ 0) {
        r <- m mod n;
        m <- n;
        n <- r;
    }
//le pgcd est n
```

Affectation

- Les variables sont indiquées avec leur type : booléen b, entier n, réel x, caractère c, chaîne s, etc.
- Le signe de l'affectation n'est pas « = », ni « := » (comme en Pascal) mais « <- » qui illustre bien la réalité de l'affectation (« mettre dedans »)

Pseudo langage : structures de donnés

- Les tableaux sont utilisés. Si A est un tableau, A[i] est le ième élément du tableau
  - Le premier élément d'un tableau est en général à l'indice 1
- Les structures sont utilisées. Si P est une structure modélisant un point et x un champ de cette structure représentant l'abscisse du point, P.x est l'abscisse de P
- Remarque : une structure est un objet sans les méthodes

Pseudo langage : séquence d'instructions

- Les instructions simples sont séquencés par « ; »
- Les blocs d'instructions sont entourés par { … }

Pseudo langage : la conditionnelle

si (condition)
instruction1;
sinon
Instruction2;

si (condition) {
    instruction1;
    instruction2;
    ...
} **sinon** {
    instruction1;
    instruction2;
    ...
}

Pseudo langage : itérations

- Trois types de boucles :
  - tant que (condition) { ... }
  - faire { ... } tant que (condition)
  - pour (i=min jusqu'à max par pas de p) { ... }

Exemple : somme de 1 à n

```
entier somme <- 0;
pour (i=1 jusqu'à n par pas de 1) {
    somme <-somme + i;
}

entier somme <- n;
tant que (n >0) {
    n<- n-1;
    somme <- somme +n;
}
```

Pseudo langage : les fonctions

- Une fonction est un « petit » programme qui en général renvoie une valeur
- Elles permettent un découpage de l'algorithme qui rend sa compréhension et son développement plus facile
- En Java, la notion de fonction est cachée dans les méthodes appliquées aux objets

Pseudo langage : les fonctions

- Une fonction a une liste de paramètre typés et un type de retour (son prototype ou sa signature)
- Le passage de paramètres se fait par valeur
- Quand le type du paramètre est un tableau ou une structure, les éléments du tableau ou les champs des structures sont par contre modifiés par le code de la fonction

Fonction pgcd

```
entier pgcd(entier m, entier n) {
    // m et n sont supposés positif
    entier r;
    tant que (n ≠ 0) {
        r <- m mod n;
        m <- n;
        n <- r;
    }
   retourner n;
}
```

Passage par valeur vs référence

- Considérons le pseudo code suivant

  entier m <- 32;

  entier n <- 14;

  pgcd(m,n);

- Est-ce que les valeurs de *m* et *n* ont changées après l'exécution de la fonction?
  - Ça dépend de la façon dont ses valeurs ont été passées à la fonction

Passage par valeur vs référence

- Passage par valeur (ou copie)
  - Les arguments d'une fonction sont copiés
  - La fonction travaille sur des copies, les modifications sont perdues
- Passage par référence
  - Les arguments originaux sont passés (plus précisément, une référence à ces arguments est passée)
  - Les modifications sont visibles après la méthode

Passage par valeur vs référence

- Dans les algorithmes
  - Passage par valeur pour les types simples
  - Passage par référence pour les tableaux et les structures
- Dans les langages, ça dépend
  - Pascal : par valeur, sauf si mot clef VAR utilisé
  - C : par valeur, sauf si utilisation de pointeurs
  - Java : par valeur pour types primitifs, référence pour objets

Exemple : permuter deux entiers

```
permutter(entier a, entier b) {
  entier tmp <-a;
   a<-b;
   b<-tmp;
}
```

- Mais ça ne marche pas… comment faire?

Exemple : crible d'Eratosthène

- Les entiers correspondent aux indices d'un tableau de booléens crible de taille N
- i premier ⇔ crible[i] = vrai
- Au départ tous les nombres sont considérés comme premier (sauf 1)

Exemple : crible d'Eratosthène

```
booléen crible[1..N]; // le crible
entier i,k;
// on s'arrête a la racine de N
   réel limite <- √N;
// initialisation de crible
pour (i de 2 à N)
     crible[i] <- vrai;
// 1 n'est pas premier
crible[1] <- faux;
```

Exemple : crible d'Eratosthène

```
I <- 2;
tant que (i < limite) {
    si (crible[i]) { // i est un nombre premier
    k <- 2*i; // on barre les multiples de i jusqu'a N
    tant que (k <= N) {
        crible[k] <- faux;
         k <- k + i;
    }
   }
// on avance dans le tableau
I <- i + 1;
}
```

Université Nice SOPHIA ANTIPOLIS

Algorithmes récursif

- Une fonction qui fait appel à elle-même dans son corps est dites « récursive »
- De nombreux problèmes ont des solutions récursives « naturelles »
- Les langages de programmation moderne permettent l'utilisation de la recursivité

Exemple

- Calcul de la factorielle d'un entier : 0! = 1 et pour n≥1 n! = n.(n-1)!
- Calcul des nombres de Fibonacci : F0 = 0, F1 = 1 et pour n ≥2 Fn = Fn-1 + Fn-2
- Calcul du coefficient binomial C(k,n) : C(0,n) = C(n,n) =1 et pour 1≤k<n, C(k,n) = C(k-1,n-1) + C (k-1,n)
- Résolution du problème des tours de Hanoi

Exemple : Factorielle

Exemple Fibonacci

```
entier Fibonacci(entier n) {
    si (n=0) retourner 0;
    si (n=1) retourner 1;
    retourner Fibonacci(n-1) + Fibonacci(n-2);
}
```

Exemple Fibonacci (itératif)

```
entier Fibonacci(entier n) {
    entier F[0..n];
    si (n=0) renvoyer 0;
    si (n=1) renvoyer 1;
    pour (i de 2 à N)
        F[i] = F[i-1] + F[i-2];
    retourner F[n];
}
```

Tours de Hanoï

```
// n nombre de disques,
// 3 piquets, A, B, C
// déplacer les disques de A vers C
hanoi(entier n, entier A, entier B, entier C) {
    si (n>0) {
        hanoi(n-1, A, C, B);
        passer(n, A, C);
        hanoi(n-1, B, A, C);
    }
}
```

Notion de structure de données

- Un algorithme travaille sur des données
- Les données peuvent être très simples
- Exemple : deux entiers pour le calcul du pgcd
- Ou demander une organisation plus complexe
- Exemple : le crible d'Ératosthène demande d'organiser une liste des entiers inférieurs à un entier donné N

Algorithmes et Données

- Une organisation particulière des données peut induire un algorithme efficace pour solutionner un problème
- Par exemple, les mots du dictionnaire sont ordonnés par ordre alphabétique. Cela induit une procédure de recherche qui permet de trouver rapidement la définition d'un mot donné
- Si les mots du dictionnaire n'étaient pas ordonnés, la recherche d'un mot serait de type séquentielle (un mot après l'autre en partant du premier) et donc impraticable !

Type de données

- Une donnée appartient à un **type**
- Un type de données est :
  - Un ensemble de valeurs
  - Un ensemble d'opérations valides pour cet ensemble de valeurs
- Exemple : les entiers signés codés sur 16 bits
  - Valeurs : $[-2^{15} ; 2^{15} -1]$
  - Opérations possibles : +, -, * …

Type de données abstrait

- Quand un type de données n'est accessible à l'utilisateur que par l'intermédiaire d'une **interface**(*) qui cache l'implantation concrète des données et propose un ensemble d'opérations utiles sur les valeurs du type, on parle de **type abstrait de données**

Exemple : les nombres complexes

- On définit le type nombre complexe
  - Valeurs : un flottant pour la partie réelle, un autre pour la partie imaginaire
- Opérations
  - Arithmétique de base : +, -, …
  - Partie réelle, partie imaginaire
  - Module, argument

Exemple : les nombres complexes

- Comment implémenter le type nombre complexe?
- On peut utiliser une structure avec deux réels (partie réelle et imaginaire)
  - En vrac ou dans un tableau
- On peut aussi stocker le module et l'argument
  - En vrac ou dans un tableau
- 4 implémentations possibles
  - Ce n'est (presque) pas important pour celui qui l'utilise

Algorithmes et Structures de Données

Listes

Généralités

- Une liste est une suite finie d'éléments d'un même ensemble
- Une des structures les plus répandues
- Beaucoup d'exemples dans la vie réelle
  - Liste de courses, feuille d'appel…
- Exemple algorithmique
  - Liste des nombres premiers dans le crible d'Eratosthène

Type abstrait Liste

- Opérations possibles
  - Connaître sa taille (nombre de ses éléments)
  - Accéder à l'élément situé au rang i
  - Insérer un élément à la suite
  - Insérer un élément au rang i
  - Supprimer un élément (s'il est dans la liste)
  - Supprimer l'élément de rang i
  - Savoir si un élément est dans la liste

Implémentation d'une liste

- 2 façons de faire
  - Utiliser un tableau
  - Utiliser un chaînage
- Chaque façon a ses avantages et ses inconvénients
  - Certaines opérations seront plus faciles à implémenter ou plus performantes
- Choisir l'implémentation en fonction des **besoins**

Liste avec tableau

- Accéder à l'élément en position *i*
  - Très facile, il suffit de regarder la case *i*
- Ajouter un élément à la fin
  - Facile, il suffit de l'ajouter dans la dernière case **libre** du tableau
  - Mais si pas de case libre? Il faut créer un nouveau tableau plus grand, recopier les éléments de l'ancien dedans et y placer le nouveau
- Ajouter/supprimer un élément en position *i*
  - Il faut décaler tout ceux après ☹

Liste avec tableau

```
structure liste {
    entier indexDernier;
    entier tableau[N];
}
entier element(entier i) {
    si (i <= indexDenier) {
        retourner tableau[i];
    } sinon retourner ERREUR;
}
```

```
entier ajouter(entier element) {
    si  estPlein(tableau) {
        entier n <- taille(tableau);
        entier tmp[2*n]
        pour i de 1 à n faire
            tmp[i]<-tableau[i];
        tableau<-tmp;
    }
        indexDernier ++;
        tableau[indexDernier]<-element;
    }
}
```

Liste avec chainage

- Dans chaque maillon de la liste on stock
  - l'élément
  - Une information pour trouver le suivant

```
structure liste {
    entier element;
    liste suivant;
}
```

- C'est une structure de données récursive

Liste avec chainage

- On utilise souvent la représentation graphique

Exemple : compter les éléments d'une liste

- 2 versions possibles
  - Récursive
  - Itérative
- Récursif
  - Si la liste est vide, c'est 0
  - Sinon, c'est 1 + le reste de la liste (i.e. la liste qui suit le maillon en cours)
- Itératif
  - On parcourt la liste et on compte les maillons

Exemple : compter les éléments d'une liste

```
entier compteElements(liste l) {
   si (l==null) {
      retourner 0;
   } sinon {
      retourner 1+compteElements(l.suivant);
   }
}
```

Exemple : compter les éléments d'une liste

```
entier compteElements2(liste l) {
    liste tmp <- l;
    entier compteur =0;
    tant que (tmp ≠ null) {
        compteur<-compteur+1;
        tmp<-tmp.suivant;
    }
}
```

Exemple : trouver le dernier élément de la liste

- Caractériser le dernier élément
  - Si la liste est vide, il n'existe pas
  - Sinon, c'est l'élément tel que le suivant n'existe pas
- Toujours deux variantes de l'algorithme

Exemple : trouver le dernier élément de la liste

```
entier dernierElement(liste l) {
  si (estVide(l)) {
     retourner -1; //ou une valeur bien choisie
  }
  liste tmp <- l ;
  tant que (l.suivant ≠ null) {
     l <- l.suivant;
  }
}
```

Algorithmes et Structures de Données

Performance des algorithmes

Performance

- Comment choisir un algorithme ou une variante ?
  - Facilité d'écriture
  - Performance
- Mais comment parler de performance ?
  - Une durée d'exécution n'a pas de sens sans langage, machine…

Performance

- Intuitivement, on peut voir des différences
- Exemple : chercher le i-ème élément d'une liste
  - Si tableau, facile c'est juste aller à la case *i*
  - Si chainage, il faut partir du premier élément, aller au suivant…
- La performance d'un algorithme est une mesure du nombre d'opérations élémentaires qu'il effectue

Performance

- Le nombre d'opérations peut dépendre des données en entrées
  - *Somme de 1 à n* fait plus d'opérations pour de grandes valeurs de *n*
- On essaie donc d'évaluer sa performance *asymptotique*
  - Pour de très grandes tailles de données en entrées