MiniMax
Des livres qu'on lit vraiment

# Divertissements
## mathématiques et informatiques

Laurent Signac

### En un clin d'œil


| | | |
|---|---|---|
| | Mise en route | 7 |
| | **Avec des nombres** | |
| 1. | Problème de Josèphe | 17 |
| 2. | Suite de Collatz | 29 |
| 3. | Suites aliquotes | 39 |
| | **Avec des lettres** | |
| 4. | Cryptographie : substitutions monoalphabétiques | 65 |
| 5. | Cryptanalyse : analyse fréqentielle | 81 |
| | **Avec des images** | |
| 6. | Spirale d'Ulam | 99 |
| 7. | Fractales de Julia | 107 |
| 8. | Stéganographie | 125 |
| | Annexes | 142 |
| | Références | 167 |
| | Index | 173 |

# Avant-propos

Les divertissements mathématiques nous accompagnent depuis fort longtemps. De l'Antiquité à nos jours, des noms ont marqué l'histoire, la parsemant de distractions pour les esprits curieux. Souvent présentés comme de simples jeux, ces divertissements ont parfois été ancrés au cœur de problèmes particulièrement difficiles. Néanmoins, leurs auteurs ont su rendre les questionnements accessibles aux amateurs motivés. C'est encore vrai aujourd'hui avec plusieurs problèmes d'arithmétique, en apparence très simples, mais qui défient des générations de mathématiciens.

Certains problèmes et thèmes ont traversé les âges, après avoir été adaptés et réinventés ; d'autres au contraire sont nés plus récemment. Le relief de leur histoire est essentiellement lié à l'existence et au génie des quelques hommes et femmes qui les ont répandus.

La fin du xx$^{e}$ siècle nous a néanmoins offert une nouveauté particulière : l'informatique familiale. Considérablement développée depuis les années 80, elle offre à l'amateur de divertissements mathématiques un moyen d'expérimentation particulièrement efficace. Ce dernier peut passer, avec l'ordinateur, d'un rôle purement contemplatif à celui d'acteur, prenant part à la découverte, à l'exploration et à la résolution. C'est sous cet angle que les divertissements de cet ouvrage sont abordés. Au fil des trois parties qui le composent, le lecteur sera invité à expérimenter sur le thème des nombres, des lettres et des images.

À l'aide d'un ordinateur, nous revisiterons le problème de Flavius Josèphe, nous étonnerons devant la conjecture de Syracuse, et calculerons des nombres parfaits, amicaux et sociables. Le thème des lettres sera dédié à certains problèmes historiques de cryptographie et de cryptanalyse. L'ordinateur se révélera ainsi être un allié particulièrement précieux pour réaliser l'analyse fréquentielle d'un texte chiffré. Enfin, le thème des images permettra de comprendre la construction des ensembles fractals de Julia et de les explorer, avant d'utiliser des techniques de stéganographie pour dissimuler une image dans une autre.

Pour commencer la lecture de cet ouvrage, il n'est pas nécessaire de déjà savoir programmer. Les éléments de programmation qui vous seront utiles pour traiter les problèmes abordés seront présentés au fur et à mesure. Ainsi, bien que le lecteur déjà familiarisé avec la programmation puisse sans doute aborder les thèmes dans l'ordre qu'il souhaite, le néophyte complet tirera un grand avantage à lire les chapitres *dans l'ordre*. Certains aspects techniques sont en effet présentés au fil de leur découverte, dans une logique d'exploration guidée. Le lecteur désirant approfondir tel ou tel point pourra profiter de la bibliographie commentée, classée par thème en fin d'ouvrage.

Puisse le lecteur trouver autant de plaisir à découvrir et programmer ces différents thèmes que j'en ai eu à rédiger ces pages.

Laurent Signac

# La collection MiniMax

Objectifs

- Aller à l'essentiel
- Comprendre comment ça marche
- Réaliser votre projet

Vous vous lancez dans la découverte de divertissements mathématiques et dans l'apprentissage d'un nouvel outil, le langage Ruby.

Les sujets les plus intéressants ont été sélectionnés pour vous : exemples à l'appui, vous découvrirez comment ça marche et comment les explorer par vous-même.

**Suivez le guide**

Ce livre est conçu comme un guide de voyage, qui présente un aperçu du pays, des informations ciblées et des phrases de conversation.

Cet ouvrage ne vous conviendra pas si vous pensez que la bonne méthode pour découvrir la Chine, c'est de lire un dictionnaire de chinois.

Note du marketing : il convient à tout le monde !

Mais vous connaissez l'adage : « Choisir, c'est renoncer. »

– Beaucoup d'ouvrages ne renoncent à rien et deviennent des pavés indigestes.
– D'autres renoncent à tout et se réduisent à des aide-mémoire qui n'expliquent rien.

Dans les MiniMax, vous trouverez enfin les compromis équilibrés et bien construits dont nous avons tous besoin pour démarrer sur un nouveau sujet.

**Résumé**

- D'abord le plus utile, ensuite l'accessoire, et rien de superflu.
- La pratique, pas la théorie.
- 1 idée = 1 exemple.

MiniMax
Des livres qu'on lit vraiment

# Table des matières

**La collection MiniMax** **4**

**0 Mise en route** **7**
1 Utiliser Ruby sur votre ordinateur . . . . . 7
2 Calculer avec *interactive Ruby* . . . . . 9
3 Utiliser des variables . . . . . 11
4 Afficher des messages . . . . . 13
5 Poser ses conditions . . . . . 13
6 Répéter l'exécution . . . . . 14

**I - Avec des nombres . . . . . 16**

**1 Problème de Josèphe** **17**
1 Comment Josèphe survécut . . . . . 17
2 Résolution manuelle . . . . . 20
3 Utiliser les tableaux . . . . . 21
4 Résolution du problème de Josèphe . . . . . 22
5 Quelques variantes amusantes . . . . . 27

**2 Suite de Collatz** **29**
1 Une suite qui a occupé nombre de brillants esprits . . . . . 29
2 Programmation de la suite de Collatz . . . . . 31
3 Caractéristiques des vols . . . . . 34

**3 Suites aliquotes** **39**
1 Les suites aliquotes et leur construction . . . . . 39
2 Nombres parfaits, amicaux et sociables . . . . . 40
3 Les nombres premiers à la rescousse . . . . . 43
4 Recherche systématique de nombres sociables . . . . . 50
5 Nombres parfaits et nombres de Mersenne . . . . . 54

**Solutions des exercices** **55**

**II - Avec des lettres . . . . . 64**

**4 Cryptographie : substitutions monoalphabétiques** **65**
1 Manipulation des chaînes de caractères . . . . . 66
2 Préparer les chaînes de caractères . . . . . 68
3 Eh bien : chiffrez maintenant ! . . . . . 71
4 Substitutions quelconques . . . . . 75

**5 Cryptanalyse : analyse fréquentielle** **81**
1 Lire un fichier texte . . . . . 82
2 Construction de la table des fréquences . . . . . 83
3 Analyse d'un cryptogramme . . . . . 87

**Solutions des exercices** **93**

**III - Avec des images . . . . . 98**

**6 Spirale d'Ulam** **99**
1 Prendre en main Ruby/SDL . . . . . 99
2 Tracer la spirale . . . . . 102

**7 Fractales de Julia** **107**
1 Manipuler les nombres complexes . . . . . 108
2 Calculer les suites de Julia . . . . . 110
3 Tracer un ensemble de Julia . . . . . 116

**8 Stéganographie** **125**
1 Principe de la méthode LSB . . . . . 126
2 Application aux images . . . . . 131

**Solution des exercices** **137**

**Annexes . . . . . 142**

**A Installer et utiliser Ruby** **143**
1 Windows . . . . . 143
2 Mac OS X . . . . . 146
3 Linux . . . . . 149

**B Installer Ruby/SDL** **156**
1 Windows . . . . . 156
2 Mac OS X . . . . . 157
3 Linux . . . . . 160
4 Vérification de l'installation . . . . . 161

**C Calcul des nombres premiers** **162**

**D Sur l'ensemble de Julia** **164**

**E Sur les nombres binaires** **165**
1 Compter . . . . . 165
2 Convertir . . . . . 165
3 Opérations . . . . . 166

**Références** **167**

**Index** **173**

# Mise en route

0 Chapitre

Objectifs

- Savoir utiliser Ruby sur son ordinateur
- Connaître les constructions algorithmiques de base :
  - variables
  - tests
  - boucles

Ce chapitre présente les premières facettes du langage Ruby qui vous permettront de démarrer la lecture des chapitres suivants.

Vous y trouverez les prérequis indispensables et suffisants à la lecture de cet ouvrage : les notions de variables, de tests et de boucles, qui sont vraiment au cœur de toute procédure de calcul systématique.

**Approfondir sa connaissance de Ruby**

De nombreux aspects du langage ne sont pas traités, ni dans ce chapitre, ni dans ceux qui suivent. Le parti pris a été de ne présenter que les constructions qui étaient utiles à la résolution des problèmes proposés.

Si vous vous découvrez une passion pour ce langage, ce qui n'aurait rien d'étonnant, vous trouverez sur le site http://www.div-math.fr une liste d'ouvrages qui vous permettra d'approfondir vos connaissances.

## 1 Utiliser Ruby sur votre ordinateur

Vous avez deux manières d'utiliser Ruby sur votre ordinateur :

1. Installer une « distribution » du langage adaptée à votre système d'exploitation.
2. Vous connecter à une page web qui contient un programme permettant d'exécuter certains programmes Ruby.

Seule la première méthode vous permettra de suivre toutes les activités qui sont proposées dans cet ouvrage. Néanmoins, il est conseillé de commencer la lecture de manière interactive en utilisant la seconde méthode, qui vous permettra de démarrer plus rapidement la lecture.

Dans les pages qui suivent c'est donc la deuxième procédure qui est détaillée. Vous pourrez faire une installation de Ruby complète plus tard, aux environs de la page 24. Si vous en avez envie, vous pouvez toutefois y procéder dès maintenant ; pour cela, rendez-vous à l'annexe A, page 143.

Pour démarrer tout de suite, vous avez besoin d'un navigateur (Firefox, Opera, Safari, Internet Explorer...) et d'un accès Internet. La première étape consiste à vérifier que

le greffon (*plug-in*) Java est installé dans votre navigateur (et que Java est bien activé dans les préférences, ce qui est le comportement par défaut). Vous pourrez alors exécuter des applets Java [1].

L'applet Java que vous allons exécuter contient une implémentation de Ruby nommée JRuby. Elle permet de converser avec Ruby en ligne de commande, dans le navigateur.

Rendez-vous sur la page http://www.div-math.fr/jruby.html et voyez si l'applet Java s'exécute. Si c'est le cas, vous devriez voir un écran comme celui de la figure 3. En revanche, si vous n'avez pas le greffon Java, vous devriez voir apparaître l'écran de la figure 1.

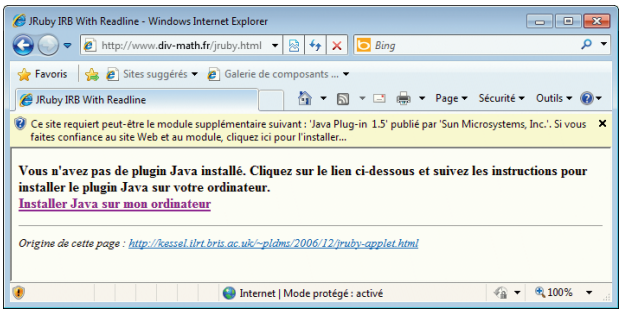
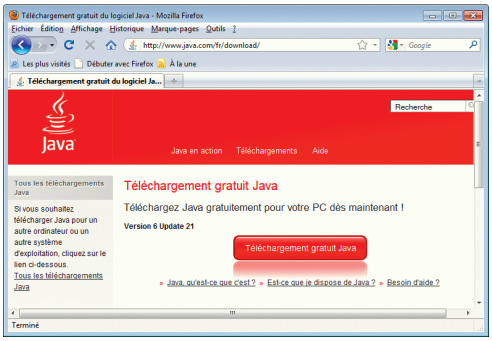



FIGURE 1 – Tentative d'affichage de l'applet sur une machine ne disposant pas de Java

Si vous n'avez pas Java, il est conseillé de l'installer en suivant le lien « *Installer Java sur mon ordinateur* ». Vous serez conduit vers une page contenant la version la plus récente de Java, dont l'installation vous permettra ensuite de l'utiliser avec tous les navigateurs de votre machine [2].




(a) Page web proposant l'installation de Java

(b) Programme d'installation de Java

FIGURE 2 – Installation de Java

L'installation de Java proprement dite ne pose pas de problème. En suivant le lien d'installation depuis le site de cet ouvrage, vous arriverez sur la page d'installation de Java (figure 2(a)). Selon le navigateur que vous utilisez, vous devrez peut-être

1. Une applet Java est une petite application, écrite avec le langage Java, qui est exécutée à l'intérieur de votre navigateur web.

2. Si vous installez Java en cliquant sur le bandeau jaune affiché par Internet Explorer, il est possible que l'installation ne vous permette ensuite pas d'utiliser des applets Java avec un autre navigateur.

# Problème de Josèphe

1 Chapitre

Objectifs

- Comprendre le problème de Josèphe et le résoudre avec un programme
- Étendre la résolution à des problèmes de décimation similaires
- Manipuler les tableaux en programmation

Après avoir détaillé le problème de Josèphe et donné son origine historique, nous écrirons un programme qui le résout. Puis, nous verrons comment rendre ce programme plus général et l'utiliserons pour obtenir la solution d'autres problèmes de décimation concernant des cartes à jouer, puis des souris.

## 1 Comment Josèphe survécut

Chargé d'organiser la défense de la Galilée, Flavius Josèphe [1] se retrouva pris dans le siège de la ville de Jotapata par l'empereur romain Vespasien en l'an 67. Il dut se retirer en cachette dans une caverne qui abritait déjà 40 nobles de la ville.

Retrouvé par les troupes romaines, il fut pressé de quitter sa retraite, et on lui promit la vie sauve. Alors qu'il s'apprêtait à se rendre, ses nobles compagnons d'infortune lui rappelèrent qu'il était leur chef et qu'ils préféraient la mort avec lui au déshonneur de la reddition.

Cependant décidé à ne pas perdre la vie [2], Flavius leur aurait proposé le macabre dénouement suivant : après avoir choisi la personne numéro 1, tout le monde se disposerait en cercle (41 personnes en tout). Puis on commencerait à compter, à partir du numéro 1 (figure 1.1(a)). Toutes les 3 positions, la personne désignée devrait mourir, de la main de la personne désignée juste après elle [3] Le dernier survivant devrait cependant se donner la mort lui-même.

En procédant ainsi seraient « supprimées » successivement les personnes en position 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36 (figure 1.1(b)). Puis 39, 1, 5, 10, 14, 19, 23, 28 (figure 1.1(c)), etc.

Flavius Josèphe avait cependant pris soin de se placer en 31$^{e}$ position dans le cercle. Trente neuf morts plus tard, il ne resta plus (figure 1.1(d)) que les deux personnes placées initialement en 16$^{e}$ et 31$^{e}$ position (Flavius). Ce dernier n'étant pas plus disposé à mourir qu'à tuer l'avant-dernier survivant, ils se rendirent tous les deux d'un commun accord.

Si l'épisode du siège de Jotapata est en partie véridique, le procédé utilisé par Josèphe a été probablement ajouté à l'histoire beaucoup plus tard, ce qui n'enlève rien à l'inté-

1. Le nom romain, qui a été retenu par l'histoire, a été attribué en 70 à Joseph ben Matthias lorsqu'il est devenu citoyen romain.
2. Dans certaines traductions, Flavius Josèphe est présenté comme un personnage couard et calculateur.
3. Évitant ainsi que chacun commette un suicide, jugé alors plus grave qu'un meurtre.

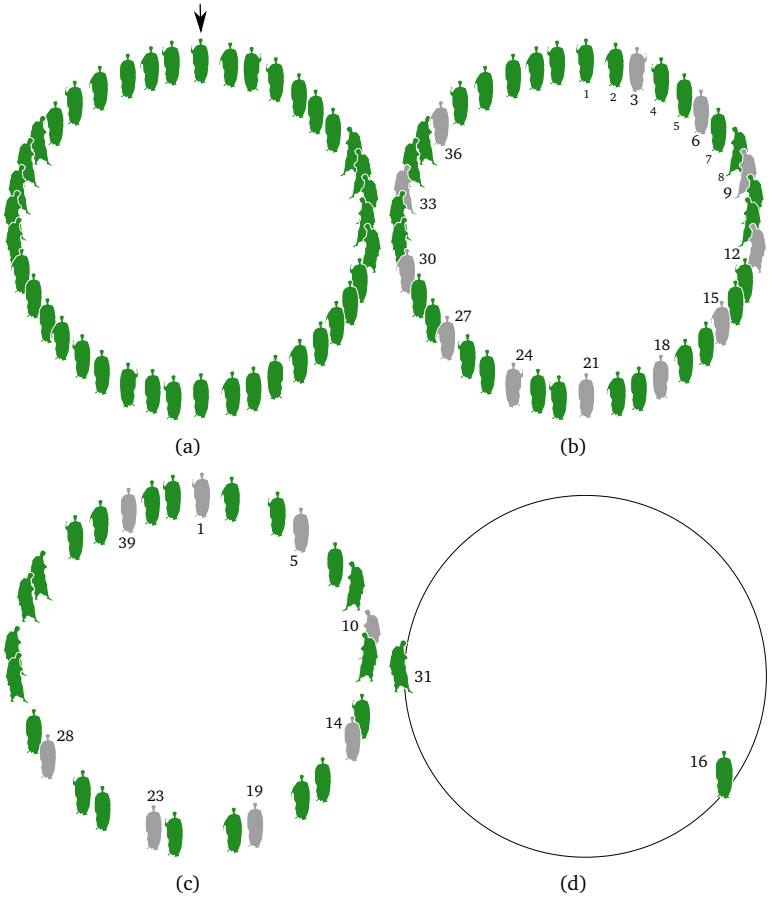



FIGURE 1.1 – Problème de Flavius Josèphe

rêt du problème. Nous allons tout d'abord voir comment le résoudre manuellement, avant de commencer à programmer.

**La véritable aventure de Josèphe**

Il est possible de consulter le texte de la *Guerre des Juifs* de Flavius Josèphe sur le site de la Bibliothèque nationale de France.

Cette version est une traduction en français de la version grecque du texte original (qui serait lui-même une adaptation, par Flavius Josèphe, de son premier texte écrit peut-être en araméen et maintenant perdu).

En voici un extrait [4] :

> *« Puisque, dit-il, nous sommes résolus à mourir, remettons-nous-en au sort pour décider l'ordre où nous devons nous entre-tuer : le premier que le hasard désignera tombera sous le coup du suivant et ainsi le sort marquera successivement les victimes et les meurtriers, nous dispensant d'attenter à notre vie de nos propres mains [...] ». Ces paroles inspirent confiance, et après avoir décidé ses compagnons, il tire au sort avec eux. Chaque homme désigné présente sans hésitation la gorge à son voisin*

## 2 Résolution manuelle

La résolution manuelle du problème est relativement facile. Elle nécessite de réaliser effectivement le processus de décimation[5] pour savoir dans quel ordre les éléments sont écartés du cercle, et finalement, quel est le dernier restant.

Pour simplifier un peu, remplaçons les 41 personnes par 8 éléments. De même que dans le problème d'origine, nous compterons de 3 en 3.
– Commençons par disposer les 8 éléments en cercle (figure 1.2(a)).
– Puis comptons 3 éléments à partir du 1 : 1, 2 et 3.
– Supprimons le 3 (figure 1.2(b)).
– Comptons à nouveau 3 éléments : 4, 5, 6.
– Supprimons le 6 (figure 1.2(b)).
– Comptons à nouveau 3 : 7, 8, 1.
– Supprimons le 1 (figure 1.2(c)).
– Continuons à compter 3 éléments, en ne prenant en compte que ceux qui restent : 2, 4, 5.
– Supprimons le 5 (figure 1.2(d)).
– Toujours la même chose : 7, 8, 2.
– Supprimons le 2 (figure 1.2(e)).

En procédant toujours de la même manière, nous sommes conduits à la suppression de l'élément 8, puis 4. Il ne reste plus que l'élément 7 (figure 1.2(f)). Si le groupe avait compté 8 individus, Josèphe aurait donc dû se placer en position 7 (la position 1 désignant celui sur lequel le compte commence).

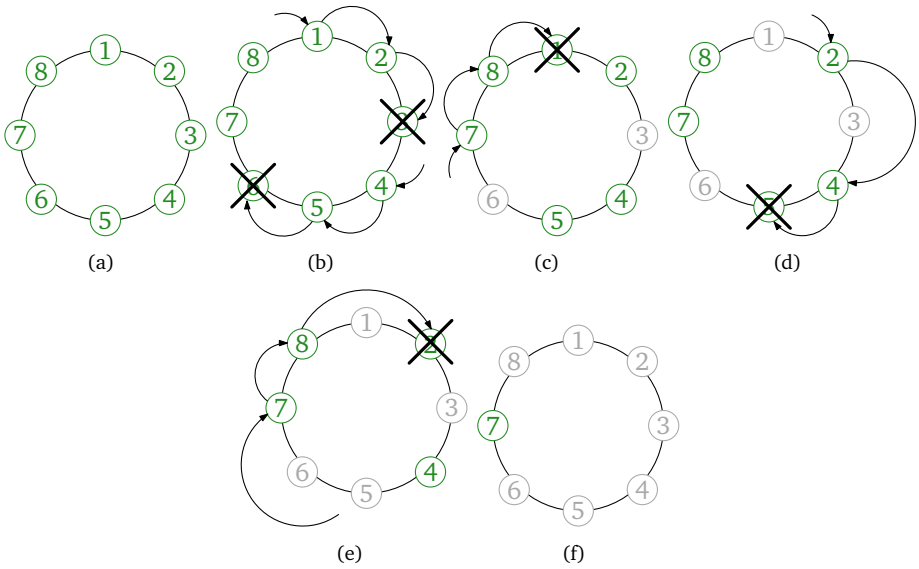



FIGURE 1.2 – « Décimation » manuelle, tous les 3 éléments, *à la Flavius Josèphe*

C'est en exécutant la procédure de décimation à la main qu'on comprend le mieux

5. Le mot « décimation », qui provient d'un châtiment romain, ne devrait être utilisé que si le compte est fait de 10 en 10 : parmi les rescapés d'une légion vaincue, on tuait une personne sur 10, au hasard. Ici, nous emploierons le terme de décimation, même si on ne compte pas de 10 en 10.

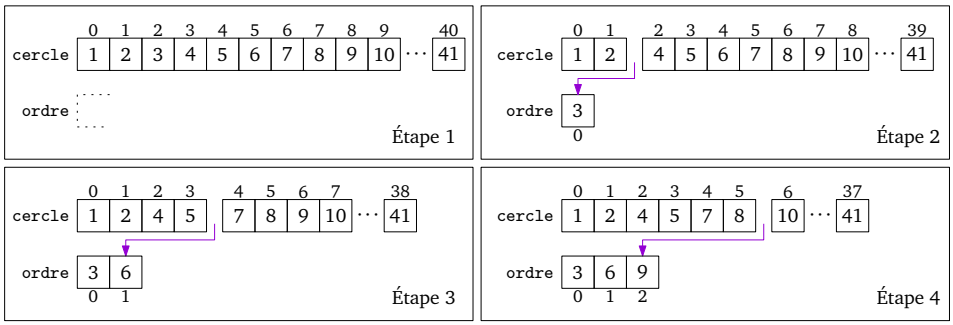



Figure 1.3 – Les quatre premières étapes de la décimation, en utilisant des tableaux pour représenter le problème

```
   35, 36, 37, 38, 39, 40, 41]
```

À présent, nous allons utiliser la variable `pos` qui contiendra le numéro de la case du tableau désignant chaque membre du cercle, et le tableau `ordre`, initialement vide, qui contiendra l'ordre dans lequel les personnes ont été écartées du cercle...

```
irb> ordre=[ ]
irb> pos=0
```

La première personne à être supprimée est la troisième, ce qui correspond à la case 2 :

```
irb> pos=pos+2
irb> ordre << cercle[pos]
=> [3]
irb> cercle.delete_at(pos)
irb> cercle
=> [1, 2,  4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
   19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
   35, 36, 37, 38, 39, 40, 41]
```

Le nombre suivant à devoir sortir du cercle est le nombre 6 qui est situé 2 cases plus loin que la case `pos` (et non 3, puisque nous venons de supprimer un élément). Nous pouvons profiter du fait que `delete_at` renvoie l'élément supprimé pour l'effacer et l'ajouter à `ordre` en une seule ligne :

```
irb> pos=pos+2
=> 4
irb> ordre << cercle.delete_at(pos)
=> [3, 6]
irb> cercle
=> [1, 2,  4, 5,  7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
   19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
   35, 36, 37, 38, 39, 40, 41]
```

Et ainsi de suite... en prenant bien garde de revenir au début du tableau si nous dépassons le dernier élément. Ce détail technique peut être géré en remplaçant la ligne `pos=pos+2` par `pos=(pos+2)%cercle.size`. Rappelons que l'opérateur `%` donne le

# Suite de Collatz

2 Chapitre

Objectifs

- Manipuler des suites et mesurer leurs caractéristiques (longueur, valeur maximum...)
- Rechercher des suites qui battent des records
- Comprendre qu'un énoncé simple peut cacher beaucoup de surprises
- Approfondir ses connaissances sur la manipulation des tableaux

La théorie des nombres réserve des problèmes qui sont fameux car extrêmement complexes à résoudre et pourtant très simples à comprendre. C'est le cas de la conjecture de Collatz présentée dans ce chapitre et des problèmes relatifs aux nombres parfaits, amicaux et sociables du chapitre suivant.

## 1 Une suite qui a occupé nombre de brillants esprits

La suite de Collatz, ou suite de Syracuse, aurait été étudiée en premier lieu par Lothar Collatz, un mathématicien allemand qui était alors étudiant, au début des années 30.

Il s'agit de construire une suite de nombres à partir d'un nombre donné $n$ :
– si $n$ est pair, le nombre suivant est $\frac{1}{2}n$ ;
– sinon, le nombre suivant est $3n+1$.
Ainsi, si nous prenons comme nombre de départ le nombre 3, les nombres suivants seront : **10** $(3\times3+1)$, **5** $(10/2)$, **16** $(3\times5+1)$, **8** $(16/2)$, **4** $(8/2)$, **2** $(4/2)$, **1** $(2/2)$, **4** $(3\times1+1)$, **2**, **1**, ...

La particularité de cette suite est que, quel que soit le nombre de départ, il semble que l'on finisse toujours par retomber sur 1, sans toutefois que la raison en soit actuellement comprise. Les suites obtenues pour un nombre de départ inférieur ou égal à 18 sont représentées sur la figure 2.1. Pour tous les nombres, on finit toujours par retomber sur 1.

**Qu'est-ce qu'une conjecture ?**

Dire que cette propriété (toujours retomber sur 1) est une conjecture, plutôt qu'un théorème, signifie que personne n'a encore pu la démontrer, ni démontrer son contraire (et pourtant, de nombreux mathématiciens ont travaillé sur le problème). La conjecture a été vérifiée pour des nombres de départ extrêmement grands, mais il a jusqu'à présent été impossible de prouver qu'elle est *toujours* vraie, ni de trouver un seul contre-exemple qui prouverait qu'elle est fausse.

La figure 2.1 nous montre aussi que certaines suites sont longues : celle issue de 18 est la plus longue suite représentée et comporte 21 nombres. D'autres atteignent des hauteurs élevées : celle issue de 15 « monte » jusqu'à 160. Ce sont ces propriétés qui vont nous intéresser par la suite : Quelles sont les suites les plus longues ? Quelle altitude atteignent-elles ?...

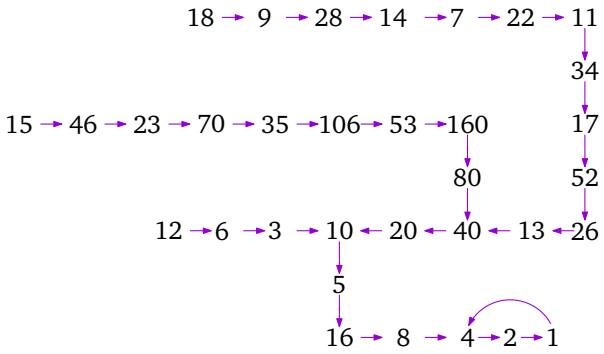



FIGURE 2.1 – Suites obtenues pour des nombres de départ jusqu'à 18

**Qui a inventé le problème ?**

La paternité de Collatz en ce qui concerne cette suite particulière est parfois remise en question. La vérité semble être qu'il aurait effectivement travaillé sur le problème (ce qui a été attesté par une lettre de lui datée de 1932), mais avec une suite différente (Collatz considérait initialement les multiples de 3 plutôt que les multiples de 2) :

$$f(3n) = 2n \qquad f(3n+1) = 4n+1 \qquad f(3n+2) = 4n+3$$

L'énoncé de ce problème est si intrigant qu'il s'est diffusé un peu partout, depuis sa découverte dans les années 30, en étant régulièrement rebaptisé. L'histoire de sa diffusion varie d'ailleurs selon les sources, ce qui ajoute un peu plus au mystère... De *problème de Collatz* dans les années 30, l'énigme a été connue durant la seconde guerre mondiale sous le nom de *problème d'Ulam*, alors que Stephen Ulam (un mathématicien polonais dont nous reparlerons au chapitre 6) la présentait à ses collègues de Los Alamos.

Quelques années plus tard, Helmut Hasse (mathématicien de l'Université de Hambourg) s'intéresse au problème et le diffuse, occasionnant le changement de nom en *algorithme de Hasse*. Puis il exportera le problème aux États-Unis, à l'Université de Syracuse, et celui-ci deviendra aussitôt *problème de Syracuse*. C'est ensuite le mathématicien japonais Shizuo Kakutani qui remit le problème sur le devant de la scène lors de conférences données dans les Universités de Yale et Chicago. Le problème était alors devenu *problème de Kakutani*. Le sujet a tellement captivé l'attention des plus brillants mathématiciens que le bruit a couru (sous forme d'une plaisanterie) qu'il avait été inventé par l'Union Soviétique pour ralentir la recherche mathématique américaine. Une rumeur similaire a évoqué un complot japonais...

Lothar Collatz est décédé en 1990, et les noms les plus répandus du problème sont maintenant : conjecture de Collatz, suite de Syracuse ou tout simplement problème $3n+1$. On retiendra la phrase du mathématicien Paul Erdös, qui en dit long sur la difficulté de la conjecture :

*Les mathématiques ne sont pas encore prêtes pour de tels problèmes.*

### Exercice : Caractéristiques des suites

solution p. 56

Les autres mesures évoquées sont laissées en exercice, dont vous trouverez une proposition de correction à la fin de cette partie. Avant de consulter la solution, essayez de programmer vous-même :

- la durée de vol en altitude, assez simple en utilisant la méthode `index` ;
- la durée de vol avant la chute, assez simple si vous pensez à utiliser la méthode `rindex`.

Le vol numéro 15 (figure 2.2) peut être utilisé pour vérifier le bon fonctionnement des programmes. Nous rappelons ici ses caractéristiques :

| Caractéristiques du vol 15 | Valeur |
|---:|:---:|
| Altitude | 160 |
| Durée de vol | 18 |
| Durée de vol en altitude | 11 |
| Durée de vol avant la chute | 14 |

Quelles sont les caractéristiques du vol 241 ?

### Exercice : Rechercher des records

solution p. 57

Lorsque vous aurez écrit ces méthodes, vous pourrez rechercher parmi les vols 1 à 999 :

- celui qui a la plus longue durée de vol en altitude ;
- celui qui a la plus longue durée de vol avant la chute ;
- celui qui a la même altitude que le vol 703 (le record est atteint deux fois parmi les vols de 1 à 999).

### Exercice : Travailler sur d'autres suites

Enfin, vous pourrez très facilement ouvrir une nouvelle page d'investigations en modifiant la suite, comme le suggère Malcom E. Lines dans *Dites un chiffre*. Remplacez par exemple $3 \times n + 1$ par $3 \times n + 7$, ce qui d'une part modifiera le cycle limite 4,2,1, et d'autre part en ajoutera d'autres (vous pouvez également essayer pour d'autres nombres impairs à la place de 1).

Vous pourrez alors essayer de décompter le nombre de cycles différents, la taille du plus grand cycle etc.

Aucune correction n'est proposée pour cet exercice.

# Suites aliquotes

3 Chapitre

Objectifs

- Chercher les parties aliquotes d'un nombre
- Calculer des nombres parfaits et amicaux
- Trouver des cycles de nombres sociables
- Améliorer le calcul des parties aliquotes en utilisant les nombres premiers

Dans ce chapitre, nous nous intéresserons à d'autres suites, appelées suites *aliquotes* et verrons ce que sont les nombres parfaits, amicaux (on rencontre parfois le terme amiables) et sociables. Ces suites ont donné lieu à une conjecture, due à Eugène Charles Catalan, que nous exposerons aussi.

## 1 Les suites aliquotes et leur construction

Chaque nombre de la suite est déduit du précédent ainsi : à partir d'un élément $n > 1$, on trouve l'élément suivant en calculant la somme des diviseurs stricts de $n$. Ces diviseurs sont aussi appelés les *parties aliquotes* de $n$, ce qui a donné leur nom aux suites.

Un diviseur strict de $n$ est un nombre $k$ strictement plus petit que $n$ tel que le reste de la division de $n$ par $k$ soit nul.

Par exemple, les diviseurs stricts de 24 sont : 1, 2, 3, 4, 6, 8, et 12 (on ne compte pas 24, puisqu'il s'agit des diviseurs *stricts*). Leur somme est égale à 36. Le terme de la suite qui vient après 24 est donc 36.

Pour connaître le terme suivant, on recherche les diviseurs stricts de 36 : 1, 2, 3, 4, 6, 9, 12, et 18. Leur somme donne 55. Le troisième terme est donc 55.

Voici la suite aliquote issue de 24 : 24, 36, 55, 17 et 1.

Le mathématicien Eugène Charles Catalan a conjecturé à la fin du XIX$^{e}$ siècle que, quel que soit le nombre de départ, on finit toujours par se retrouver dans un des deux cas suivants :

1. On finit par tomber sur 1 et le processus est arrêté car 1 n'a aucun diviseur strict. C'est ce que nous venons d'obtenir pour 24.
2. On obtient un cycle. Ce cycle peut avoir différentes longueurs.

La conjecture de Catalan exclut donc que l'on puisse avoir une suite contenant des nombres de plus en plus grands, sans jamais s'arrêter. Cette conjecture n'est pas aussi largement acceptée que celle du chapitre précédent. En particulier, le cas du nombre 276, que nous verrons bientôt, autorise à douter de sa véracité.

À titre d'exemple, toutes les suites qui commencent avec un nombre inférieur ou égal à 20 sont représentées sur la figure 3.1. Toutes finissent par atteindre 1, sauf celle qui

commence par 6. Nous allons bientôt voir que cette particularité du 6 est due à son caractère « parfait ».

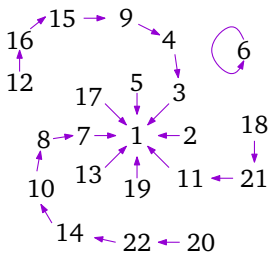



FIGURE 3.1 – Suites obtenues pour des nombres de départ entre 2 et 20

## 2 Nombres parfaits, amicaux et sociables

Pour construire des suites aliquotes puis étudier certaines de leurs propriétés, nous avons besoin d'une méthode qui calcule les diviseurs stricts d'un nombre.

Pour cela, nous pouvons essayer de diviser `n` par tous les nombres strictement plus petits que lui. Si la division tombe juste (c'est-à-dire si le reste de la division entière vaut zéro), nous ajoutons ce nouveau diviseur au tableau `divs`, initialement vide, qui contiendra tous les diviseurs de `n`. Entrez la méthode suivante dans le nouveau fichier `aliquotes.rb` :

**aliquotes.rb**

```
def parties_aliquotes1(n)
  divs=[ ]
  for i in 1..(n-1)
    divs<<i if n%i==0
  end
  return divs
end
```

Notez ici :
– l'utilisation de `if` comme *modifieur* (nous en avons parlé page 31) ;
– l'emploi de l'opérateur `<<` qui ajoute un élément en fin de tableau (voir page 21) ;
– l'emploi de l'opérateur `%` qui donne le reste de la division entière.

**Ruby : deux types d'intervalles**

Il existe deux types d'intervalles en Ruby : `1..20` qui contient tous les nombres de 1 à 20 inclus ou `1...20` (avec trois points) qui exclut la borne supérieure. L'intervalle `1..(n-1)` pourrait donc aussi être écrit `1...n`.

Vérifions que notre méthode fonctionne correctement :

```
irb> load "aliquotes.rb"
irb> parties_aliquotes1(6)
```

# Solutions des exercices

## Un chat et des souris

énoncé p. 28

Pour résoudre le problème du chat, une seconde méthode vous est proposée. Elle s'appelle tout simplement `chat` et prend en paramètre le nombre `n` de souris. Exécuter `chat(37)` affichera toutes les valeurs de `k` pour lesquelles le chat engloutit 37 souris en terminant par la souris blanche.

**josephe.rb**

```
# Résolution du problème du chat et
# des souris

def chat(n)
 for k in 1..n
   ordre=josephe(n,k)
   if ordre.last==1
      puts k
   end
 end
end

# Résolution du problème de Josèphe

def josephe(n,k)
  cercle=Array.new(n) {|i| i+1}
  ordre=[ ]
  pos=0
  while cercle.size>0
    pos=(pos+k-1)%cercle.size
    ordre << cercle.delete_at(pos)
  end
  return ordre
end
```

Voici la solution obtenue :

```
irb> load "josephe.rb"
irb> chat(37)
5
21
30
34
=> 2..37
```

Les valeurs de `k` qui permettent de terminer par la souris blanche sont : 5, 21, 30 et 34. Puisque c'est la plus petite qui est demandée, **le chat doit donc compter de 5 en**

# Cryptographie : substitutions monoalphabétiques

4 Chapitre

Objectifs

- Connaître le chiffre de César
- Connaître l'Atbash
- Construire des substitutions monoalphabétiques quelconques
- Chiffrer et déchiffrer des messages

La dissimulation d'informations et les affaires militaires ont, cela n'étonnera personne, un passé commun assez riche. Une des plus anciennes méthodes de cryptographie [1] attestée par l'histoire est la méthode appelée aujourd'hui « chiffre de César ».

Suétone relate dans *Vie des douze Césars* la manière dont le premier César (Jules) chiffrait sa correspondance [2] :

> *On possède enfin de César des lettres à Cicéron, et sa correspondance avec ses amis sur ses affaires domestiques. Il y employait, pour les choses tout à fait secrètes, une espèce de chiffre qui en rendait le sens inintelligible (les lettres étant disposées de manière à ne pouvoir jamais former un mot), et qui consistait, je le dis pour ceux qui voudront les déchiffrer, à changer le rang des lettres dans l'alphabet, en écrivant la quatrième pour la première, c'est-à-dire le d pour l'a, et ainsi de suite.*

Pour ajouter à la confusion, il lui arrivait même parfois, en plus de ce décalage, de remplacer les lettres latines par des lettres grecques, comme le mentionne Dion Cassius dans *Histoire Romaine* [3] :

> *[...] et écrivit en grec tout ce qu'il voulait faire savoir à Cicéron. De cette manière, sa lettre, vînt-elle à tomber entre les mains des barbares qui ne savaient pas encore le grec, ne leur apprendrait rien. Il avait d'ailleurs l'habitude, quand il communiquait un secret par écrit, de remplacer toujours la lettre qu'il aurait dû mettre la première par celle qui, dans l'ordre alphabétique, vient la quatrième après elle, afin que ce qu'il écrivait ne pût être compris par le premier venu.*

Le chiffre de César, très simple, fait partie des *substitutions monoalphabétiques*, ce qui signifie deux choses :
– les lettres sont *remplacées* par d'autres symboles (substitution) ;
– une lettre donnée est toujours remplacée par *le même* symbole (monoalphabétique).

1. La cryptographie est la manière de rendre une information inintelligible pour des personnes non autorisées.
2. Traduction M. Nisard, 1855.
3. La lecture du passage qui suit peut laisser penser qu'avant la substitution, les missives étaient *écrites* en grec. Cependant, les sources ne permettent pas de décider si, dans ces circonstances, César écrivait en grec ou s'il utilisait l'alphabet grec pour écrire des messages en latin. Peut-être utilisait-il l'une ou l'autre complication, agrémentée ou pas du décalage de lettres, selon les destinataires et les occasions.

Une fois la substitution réalisée, nous obtenons un *cryptogramme*.

Notons que l'ordre des lettres n'est pas modifié : la première lettre du cryptogramme correspond à la première lettre du message d'origine et ainsi de suite. Si l'ordre des symboles était affecté par la méthode de chiffrement, nous aurions affaire à une *transposition*.

Le tableau ci-dessous indique, dans le cas du chiffre utilisé par César, comment remplacer une lettre *en clair* par une lettre *chiffrée* :

| Clair | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Chiffré | D | E | F | G | H | I | J | K | L | M | N | O | P |
| Clair | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Chiffré | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |

Le message « Avé César » deviendra donc « Dyh Fhvdu ».

L'objectif de ce chapitre est de chiffrer des chaînes de caractères (que nous appellerons parfois simplement « chaînes » dans la suite) avec différentes substitutions monoalphabétiques, dont le chiffre de César.

Nous verrons aussi qu'à une substitution particulière correspond une autre substitution qui permet de déchiffrer le texte.

Avant de passer au chiffrement proprement dit, voyons comment manipuler les chaînes de caractères.

# 1 Manipulation des chaînes de caractères

Commençons par une session `irb` pour comprendre l'utilisation des chaînes :

```
irb> s="Il est morne et taciturne."
irb> puts s
Il est morne et taciturne.
irb> s[7]
=> 109
irb> s[7..14]
=> "morne et"
irb> s[7..7]
=> "m"
irb> s[7..14]="morne, il est"
=> "morne, il est"
irb> puts s
Il est morne, il est taciturne.
irb> s << " Il préside aux choses du temps."
irb> puts s
Il est morne, il est taciturne. Il préside aux choses du temps.
```

La session précédente nous aide à comprendre les points suivants :
– Une chaîne de caractères est entrée entre guillemets.
– Une chaîne de caractères peut être indicée à la manière d'un tableau : il est ainsi possible d'accéder à chaque caractère séparément en donnant simplement son nu-

# Cryptanalyse : analyse fréquentielle

5 Chapitre

Objectifs

- Connaître la méthode d'analyse de fréquences
- Être capable de déchiffrer n'importe quelle substitution monoalphabétique

Le nombre de substitutions monoalphabétiques obtenues avec des permutations de l'alphabet est considérable [1]. Cependant, ces substitutions ne peuvent résister à une technique de cryptanalyse rapportée dans un ouvrage du IXe siècle dû au philosophe arabe connu en Occident sous le nom de *Al-Kindi* (Abu Yusuf Yaqub Al-Kindi était aussi mathématicien, physicien, astronome, médecin, philosophe, musicien...)

Nous allons voir dans le présent chapitre comment appliquer cette méthode.

Nommée *analyse fréquentielle*, cette technique de cryptanalyse utilise le fait que, dans les langues basées sur des lettres, celles-ci n'apparaissent pas au hasard : certaines lettres, ainsi que certaines suites de lettres, apparaissent plus souvent que d'autres.

Ces fréquences d'apparition dépendent de la langue. En français, la lettre la plus fréquente est le `E`. Dans un texte, environ une lettre sur six est un `E`. Les lettres les moins fréquentes sont le `K` et le `W`. Il faut plus de 2 000 lettres pour avoir de bonnes chances de trouver un `W`.

Puisqu'une substitution monoalphabétique transforme une lettre de manière toujours identique, il suffit de repérer dans le texte crypté la lettre qui revient le plus souvent. Il y a de bonnes chances pour qu'elle corresponde au `E`. Les autres lettres fréquentes sont `A`, `N`, `S`, `T` et `I`. Chaque nouvelle lettre découverte en utilisant la méthode des fréquences nous fait progresser dans la voie du déchiffrement.

En plus des fréquences des lettres seules, nous pouvons utiliser les fréquences des bigrammes (deux lettres consécutives) ou des trigrammes. Le bigramme le plus fréquent en français est `ES`.

L'analyse fréquentielle ne peut être utile que si l'on dispose d'une quantité suffisante de données. Nous allons donc être amenés à manipuler de longues chaînes de caractères qu'il est hors de question de « retaper ». Par exemple, le texte qui servira de base à nos statistiques comporte un demi-million de caractères. Nous verrons donc dans un premier temps comment, avec Ruby, charger le contenu d'un fichier entier dans une chaîne de caractères.

Puis, après avoir construit notre propre table de fréquences, basée sur le roman de Jules Verne *La Jangada*, nous l'utiliserons pour analyser un texte chiffré d'environ 450 caractères et déduirons de cette analyse la totalité du texte en clair et la quasi-totalité de la clé de chiffrement.

1. Nous avons vu au chapitre précédent qu'il y en avait pratiquement 400 millions de milliards de milliards qui avaient la particularité de bien brouiller les textes.

Nous constatons une grande similitude entre ces trois sources d'informations. L'histogramme de la figure 5.1 qui indique le pourcentage d'apparition de chaque lettre selon chacune des trois sources illustre encore plus précisément la constance des fréquences d'apparition des lettres

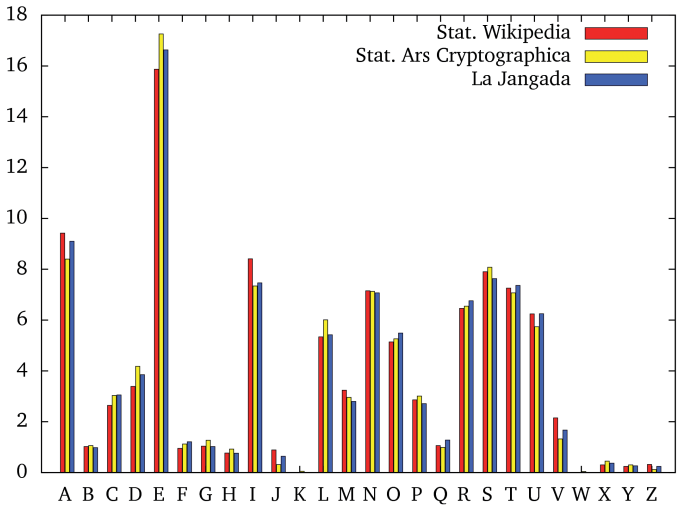



Figure 5.1 – Pourcentage d'apparition de chaque lettre dans un texte français, selon trois sources

La conclusion de cette comparaison est que l'ordre des lettres que nous venons d'obtenir est relativement constant quel que soit le texte en français utilisé, pourvu qu'il soit d'une bonne longueur.

Naturellement, nous pourrions améliorer nos propres statistiques en prenant plusieurs textes et surtout en diversifiant les sources (c'est ce qui a été fait sur le site Ars Cryptographica). L'utilisation d'un seul roman et donc l'apparition récurrente des mêmes personnages (et donc de leur nom) peut en effet fausser légèrement les statistiques.

Nous disposons maintenant d'un outil puissant qui, à partir d'un texte, nous permet d'en classer les lettres par fréquence d'apparition décroissante. Ce n'est pas rien... En utilisant cet outil et la méthode `substitution` du chapitre précédent, nous allons pouvoir nous « attaquer » à un texte crypté de quelques centaines de caractères et essayer de le déchiffrer.

## 3 Analyse d'un cryptogramme

Pour tester nos méthodes, nous allons utiliser un cryptogramme qui est disponible sur le site web de cet ouvrage [4].

En voici une copie :

```
SHMHM WZORO ZOBNL OLZFM ODDOL URTDF MOPHR DOZHF UOODU NLLNL
```

4. http://www.div-math.fr

# Spirale d'Ulam

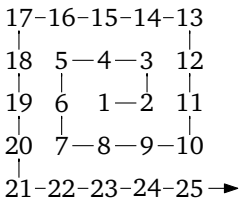
6 Chapitre

Objectifs

- Tracer la spirale d'Ulam
- Se familiariser avec la bibliothèque SDL

Comprendre la répartition des nombres premiers parmi les nombres entiers fait partie d'un domaine très sérieux des mathématiques : la théorie des nombres. De ce domaine de recherche découle, sans que les précurseurs s'y soient attendus, des applications extrêmement pratiques, comme la sécurité des cartes bancaires car leurs algorithmes cryptographiques utilisent justement les nombres premiers.

Bien que les problèmes de théorie des nombres soient souvent très difficiles à résoudre, il est possible à chacun de visualiser certaines propriétés. En particulier, dans les années 1960, le mathématicien Stanislaw Ulam inventa la spirale qui porte maintenant son nom (figure 6.1). Les nombres entiers sont écrits en spirale, et on colorie un petit carré à l'emplacement du nombre uniquement si celui-ci est premier.




(a) Les nombres sont placés en spirale

| | | | | |
|---|---|---|---|---|
| 17 | 16 | 15 | 14 | 13 |
| 18 | 5 | 4 | 3 | 12 |
| 19 | 6 | 1 | 2 | 11 |
| 20 | 7 | 8 | 9 | 10 |
| 21 | 22 | 23 | 24 | 25 |

(b) Les nombres premiers sont noircis

FIGURE 6.1 – Construction de la spirale d'Ulam

À l'aide d'un ordinateur, il est possible de construire cette image en considérant les nombres entiers jusqu'à 100 000 par exemple, ce que nous allons faire à présent. Cette image est une manière de représenter la répartition des nombres premiers.

Après une prise en main rapide de la bibliothèque graphique, nous nous intéresserons au parcours et au tracé de la spirale d'Ulam.

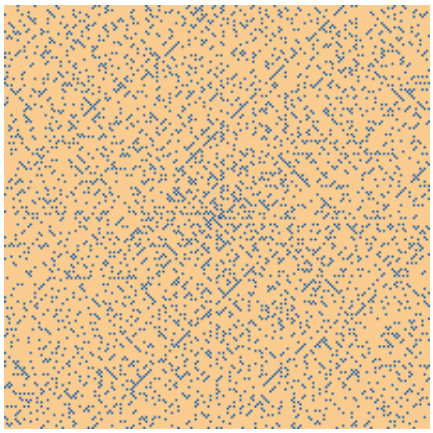
## 1 Prendre en main Ruby/SDL

L'utilisation de la bibliothèque SDL conjointement avec `irb` n'est actuellement pas possible avec Mac OS. Aussi, dans cette partie, nous entrerons systématiquement nos programmes graphiques dans un fichier que nous exécuterons :
– avec l'interpréteur `ruby` sous Windows ou Linux ;
– avec l'interpréteur modifié `rsdl` sous Mac OS (voir page 160).

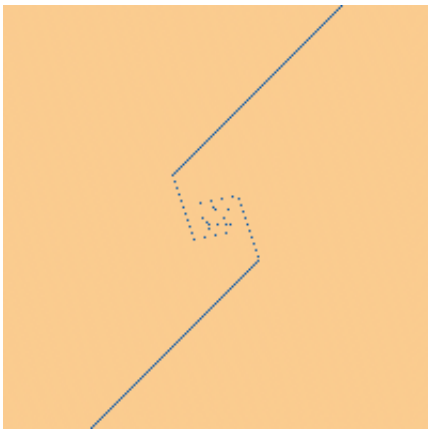
```
> ruby ulam.rb
```

Si vous utilisez l'éditeur Scite et que le fichier `ulam.rb` est ouvert, vous pouvez appuyer simplement sur F5 pour l'exécuter. Vous devriez alors obtenir une image semblable à la figure 6.4(a).

La spirale d'Ulam présente des alignements de points qui ne semblent pas dus au hasard. Les « inventeurs » de la spirale font remarquer que l'apparence des structures qui apparaissent est différente de celle d'un ensemble aléatoire qui aurait la même densité que les nombres premiers. Autrement dit, les nombres premiers ne sont pas disposés au hasard. S'il est facile de comprendre pourquoi certaines lignes ne comportent aucun nombre premier (il y a des lignes composées uniquement de nombres pairs, ou des diagonales uniquement composées de carrés parfaits), il est en revanche plus mystérieux d'observer des lignes particulièrement riches en nombres premiers. Par exemple, la formule due à Euler : $n^2 + n + 41$ produit beaucoup de nombres premiers. Pour $n < 1000$, plus d'un nombre sur deux obtenu par cette formule est premier. Or, dans la spirale d'Ulam, ces nombres sont disposés de manière visuellement repérable (voir figure 6.4(b)). Il est donc normal d'observer les nombres d'Euler dans la spirale d'Ulam. D'autres alignements, en revanche, n'ont pas été expliqués.



(a) Spirale d'Ulam



(b) Disposition des nombres de la forme $n^2 + n + 41$ dans la spirale d'Ulam

Ce chapitre étant une introduction à l'utilisation de la bibliothèque SDL, il ne contient pas d'exercice. Que cela ne vous empêche pas d'expérimenter seuls sur le thème de la spirale d'Ulam avant de passer au tracé d'images fractales, dans le chapitre suivant.

# Fractales de Julia

7 Chapitre

Objectifs

- Tracer des images fractales
- Jongler avec les palettes de couleur

Les images fractales sont parmi les objets mathématiques les plus connus du grand public, en raison de leur beauté, mais aussi de la relative simplicité de leur génération.

Après un bref historique sur les fractales et quelques mots sur l'utilisation des nombres complexes, nous détaillerons les suites qui sont utilisées pour construire les ensembles de Julia. Puis nous passerons au tracé proprement dit et, enfin, apporterons diverses améliorations au programme initial.

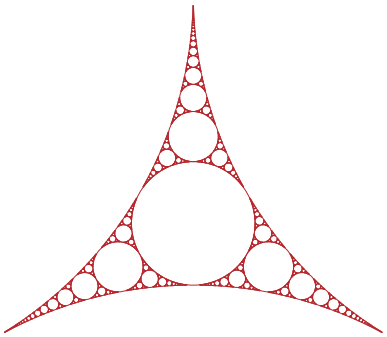


FIGURE 7.1 – La baderne d'Apollonius est obtenue en construisant un nouveau cercle tangent à trois cercles tangents (qui forment un triangle curviligne), puis en recommençant.

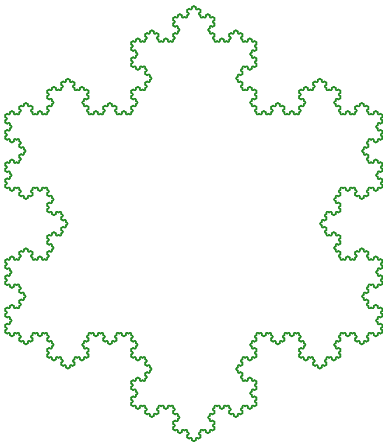
Si le terme même d'objet « fractal » est relativement récent, les objets auto-similaires (l'auto-similarité est étroitement liée au caractère fractal d'un objet) ont une histoire qu'on peut faire remonter au IIIe siècle avant Jésus-Christ avec la baderne d'Apollonius de Perge (figure 7.1). Cet objet est obtenu en construisant un cercle inscrit dans un triangle curviligne (un triangle dont les côtés sont des arcs de cercle). L'ajout de ce cercle forme trois nouveaux triangles curvilignes, dans lesquels on trace le cercle inscrit et ainsi de suite...

L'étrangeté de ces structures, pendant longtemps considérées comme des curiosités plutôt que comme des objets dignes d'une théorie mathématique, a régulièrement attiré l'attention au fil des siècles.

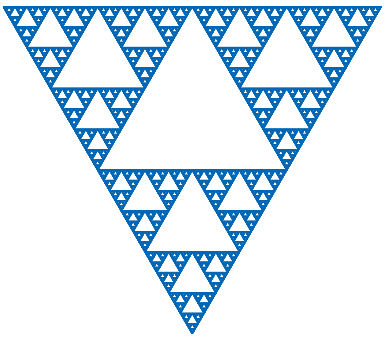
À la fin du XIXe siècle, simultanément à un intérêt grandissant pour la notion d'infini, de nombreux objets présentant la propriété d'auto-similarité ont été inventés ou réinventés. Les plus célèbres sont peut-être le flocon de von Koch (figure 7.2(a)) et le triangle de Sierpinski (figure 7.2(b)).

Au début du XXe siècle, Gaston Maurice Julia [1] travaillait sur des sujets liés à l'itération de fonctions complexes. Il ne disposait néanmoins pas des moyens techniques d'aujourd'hui (ordinateur avec écran graphique) qui lui auraient permis de représenter les objets qu'il manipulait.

1. Des travaux similaires ont été menés indépendamment et en même temps par Louis Pierre Fatou.



(a) Chacun des trois côtés du flocon de von Koch (initialement un triangle) est obtenu par quatre répétitions du même motif à l'échelle $\frac{1}{3}$.



(b) Le triangle de Sierpinski est obtenu en répétant trois fois le motif à l'échelle $\frac{1}{2}$.

FIGURE 7.2 – Le flocon de von Koch et le triangle de Sierpinski

C'est durant les années 1970 que Benoît Mandelbrot, lors de son travail sur les fractales (il venait alors d'inventer le terme), a remis au goût du jour les travaux de Julia.

Un peu plus tard, avec le développement de l'informatique personnelle, chacun a pu obtenir avec son ordinateur des images que leur inventeur n'avait peut-être pas imaginées si belles. Pour en savoir plus sur les fractales, vous pouvez consulter les liens web et références en fin d'ouvrage. Nous allons à présent nous focaliser sur les fractales de Julia, dont un exemple est donné figure 7.6 page 120.

Le tracé d'un ensemble de Julia est réalisé selon le principe suivant :
– associer à chaque pixel de l'écran un nombre ;
– décider si ce nombre fait partie de l'ensemble de Julia ou non ;
– choisir la couleur du pixel en conséquence.
Avant d'en arriver au tracé de l'image, nous allons devoir franchir quelques étapes. La première est la manipulation des nombres complexes. La seconde consiste à étudier un peu plus précisément une suite de nombres particulière, sur laquelle l'ensemble de Julia est construit.

## 1 Manipuler les nombres complexes

Ruby possède une classe pour représenter et manipuler les nombres complexes ; il n'est donc pas nécessaire de tout savoir sur ces nombres pour parvenir à nos fins. Nous nous contenterons des quelques connaissances pratiques suivantes :

1. Un nombre complexe est « composé » de deux nombres réels, qui sont appelés partie réelle et partie imaginaire.
2. Les règles de calcul (addition, multiplication…) sont différentes des règles habituelles, mais Ruby saura les utiliser à notre place puisqu'il sait manipuler les nombres complexes.

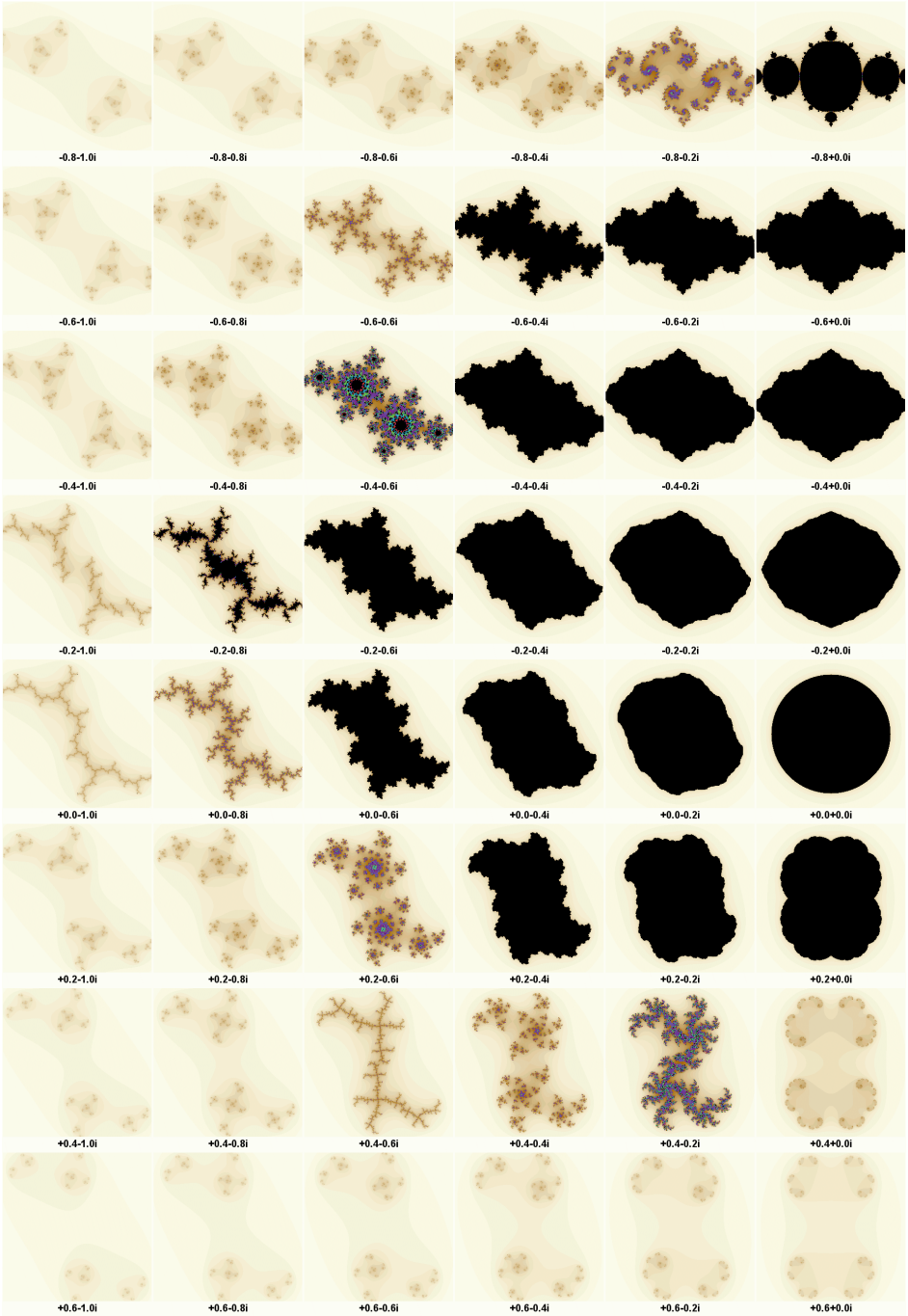



Figure 7.7 – Ensembles de Julia obtenus pour des valeurs de $c$ typiques

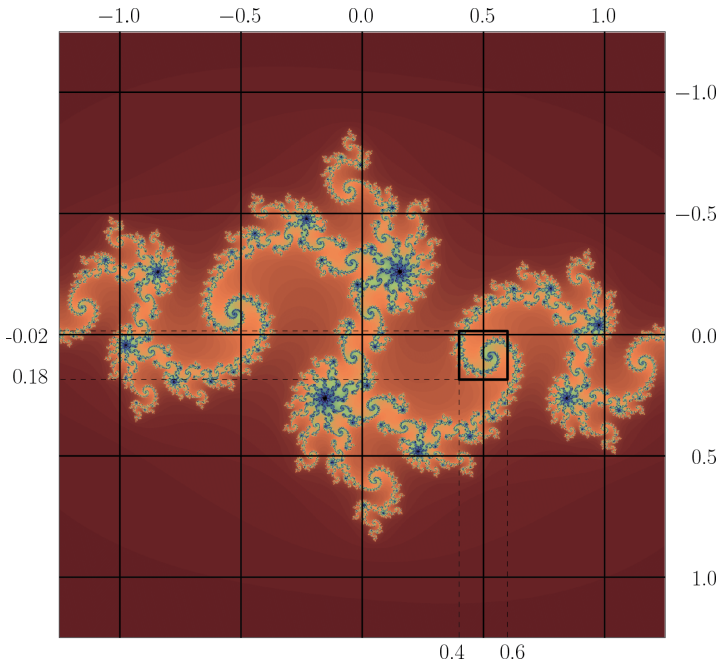



Figure 7.9 – Ensemble de Julia obtenu pour $c = -0.8 - 0.18\,i$. La grille nous permet de mesurer approximativement les bornes des futurs tracés

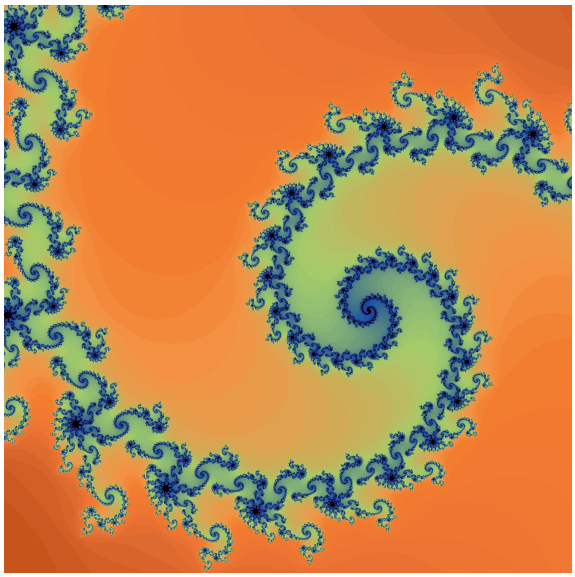


Figure 7.10 – Image obtenue pour $c = -0.8 - 0.18\,i$ en zoomant sur le petit carré de la figure 7.9

# Stéganographie

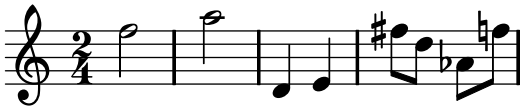
8 Chapitre

Objectifs

- Comprendre ce qu'est la stéganographie
- Manipuler les nombres binaires
- Appliquer la méthode LSB
- Écrire un programme qui dissimule une image dans une autre

La stéganographie est souvent associée à la cryptographie, bien qu'elle en diffère sur le point suivant : alors que la cryptographie a pour objectif de dissimuler *le sens* d'un message, la stéganographie tente d'en dissimuler *l'existence*.

Imaginons que vous trouviez un document sur lequel serait inscrit : « IPDCJIJP ». Vous seriez sans doute intrigué. Vous auriez probablement des difficultés (car il est trop court) à deviner que le message a été chiffré avec le chiffre de Vigénère[1] en utilisant le mot-clé « Ruby », et qu'il signifie « RV CE SOIR ». Mais vous n'en resteriez pas moins convaincu qu'un message secret a été échangé et cette simple information est importante.

En revanche, en trouvant un document contenant ceci :

il est probable que votre curiosité ne soit pas attirée. Tout au plus, vous remarquerez qu'il existe de mauvais compositeurs et considérerez que votre trouvaille est anodine.

Or, en associant la lettre `A` à la note do3 (le do situé au-dessous de la portée), puis chaque lettre suivante successivement à la note située un demi-ton plus haut (`B` à do#, `C` à ré...), nous obtenons à nouveau le message « RV CE SOIR ». De plus, chaque mot est représenté par une mesure exactement. Ce message est beaucoup plus facile à « déchiffrer » que le précédent étant donné qu'il n'y a pas vraiment de chiffrement, mais une simple dissimulation du message. En revanche, la présence du message est difficile à détecter pour qui ne l'attend pas. De la même manière, il est probable qu'un lecteur moyennement attentif laissera échapper le fait que dans ce paragraphe, le message « rv ce soir » est formé par les lettres en italique disséminées çà et là[2]...

Une autre méthode de stéganographie, particulièrement connue, est adaptée aux fichiers numériques. Cette méthode, nommée LSB (pour *Least Significant Bits*, c'est-à-dire « bits de moindre importance »), décrit de quelle manière un nombre peut être caché dans un autre. Étant donné que sur un ordinateur, image, son et vidéo sont représentés par des nombres, la méthode LSB peut s'appliquer dans ces trois domaines.

1. Vous apprendrez ce qu'est le chiffre de Vigénère en consultant par exemple : http://www.apprendre-en-ligne.net/crypto/vigenere/carrevig.html.

2. Même en le sachant, c'est assez pénible à vérifier. Cette technique a été utilisée il y a quelques années dans une contexte très sérieux (voir les liens au sujet du *Smithy Code* sur http://www.div-math.fr).

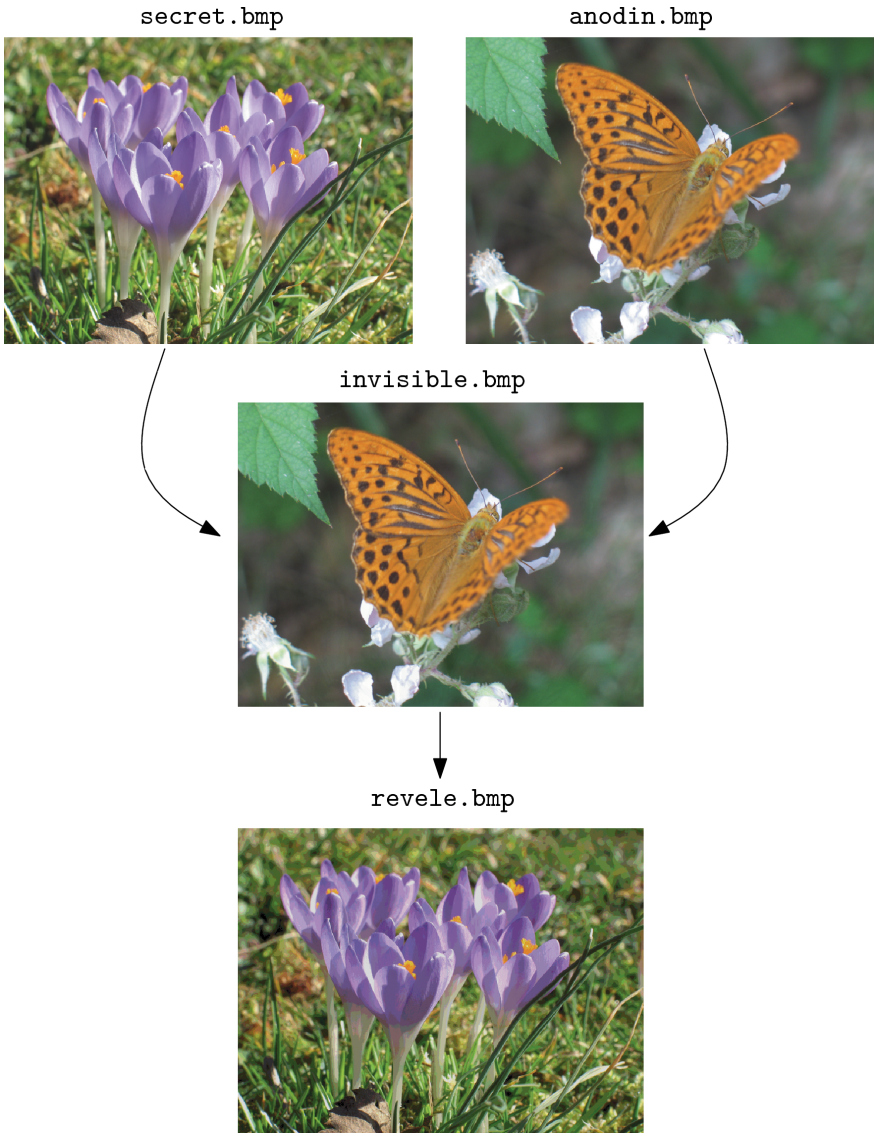



Figure 8.1 – Les deux images de départ `secret.bmp` et `anodin.bmp`, puis l'image `invisible.bmp`, résultat de la dissimulation des crocus dans le papillon et enfin, l'image extraite `revele.bmp`

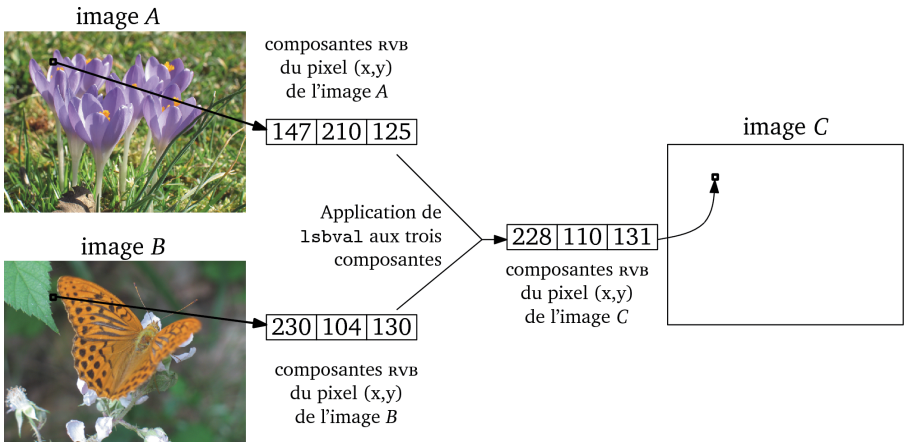



FIGURE 8.6 – Comment cacher l'image $A$ dans l'image $B$, le résultat étant l'image $C$

vient donc à cacher le premier triplet dans le second. Pour cela, il suffit d'opérer composante par composante (avec la méthode `lsbval`).
– pour le rouge, dissimuler 147 dans 230 donne 228
– pour le vert, dissimuler 210 dans 104 donne 110
– pour le bleu, dissimuler 125 dans 130 donne 131
On obtient ainsi un nouveau triplet : $(228, 110, 131)$, dont les valeurs sont assez proche de celui de l'image $B$, et qui constituera la couleur du pixel $(x, y)$ de l'image $C$. Inversement, lorsqu'on dispose de l'image $C$ et que l'on désire reconstituer l'image secrète ($A$), on procède comme indiqué sur la figure 8.7. Pour un pixel de coordonnées $(x, y)$, il faut extraire ses composantes RVB sur l'image $C$ : $(228, 110, 131)$ dans l'exemple. Puis on extrait le nombre caché dans chacune des trois composantes (méthode `unlsbval`) :
– pour le rouge, on extrait 144 de la valeur 228
– pour le vert, on extrait 208 de la valeur 110
– pour le bleu, on extrait 112 de la valeur 131

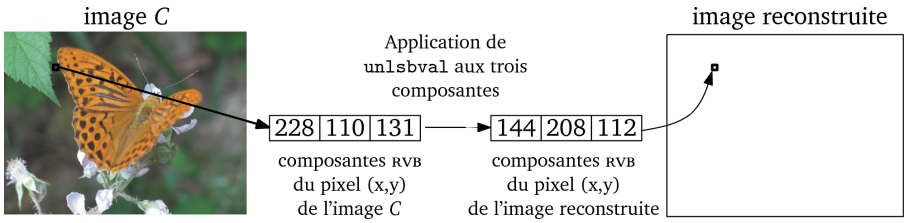



FIGURE 8.7 – À partir de l'image $C$, on reconstitue de manière approximative l'image $A$.

On obtient ainsi le triplet reconstruit $(144, 208, 112)$ qui servira de couleur pour le pixel $(x, y)$ de l'image reconstruite. On note au passage que le triplet reconstruit $(144, 208, 112)$ est assez proche (mais différent) du triplet de départ $(147, 210, 125)$ de l'image $A$.

Techniquement, nous avons donc besoin d'accéder à chaque pixel d'une image, et à chaque composante d'un pixel. Voyons à présent comment réaliser cela avec Ruby.

# Installer et utiliser Ruby

A
Chapitre

Objectifs

- Installer Ruby sur son ordinateur
- Savoir entrer et exécuter un programme Ruby

Pour travailler confortablement, vous aurez besoin d'une distribution de Ruby et d'un éditeur de texte. Les sections suivantes détaillent comment installer et configurer vos futurs outils pour les systèmes Windows, Mac OS et Linux.

## 1 Windows

### 1.1 Ruby *Installer*

L'installation de Ruby *Installer* vous donnera accès à l'interpréteur Ruby, au programme *Interactive Ruby* (`irb`) et au programme `gem`.

Commencez par télécharger le logiciel [1] :
– soit depuis le site d'origine http://rubyinstaller.org (figure A.1) ;
– soit depuis http://www.div-math.fr.

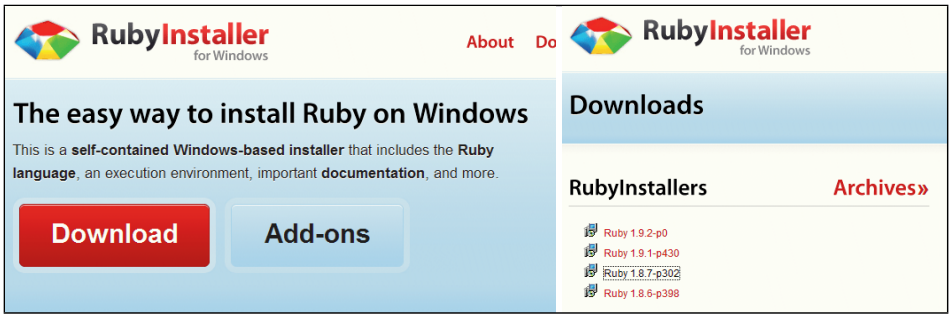



FIGURE A.1 – Accéder au programme d'installation à partir du site rubyinstaller.org

Exécutez à présent le fichier téléchargé. Vous devriez voir apparaître les écrans d'installation de la figure A.2.

Pensez à cocher les deux cases :
– *Add Ruby executables to your PATH* ;
– *Associate .rb and .rbw files with this Ruby Installation* ;
Une fois l'installation terminée, le logiciel sera accessible dans le menu Démarrer de votre machine (figure A.3).

1. En 2011, préférez la version 1.8.x, stable, à la version 1.9.x, en développement.

# Sur les nombres binaires

E
Chapitre

Objectifs

- Savoir lire des nombres écrits en binaire
- Convertir de décimal vers binaire et l'inverse
- Connaître les opérations logiques : et, ou

Compter en base 10 est naturel pour nous, mais le choix de cette base est assez arbitraire et tient probablement au fait que nous possédons 10 doigts (nos ancêtres ont donc trouvé naturel de « grouper » par paquets de 10). Mais il est possible de représenter les nombres et de les manipuler dans n'importe quelle base. En particulier, la base 2 joue un rôle important en informatique.

## 1 Compter

Si la base 10 comporte 10 chiffres (de 0 à 9), la base 2 en a donc 2 (0 et 1). Pour compter en base 2, il faut procéder comme pour la base 10. En base 10, nous comptons 0, 1, 2 ... 8, 9 puis nous changeons de dizaine : 10, 11, 12, ... 18, 19 puis nous changeons de dizaine : 20, 21... jusqu'à 98, 99, puis nous changeons de centaine ($10 \times 10$) : 100, 101... En base 2, comme il y a moins de chiffres, cela va plus vite : 0, 1 puis on change de « deuzaine » 10 (lire « un, zéro »), 11 (lire « un, un ») et on change de « quatraine » ($2 \times 2$) : 100, 101 etc.

Voici donc les nombres de 0 à 15 écrits en binaire :

| décimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| binaire | 0 | 1 | 10 | 11 | 100 | 101 | 110 | 111 |

| décimal | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|
| binaire | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

## 2 Convertir

Pour connaître le nombre représenté par son écriture en base 2, on associe une valeur à chaque position binaire (qu'on appelle aussi *bit*, contraction de l'anglais *binary digit*, chiffre binaire). Au bit de poids faible (c'est le plus à droite, l'équivalent des unités) on associe la valeur 1. Au bit situé à sa gauche, on associe la valeur 2, puis au suivant la valeur 4, puis 8, 16, etc., jusqu'au bit de poids fort (le plus à gauche).

Si on ajoute entre elles les valeurs associées aux bits qui sont à 1, on obtient la valeur représentée par le nombre écrit en base 2. Pour un nombre de 4 bits, les valeurs associées aux positions binaires sont : 8, 4, 2 et 1. Si le nombre à convertir vaut 1101, on ne retient donc dans la somme que les valeurs 8, 4 et 1 (puisque le bit de valeur 2 est à 0), ce qui donne un total de 13. 1101 est donc l'écriture en base 2 de 13.

# Index


**Symboles**

`#` ..... 13, 42
`&` ..... 129
`*` ..... 10
`**` ..... 10
`+` ..... 10
`-` ..... 10
`(..)` ..... 14, 66
`(...)` ..... 40
`/` ..... 10
`:` ..... 22
`<<` ..... 21, 72
`==` ..... 14, 31
`>>` ..... 128
`[ ]` ..... 21, 55, 66, 72, 100
`%` ..... 10, 23

**A**

`abs` ..... 113
affixe ..... 109
Al-Kindi, Abu Yusuf Yaqub ..... 81
aliquote ..... 39
analyse fréquentielle ..... 81
Apollonius de Perge (baderne d') ..... 107
applet ..... 7
apt-get ..... 149
`Array` ..... 21
arrondi ..... 10, 85
auto-similarité ..... 107

**B**

baderne ..... 107
base ..... 165
`bash` ..... 150
bigramme ..... 81
binaire ..... 165
bit ..... 126, 165
    décalage de ..... voir décalage
bloc ..... 34
booléen ..... 25, 45
boucle ..... 14

**C**

calculs arithmétiques ..... 10
Catalan, Eugène Charles ..... 39
`cd` ..... 146, 147, 150
`chmod` ..... 150
`class` ..... 70
classe ..... 70
comparaison ..... 14
`Complex` ..... 109
complexes ..... voir nombres complexes
composante ..... 100, 131, 133
conditions ..... 13
constante ..... 117
cryptogramme ..... 66
cryptographie ..... 65, 125

**D**

débuggage ..... 26
décalage ..... 128
décimal ..... 165
`def` ..... 26
`delete_at` ..... 22
déverminage ..... 26
`dir` ..... 146

**E**

`each` ..... 34, 41
`each_char` ..... 83
`each_index` ..... 34
`each_key` ..... 85
`each_value` ..... 85
`else` ..... 14
Eratosthène ..... 45
et ..... 129, 166
Euler, Leonhard ..... 60
`Event.poll` ..... 120
expression rationnelle ..... 69

**F**

`false` ..... 25, 45
`File.read` ..... 74, 82
`fillRect` ..... 101

`find` ..... 35
`first` ..... 36
`flip` ..... 100
flottant ..... voir virgule flottante
fonction ..... 26
`for` ..... 14
fractale ..... 107

**G**

`gsub` ..... 69
Gutenberg ..... voir projet Gutenberg

**H**

hachage ..... 83
`has_key?` ..... 84
`Hash` ..... 83

**I**

`if` ..... 13
`in` ..... 14
`index` ..... 35
intervalle ..... 15, 40, 66
itérateur ..... 34

**J**

JRuby ..... 7
Julia, Gaston ..... 107

**L**

`last` ..... 55
`length` ..... 55
`ls` ..... 147, 150
LSB ..... 125

**M**

`man` ..... 150
Mandelbrot, Benoît ..... 107
masque ..... 128
Mersenne, Marin ..... 54
méthode ..... 26
modifieur ..... 31, 40
module ..... 45, 113

**N**

`new` ..... 21
`nil` ..... 13, 21
nombres ..... 10
  amicaux ..... 42
  complexes ..... 108
  de Mersenne ..... 42, 54
  parfaits ..... 41
  premiers ..... 43
  sociables ..... 40
non ..... 166

**O**

octet ..... 126, 165
`open` ..... 100
ou ..... 166

**P**

`p` ..... 25, 53
Paganini, Niccolo ..... 60
pixel ..... 100
premier(s) ..... voir nombres premiers
`print` ..... 34
procédure ..... 26
profondeur ..... 100
projet Gutenberg ..... 82
prompt ..... 7
`puts` ..... 13

**R**

`rand` ..... 76, 101
`read` ..... voir `File.read`
`reject` ..... 35
repère ..... 101
`return` ..... 26, 31
`reverse` ..... 85
`rindex` ..... 35, 51
routine ..... 26
`rsdl` ..... 99

**S**

SDL ..... 99, 156
`select` ..... 35
Sierpinski, Waclaw (triangle de) ..... 107
`size` ..... 22
`sort_by` ..... 85
stéganographie ..... 125
`su` ..... 149
substitution ..... 66
`sudo` ..... 149, 159
surface ..... 133
synthèse additive ou soustractive ..... 100

**T**

tableau ..... 21, 46
terminal ..... 150
`then` ..... 13
`times` ..... 50, 103

`to_a` ..... 85
transposition ..... 66
trigramme ..... 81
`true` ..... 25, 35, 45

**U**

Ulam, Stanislaw ..... 99
`upcase` ..... 69

**V**

variable ..... 11
Vigénère, Blaise de ..... 125
virgule flottante ..... 10, 85
von Koch, Helge (flocon de) ..... 107

**W**

`while` ..... 15

**X**

xor ..... 166