collection DataPro

2ième édition

Algorithmique

Raisonner pour concevoir

Christophe HARO

Téléchargement
www.editions-eni.fr

eni éditions

Les exemples à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence ENI de l'ouvrage **DP2ALG** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Chapitre 1
Qu'est-ce que l'algorithmique ?

1. Qu'est-ce que l'algorithmique ? 11
2. Structure du livre 16
3. Public visé 19
4. Conventions adoptées 20
5. Préface à la seconde édition 21

Partie 1
Chapitre 2
Programmes directs

1. Introduction 25
2. Premiers exemples 26
3. Définition informelle d'un algorithme 30
4. Spécifications 34
5. Premiers algorithmes 51
6. Exercices résolus 62
7. Exercices 92
8. Notes bibliographiques. 94
9. Résumé 94
10. Bibliographie 95

Chapitre 3
L'alternative

1. Introduction 97
2. Définition de l'alternative 97
3. Exercices résolus 107
4. Exercices 122
5. Résumé 125

Chapitre 4
Structures élémentaires

1. Introduction 127
2. Les chaînes de caractères 128
   2.1 Les caractères 128
   2.2 Les chaînes de caractères 135
   2.3 Exercices d'application sur les chaînes de caractères 146
3. Le tableau 147
   3.1 Les tableaux simples 147
   3.2 Tableaux composés 154
   3.3 Deux fonctions utiles sur les tableaux 155
      3.3.1 La fonction appartient 155
      3.3.2 La fonction sous_tableau 156
   3.4 Exercices d'application sur les tableaux. 158
4. Définir un nouveau type de données 158
   4.1 Définir un type de données 159
   4.2 Discussion sur les invariants 162
   4.3 Exercices d'application sur les types de données 164
5. Notes bibliographiques. 173
6. Résumé 173
7. Bibliographie 173

Chapitre 5
Itération

1. Introduction 175
2. Premiers exemples de construction d'itérations 176
   2.1 La table de multiplication 176
      2.1.1 Le problème 176
      2.1.2 Construction de l'itération 179
      2.1.3 Une autre version 189
   2.2 Itérer dans un tableau 194
      2.2.1 Rang de la composante minimum d'un tableau 194
      2.2.2 Rechercher dans un tableau trié 203
      2.2.3 Recherche par dichotomie 203
      2.2.4 Extensions 215
3. Explorer un tableau 216
   3.1 Rechercher une identité : le problème 218
   3.2 L'algorithme à écrire : chercher_identité 220
   3.3 Utilisation : effacer tous les clients d'identité donnée 223
   3.4 Définition de chercher_identité 233
   3.5 Vieillir les clients 237
   3.6 Exercices 243
4. Algorithme ou programme ? 243
   4.1 Un exemple édifiant 244
      4.1.1 Quelques maladresses : les notations 245
      4.1.2 Quand les fautes font oublier les maladresses 251
   4.2 Une solution au problème de la moyenne 253
   4.3 Compléter l'exercice : les spécifications qui manquent 260
   4.4 Compléter l'exercice : remplir le tableau 263
5. Exercices d'application 269
6. Notes bibliographiques 273
7. Résumé 274
8. Bibliographie 274

Chapitre 6
Récursivité

1. Introduction 275
2. Introduction à la récursivité : les chaînes de caractères 276
   2.1 Présentation de la récursivité 276
   2.2 Quelques exemples de spécifications récursives 282
   2.3 Exercices résolus 287
   2.4 Exercices 313
3. Les nombres et la récursivité 315
   3.1 Arithmétique 315
   3.2 Factorielle et autres exercices usés. 316
   3.3 Fractions 317
   3.4 Fonction réelle 318
4. Nombres et chaînes de caractères : édition d'un entier 319
5. Problèmes 320
   5.1 Recherche par dichotomie dans un tableau trié 320
   5.2 Palindromes. 320
   5.3 Le drapeau de Dijkstra 322
6. Résumé 324

Chapitre 7
Récursivité ou itération ?

1. Introduction 325
2. Retour sur la récursivité 326
   2.1 Premier exemple 326
   2.2 Deuxième exemple 330
   2.3 Troisième exemple 331
3. Récursivité ou itération ? 332
4. Exercices 336
5. Résumé 347

Partie 2
Chapitre 8
Trier

1. Introduction 349
2. Spécifier un algorithme de tri 350
   2.1 Présentation du problème du tri 350
   2.2 Étude de la postcondition du tri 352
3. Quelques algorithmes simples 362
   3.1 Tri par permutations : introduction 363
   3.2 Tri par permutations 366
   3.3 Tri « à bulle » (bubble sort) 368
   3.4 D'autres tris par permutations 370
4. Fusionner deux tableaux triés 384
   4.1 Définition d'un vecteur 385
      4.1.1 Définition des prédicats 387
      4.1.2 Primitives de placement dans le vecteur 388
      4.1.3 Accès aux composantes du vecteur 389
      4.1.4 Exemples 391
   4.2 Fusion de deux vecteurs triés 395
      4.2.1 Spécification de l'algorithme de fusion 397
      4.2.2 Analyse de la fusion 398
5. Exercices 401
   5.1 Tri par insertion dichotomique 401
   5.2 Un tri topologique 402
   5.3 Compléter les spécifications 402
6. Notes bibliographiques 404
7. Résumé 404
8. Bibliographie 404

Chapitre 9
Édition d'un nombre

1. Introduction 405
2. Édition d'un entier dans une base quelconque 406
   2.1 Nombre de chiffres d'un entier 406
   2.2 Résolution du problème d'édition 408
   2.3 Résolution du problème réciproque 411
3. Conversion des adresses Internet 415
   3.1 Conversion d'un entier en adresse « Internet Protocol » 415
      3.1.1 Introduction 415
      3.1.2 Conversion d'une adresse IPv4 en un entier 416
      3.1.3 Conversion d'une adresse entière en une adresse IPv4 429
   3.2 Exercice 439
4. Conversion d'un nombre entier en chiffres romains 441
5. Vérification des identifiants d'entreprises 441
6. Vérification des identifiants de livres 445
7. Résumé 447
8. Bibliographie 447

Chapitre 10
Introduction aux fichiers

1. Introduction 449
2. Notions élémentaires 450
   2.1 Fichiers et articles 450
   2.2 Organisation et accès aux fichiers 453
   2.3 Association d'un fichier physique à un programme 454

3. Organisation séquentielle 457
3.1 Introduction 457
3.2 Traitement d'un fichier séquentiel en lecture 462
3.3 Parcours d'un fichier séquentiel 466
3.4 Traitement en écriture d'un fichier séquentiel 471
3.5 Mise à jour d'un fichier à organisation séquentielle 475
4. L'organisation directe et l'accès sélectif 478
4.1 Correspondance à l'aide d'une table d'accès 479
4.2 Correspondance à l'aide d'une fonction de répartition 480
5. Problèmes 481
5.1 Statistiques d'import/export 482
5.2 Exploiter un questionnaire d'attitude 483
5.3 Exploiter les réponses à une enquête d'utilité publique 484
5.4 Rechercher les anagrammes dans un dictionnaire 485
6. Notes bibliographiques 488
7. Résumé 488
8. Bibliographie 488

Chapitre 11
Simuler

1. Introduction 489
2. Générer des nombres pseudo-aléatoires 491
2.1 Quelques générateurs 491
2.1.1 Le 147-générateur 493
2.1.2 Générateurs de Hamming 495
2.2 Tester une suite de nombres pseudo-aléatoires 499
2.2.1 Test de l'histogramme 503
3. Jeux de hasard 506
3.1 Simuler une roulette 506
3.2 Simuler un dé 509

4. Simulation de processus dynamiques 510
4.1 Propagation d'une rumeur 510
4.2 Course poursuite 512
5. Simulation statistique de phénomènes déterministes 515
5.1 Calculer π 516
5.2 Évaluer une intégrale définie 518
6. Simulation de phénomènes aléatoires 519
6.1 À la chasse aux mouches 519
6.2 Propagation d'une rumeur 525
6.3 Fiabilité des systèmes 532
6.4 Dispersion des valeurs des composants d'un circuit électronique 534
7. Notes bibliographiques 538
8. Résumé 538
9. Bibliographie 538

Chapitre 12
Crypter

1. Introduction 539
2. Intégrité 539
2.1 Présentation 539
2.2 Comparer deux empreintes 542
2.3 Condenser le contenu d'un fichier 546
3. Confidentialité 549
3.1 Principes mathématiques de la confidentialité 550
3.2 Cryptographie à clé secrète 550
3.3 Cryptographie à clé symétrique 551
3.4 Codage élémentaire : XOR 552
3.5 Codage de Vernam 554

3.6 Codage élémentaire à substitution mono-alphabétique simple 557
3.7 La méthode de Vigenère 558
4. Notes bibliographiques 560
5. Conclusion 560
6. Bibliographie 560

Index 561

Partie 2
Chapitre 8
Trier

1. Introduction

Ce chapitre présente le tri de données comparables. On considère une collection C de données de même type **`T`**, quelconque mais **`COMPARABLE`**. On veut obtenir une représentation ordonnée des éléments de la collection. Les données seront organisées le plus souvent en tableau. On ne cherche donc pas ici des tris de qualité, évalués selon leurs performances, mais plutôt un prétexte à la construction raisonnée d'algorithmes.

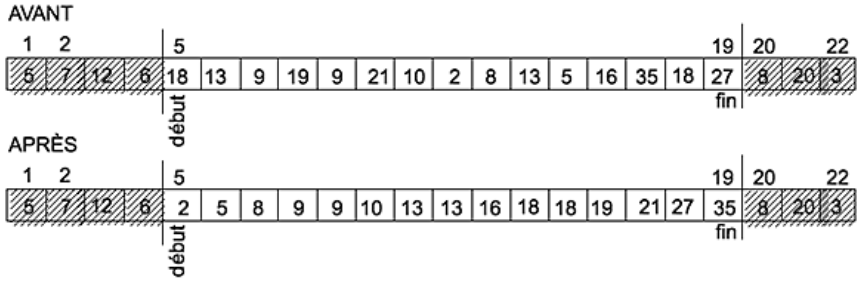
La section « Spécifier un algorithme de tri » commence par poser le problème. C'est la partie difficile du chapitre et elle peut être ignorée en première lecture. En particulier, la spécification algorithmique complète d'un algorithme de tri impose la définition des multi-ensembles invariables et de certaines opérations applicables difficiles à spécifier. La section suivante étudie quelques algorithmes usuellement définis dans une initiation à l'algorithmique. La section « Fusionner deux tableaux triés » montre comment les fusionner pour obtenir un nouveau tableau trié. La section « Exercices », enfin, propose des exercices plus difficiles.

2. Spécifier un algorithme de tri

Cette section est faite de deux parties. La première présente le problème du tri de données comparables. La seconde étudie la postcondition d'un tri. C'est la partie vraiment difficile du chapitre et elle peut être ignorée en première lecture.

2.1 Présentation du problème du tri

Soit `t` un tableau dont les composantes sont d'un type **T** dérivé de **COMPARABLE**. On veut un algorithme qui replace dans `t` ses composantes en ordre, par exemple, croissant. La figure suivante représente un tableau d'entiers avant et après le tri.



Comme d'habitude, l'algorithme à définir intervient sur une partie précisée du tableau, entre les cases de numéros `début` et `fin`. Les données sont replacées en ordre croissant dans le même tableau, qui se trouve donc modifié par l'algorithme. Par conséquent, il s'agit d'écrire une procédure. Les données en entrée sont le tableau `t` et les numéros des composantes extrêmes `début` et `fin`. La signature de la procédure et sa précondition peuvent donc être précisées :

```
Algorithme tri_ascendant
    # Trier `t[début .. fin]' en ordre croissant.

Entrée
    t : TABLEAU[T → COMPARABLE] # Le tableau à trier
    début, fin : ENTIER # Numéros des composantes extrêmes à trier
```

```
précondition
    # `début' et `fin' sont des index valides de `t'
    début ≤ fin => index_valide(t, début) et index_valide(t, fin)

    # `t[début .. fin]' a été initialisé
    est_défini(t, début, fin)
```

L'algorithme **`est_défini`** est un prédicat qui retourne `VRAI` si et seulement si le sous-tableau `t[début .. fin]` a été initialisé.

Algorithme 1 : Spécifications de ***`est_défini`***

```
Algorithme est_défini
    # `t[début .. fin]' a-t-il été initialisé ?

Entrée
    t : TABLEAU[T → COMPARABLE] # Le tableau à explorer
    début, fin : ENTIER # Numéros composantes extrêmes à vérifier

Résultat : BOOLÉEN

précondition
    # `début' et `fin' sont des index valides de `t'.
    début ≤ fin => index_valide(t, début) et index_valide(t, fin)

postcondition
    # VRAI lorsque les indices ne sont pas en ordre
    début > fin => Résultat = VRAI

    # VRAI si et seulement si les composantes de `t[début .. fin]'
    # sont non NUL i.e. sont initialisées :
    # Résultat = VRAI ou sinon (∃k ∈ ℕ)(début ≤ k ≤ fin => t[k] = NUL)
    début = fin => Résultat = (t[début] ≠ NUL)
    début < fin => Résultat =
        (
            (t[début] ≠ NUL)
          et alors
            est_défini(t, début+1, fin)
        )

fin est_défini
```

La postcondition du tri est plus difficile à obtenir. C'est elle qui est étudiée dans la section suivante.

2.2 Étude de la postcondition du tri

Cette section est difficile et peut être ignorée en première lecture. Dans ce cas, on peut passer directement à la section « Quelques algorithmes simples ».

La postcondition précise d'abord que le sous-tableau `t[début .. fin]` du tableau `t` est triée, ici en ordre croissant. La clause qui exprime cet état a déjà été écrite au chapitre « Itération ». Elle utilise le prédicat **`est_trié_en_ordre_ascendant`** dont le chapitre « Itération » a étudié une version itérative.

On obtient donc :

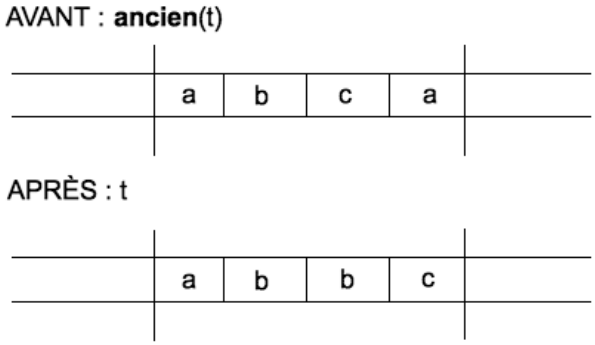
```
...
postcondition
    # `début' et `fin' ne sont pas modifiés
    ancien(début) = début
    ancien(fin)   = fin

    # t[début .. fin] est trié en ordre croissant
    est_trié_en_ordre_ascendant(t, début, fin)
...
```

Cependant, nous devons exprimer que la partie `t[début .. fin]` a certes été modifiée, mais que les mêmes éléments restent présents. Ainsi, les tableaux `t` et **`ancien`**`(t)` ont les mêmes éléments, dans les mêmes sous-tableaux, entre les cases de numéros `début` et `fin`, mais qu'ils ne sont pas nécessairement identiques puisque certaines composantes ont peut-être été déplacées. Nous ne pouvons pas écrire la condition (c1) :

```
(∀k ∈ ℕ)(début ≤ k ≤ fin)(∃i ∈ ℕ)(début ≤ i ≤ fin)
(t[k] = ancien(t)[i])
```

car il peut exister des composantes de `t` en plusieurs exemplaires entre `début` et `fin`.

Considérons, par exemple, les tableaux de la figure suivante :



Le résultat `t` est trié en ordre croissant et il passe le test exprimé par la condition (c1). Pourtant, le tri réalisé n'est pas correct puisqu'on ne retrouve pas dans `t[début .. fin]` tous les éléments de **`ancien`**`(t)[début .. fin]`. Le second exemplaire de l'élément `a` a été remplacé par un nouvel exemplaire de l'élément `b`. Nous devons pouvoir nous assurer que `t` et **`ancien`**`(t)` possèdent exactement le même nombre d'exemplaires des mêmes composantes. Pour définir un prédicat qui permettra de les comparer, on peut « éliminer », à chaque étape de la comparaison, les composantes trouvées dans les deux tableaux. Pour simplifier les notations, écrivons les composantes des tableaux en utilisant la notation ensembliste. Pour les tableaux de la figure ci-dessus, on a, initialement :

```
E = {a, b, c, a}
F = {a, b, b, c}
```

`E` et `F` sont appelés des multi-ensembles.

Définition

On appelle **multi-ensemble** un ensemble dans lequel les éléments peuvent apparaître en plusieurs exemplaires.

Pour vérifier que les deux multi-ensembles sont égaux, on compare les deux premiers éléments et on les trouve égaux. Ils sont éliminés et on obtient alors les deux multi-ensembles : {b, c, a} et {b, b, c}. Une nouvelle comparaison permet de constater que les premiers éléments de chacun d'eux sont égaux. Après élimination, on obtient {c, a} et {b, c}. Le premier élément du premier multi-ensemble est alors égal au deuxième élément du second multi-ensemble et on obtient {a} et {b}. La dernière comparaison établit que les deux multi-ensembles initiaux n'étaient pas égaux. Comme pour deux ensembles quelconques, l'égalité de deux multi-ensembles est définie à l'aide de l'inclusion :

Définition

Soient E et F deux multi-ensembles. On a : (E = F) ⇔ (E ⊆ F **et** F ⊆ E)

Autrement dit, comme pour deux ensembles quelconques, deux multi-ensembles sont égaux lorsque l'un est inclus dans l'autre et réciproquement. Soit alors **`est_égal_à`** le prédicat qui établit l'égalité de deux tableaux considérés comme des multi-ensembles. Sa spécification est :

Algorithme 2 : Spécification de l'égalité de deux multi-ensembles

```
Algorithme est_égal_à
    # `t[début_t .. fin_t]' et `u[début_u .. fin_u]' sont-ils égaux ?

Entrée
    t, u : TABLEAU[T → COMPARABLE]
    début_t, fin_t, début_u, fin_u : ENTIER

Résultat : BOOLÉEN

précondition
    index_valide(t, début_t)
    index_valide(t, fin_t)
    est_défini(t, début_t, fin_t)
    index_valide(u, début_u)
    index_valide(u, fin_u)
    est_défini(u, début_u, fin_u)

postcondition
    Résultat =
        (
```