Bonnes pratiques algorithmiques

Philippe Lac
(philippe.lac@ac-clermont.fr)
Malika More
(malika.more@u-clermont1.fr)

IREM Clermont-Ferrand

Stage Algorithmique

Année 2010-2011

Contenu

Adopter des habitudes de rigueur

Lisibilité des algorithmes

- Pour se simplifier la vie quand les algorithmes deviennent longs et/ou nombreux
- Pour se faire comprendre
- Pour l'intérêt pédagogique, encore renforcé par la programmation

Formalisation des algorithmes

- Pour faciliter la conception d'algorithmes corrects
- Pour faciliter l'étude théorique des algorithmes
- (Pour l'efficacité des algorithmes)

Présentation naïve

Algorithme 1 : Mon algo

Lire le nombre a
Lire le nombre b
tant que $b \neq 0$ **faire**
    Donner à *temp* la valeur b
    Donner à b la valeur $a \mod b$
    Donner à a la valeur *temp*
fin
Afficher a

Premier principe

Critiques

- Titre non informatif
- Données entrées au clavier : intervention humaine nécessaire
- Affichage du résultat : il n'est pas réutilisable (par exemple par un autre programme)

Conseils

- Séparer les entrées/sorties du traitement, c'est-à-dire de l'algorithme proprement dit
- On obtient alors une **fonction** qui prend en entrée des **données** et renvoie un **résultat**

Fonction `Euclide(`*a,b : entiers*`)`

Entrée : Deux entiers *a* et *b*
Sortie : Le pgcd de *a* et *b*
Description : Implémentation de l'algorithme d'Euclide
début
  ...
fin

Intérêt

- Vision des algorithmes comme des briques réutilisables, composables
- Description conceptuelle, indépendante des détails techniques
- On peut envisager une étude théorique des algorithmes
- (On peut envisager des algorithmes récursifs)

Exemple

Fonction `Euclide(`*a,b : entiers*`)`

début
  Donner à x la valeur a
  Donner à y la valeur b
  tant que $y \neq 0$ **faire**
    Donner à *temp* la valeur y
    Donner à y la valeur $x \mod y$
    Donner à x la valeur *temp*
  fin
  retourner : x
fin

Remarque

- Séparation entre les données a, b et les variables $x, y, temp$
- Pour un langage de programmation, c'est un peu plus compliqué

Algorithme vs. Fonction (mathématique)

Point commun essentiel

Un objet qui prend en entrée des données et renvoie un résultat

Différences

- Effectivité des algorithmes (encore que...)
- Paramètres vs. variables (encore que...)
- Etc.

Second principe

Améliorer la lisibilité de l'algorithme

- Pour soi (l'année suivante...)
- Pour les autres : élèves (mais aussi dans la vraie vie)

Conseils

- Commentaires décrivant le déroulement de l'algorithme
- Noms de variables explicites
- (Messages à l'écran pour l'utilisateur)
- Etc.

Euclide encore

Exemple

Algorithme 4 : Euclide commenté

```
début
    Afficher "Entrez le nombre a"
    Lire le nombre a                          % a et b sont les nombres %
    Afficher "Entrez le nombre b"             % dont on calcule le pgcd %
    Lire le nombre b
    tant que r ≠ 0 faire
        Donner à a la valeur b                % le pgcd de a et de b est %
        Donner à b la valeur r        % égal au pgcd de b et du reste r %
        Donner à r la valeur a  mod b             % et on recommence %
    fin
    Afficher "Le pgcd est" b  % le dernier reste non nul est le pgcd %
fin
```

Exercice

Que fait l'Algorithme 5 ? Commenter cet algorithme.

Algorithme 5 : Inconnu

début
  Donner à *nb* la valeur 1000
  Donner à *somme* la valeur 0
  pour *i de 1 à nb* **faire**
    Donner à *tir* la valeur 1
    Donner à *d* une valeur entière aléatoire entre 1 et 6
    tant que $d \neq 6$ **faire**
      Donner à *tir* la valeur $tir + 1$
      Donner à *d* une valeur entière aléatoire entre 1 et 6
    fin
    Donner à *somme* la valeur $somme + tir$
  fin
  Afficher $somme/nb$
fin

Moralité

L'absence de commentaires peut rendre ainsi un problème, simple au départ, complexe à l'arrivée...

Question

On peut également s'interroger sur la lisibilité de certaines solutions choisies.

Exemple (fréquemment rencontré)

Algorithme 6 : Séquence

début
  % a et b désignent deux nombres %
  Donner à a la valeur $a + b$
  Donner à b la valeur $a - b$
  Donner à a la valeur $a - b$
fin

Quel intérêt à utiliser cette séquence dans un algorithme ?

Haro sur les astuces !

Moralité

Il arrive que des solutions astucieuses (ou élégantes intellectuellement parlant), compliquent la lecture de l'algorithme.

Remarque

L'écriture d'une preuve mathématique soulève les mêmes questions.

Avertissement

- Les bonnes pratiques de conception des algorithmes sont nécessairement liées aux bonnes pratiques de programmation
- Mais il y a des notions générales indépendantes du langage considéré

Des concepts spécifiques

Exemple

- On utilise des listes ou des tableaux pour manipuler des variables indicées.
- Les instructions associées permettent d'ajouter ou de supprimer des éléments, de connaitre la longueur d'une liste, etc.
- Selon les langages de programmation, la gestion des listes et des tableaux est très variable.
- Selon la méthode choisie pour résoudre un problème, le choix d'une, ou l'autre de ces structures de données, s'avère crucial.

... sur certains desquels nous reviendrons pour une étude spécifique.

Rappels

Quelques règles de bonne pratique déjà rencontrées

- Donner le type des variables
- Initialiser toutes les variables
- Ne pas modifier la valeur des paramètres
- Ne pas modifier les compteurs de boucles

Remarque

Toute règle a ses exceptions justifiées

Nouveautés

Quelques recommandations supplémentaires

- Préférer les test $\geq$ ou $\leq$ aux tests $=$ (on ne sait jamais)
- Éviter les sorties de boucles non controlées (instruction `break`)
- Déclarer toutes les variables locales en début d'algorithme (certains langages de programmation l'imposent)

Remarque

Toute règle a ses exceptions justifiées

Résultat

Fonction `Euclide(`*a,b : entiers*`)`

Entrée : Deux entiers *a* et *b*
Résultat : Le *PGCD* de *a* et *b*
Description : Algorithme d'Euclide

variables locales : x, y, *temp* : entiers

début
- Donner à x la valeur a `% On initialise` x `à` a `%`
- Donner à y la valeur b `% On initialise` y `à` b `%`
- `% Début du calcul %`
- **tant que** $y > 0$ **faire**
  - Donner à *temp* la valeur y `% On stocke` y `%`
  - Donner à y la valeur $x \mod y$ `% On remplace` y `par le reste %`
  - Donner à x la valeur *temp* `% x prend la valeur de` y `%`
- **fin**
- `% Fin du calcul : À la sortie` $y = 0$ `donc` $x = \text{pgcd}(a, b)$ `%`
- **retourner** : x

fin

Une erreur de conception

Fonction `Test de primalité(`*n : entier*`)`

début
  Donner à *b* la valeur $\lceil \sqrt{n} \rceil$
  Donner à *test* la valeur `vrai % n est potentiellement premier %`
  pour *k* **de** *2* **à** *b* **faire**
    si *k divise n* **alors**
      Donner à *test* la valeur `faux % n n'est pas premier %`
      Donner à *k* la valeur $b + 1$ `% on sort de la boucle %`
    fin
  fin
  retourner : *test*
fin

Analyse

- L'algorithme précédent ne respecte pas un principe important : *la non modification du compteur dans une boucle* `pour`.
- Par ailleurs, on peut noter que, sans l'instruction litigieuse `Donner à` k `la valeur` $b+1$, l'algorithme fonctionne encore :

Fonction `Test de primalité(`*n : entier*`)`

début
  Donner à *b* la valeur $\lceil \sqrt{n} \rceil$
  Donner à *test* la valeur `vrai % n est potentiellement premier %`
  pour *k* **de** *2* **à** *b* **faire**
    si *k divise n* **alors**
      Donner à *test* la valeur `faux % n n'est pas premier %`
    fin
  fin
  retourner : *test*
fin

- Mais ce nouvel algorithme fait des calculs inutiles
- Cela soulève une autre préocupation à avoir dans la mise en place de bonnes pratiques, à savoir l'efficacité des algorithmes
- Nous y reviendrons largement à propos de la complexité algorithmique

Le choix de la boucle

- Cette version est plus correcte d'un point de vue algorithmique, mais moins efficace.
- Il s'agit d'un problème essentiel de conception : le type de boucle choisi n'est évidemment pas le bon.

Exercice

Réécrire l'algorithme précédent, à l'aide de la structure de boucle adaptée.

Remarque

Réciproquement, lorsqu'une boucle `tant que` répond au cahier des charges d'une boucle `pour`, on lui préfèrera l'utilisation de cette dernière, dans un souci de clarté.

D'une façon générale

Dans un souci d'efficacité et de clarté

On privilégie une programmation structurée, et donc l'utilisation des outils adaptés (fonctions, structures de données . . .)

N'exagérons pas !

On est parti d'un algorithme présenté naïvement :

Algorithme 10 : Mon algo

Lire le nombre a
Lire le nombre b
tant que $b \neq 0$ **faire**
    Donner à *temp* la valeur b
    Donner à b la valeur $a \mod b$
    Donner à a la valeur *temp*
fin
Afficher a

Pour arriver à un algorithme très structuré :

N'exagérons pas !

Fonction `Euclide(`*a,b : entiers*`)`

Entrée : Deux entiers *a* et *b*
Résultat : Le *PGCD* de *a* et *b*
Description : Algorithme d'Euclide

variables locales : $x, y, temp$: entiers

début
- Donner à x la valeur a `% On initialise` x `à` a `%`
- Donner à y la valeur b `% On initialise` y `à` b `%`
- `% Début du calcul %`
- **tant que** $y > 0$ **faire**
  - Donner à *temp* la valeur y `% On stocke` y `%`
  - Donner à y la valeur $x \mod y$ `% On remplace` y `par le reste %`
  - Donner à x la valeur *temp* `% x prend la valeur de` y `%`
- **fin**
- `% Fin du calcul : À la sortie` $y = 0$ `donc` $x = \text{pgcd}(a, b)$ `%`
- **retourner** : x

fin

N'exagérons pas !

Mais on peut considérer que cette troisième version est suffisante pour un cas aussi simple et aussi bien connu :

Fonction `Euclide(`*a,b : entiers*`)`

début
  Donner à x la valeur a
  Donner à y la valeur b
  tant que $y \neq 0$ **faire**
    Donner à *temp* la valeur y
    Donner à y la valeur $x \mod y$
    Donner à x la valeur *temp*
  fin
  retourner : x
fin

FIN