Ismael Belghiti, Roger Mansuy
& Jill-Jênn Vie

Les clefs pour l'Info

Calvage & Mounet

Ismael Belghiti, diplômé du département d'informatique de l'ENS Paris, est préparateur de l'équipe de France Olympique pour les Olympiades Internationales d'Informatique (au sein de l'association France-ioi – Algoréa).

Roger Mansuy est professeur de mathématiques et d'informatique au lycée Louis-le-Grand et fait partie du comité de rédaction de la Revue de Mathématiques Spéciales (RMS). Il est co-auteur avec Marc Yor de livres de probabilités avancées et, dans cette collection, de recueils d'exercices.

Jill-Jênn Vie, ancien élève de l'ENS Cachan, effectue une thèse d'informatique à l'université Paris-Saclay. Il a présidé en 2012 et 2013 l'association Prologin qui organise le concours national d'informatique du même nom, et est co-auteur avec Christoph Dürr d'un recueil de résolution de problèmes algorithmiques.

Les deux dessins illustrant la couverture correspondent au graphe de Nauru. Ce graphe admet vingt-quatre sommets et trente-six arêtes et possède plusieurs propriétés intéressantes : il est 3-régulier (tous les sommets sont de degré 3), biparti (ce qui est illustré par la coloration des sommets sur les deux dessins), et à distance-unité (dans le dessin en quatrième de couverture, toutes les arêtes sont de même longueur).

ISBN 978-2-916352-65-7

♾ Imprimé sur papier permanent

Préface

Vous tenez entre les mains un livre d'exercices d'informatique unique en son genre ; il a été conçu pour assurer une préparation à l'épreuve orale d'informatique théorique des Écoles normales supérieures. Cette ambition de soutenir tous les étudiants motivés pour une épreuve à la fois exigeante et atypique a entraîné quelques contraintes fortes qui font les spécificités de cet ouvrage.

Les **énoncés** correspondent au format très particulier de cette épreuve : ils sont longs, exigeants et requièrent d'importantes capacités de réflexion et d'abstraction sur des concepts à la limite du programme des classes préparatoires (ou au-delà de ce programme mais accessibles à partir de celui-ci). Pour obtenir un panorama assez complet de cette épreuve, les auteurs ont utilisé différents types d'énoncés :

- les énoncés rendus publics par le concours ; que les examinateurs soient remerciés de cet effort qui a permis à certains candidats isolés de connaître les modalités de l'épreuve. On peut regretter que ce mouvement d'ouverture n'ait pas lieu tous les ans.
- les énoncés reconstitués à partir des souvenirs (plus ou moins vaillants) des anciens candidats. Nous avons choisi de coller au maximum aux souvenirs des étudiants voire de confronter les souvenirs de plusieurs étudiants afin de construire des sujets proches de la réalité.
- les énoncés entièrement composés par les auteurs pour ajouter à cet ouvrage des thématiques qui ne figuraient pas dans les deux catégories précédentes mais semblaient essentielles pour une préparation optimale.

Les **corrigés** sont détaillés, commentés et complétés avec une courte bibliographie. Rédiger des réponses précises à certaines questions a imposé de formaliser des arguments parfois très intuitifs et convaincants. Il paraît naturel que lors d'une épreuve orale, la connivence avec l'examinateur et l'assurance du candidat puissent permettre d'éluder la sophistication un tantinet rigide qu'une publication requiert. Les commentaires servent selon les occasions à indiquer une idée générale de résolution, illustrer un

argument technique ou proposer un développement. Si ce livre a été pensé pour des candidats aux ENS, il sera a priori très profitable aux agrégatifs de mathématiques choisissant l'option Informatique. Les commentaires deviennent alors une lecture indispensable et une bonne préparation serait de considérer ces exercices comme une introduction vers d'autres lectures.

⋆ ⋆ ⋆ ⋆ ⋆

L'ouvrage s'articule entre différentes thématiques loin d'être disjointes. Voici le plan d'ensemble de l'ouvrage.

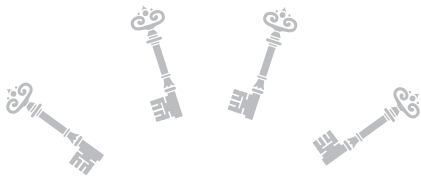
- Un premier chapitre d'algorithmique permet de reprendre les bases de l'analyse des programmes en particulier de l'analyse en complexité temporelle. Un point fort de ce chapitre repose sur les stratégies adversaires pour montrer qu'une borne obtenue pour la complexité dans le pire des cas est atteinte.
- Le chapitre sur les graphes (et le cas particulier des arbres) permet de revisiter différents aspects de ce vaste sujet : on retrouve certes de l'algorithmique mais aussi de la combinatoire et des raisonnements par induction. Les derniers exercices concernent les graphes infinis.
- Le troisième chapitre rassemble des exercices autour de la réécriture et la terminaison de sorte à mettre en avant les différentes notions de confluence. Les jeux à la Sprague-Grundy complètent ce panorama.
- La combinatoire revient en force avec l'étude des mots, autrement dit l'étude de certaines parties du monoïde Σ^* où Σ est un alphabet fini.
- Prenant un point de vue plus global, le chapitre 5 passe des mots aux langages et aux inévitables questions de rationalité qui s'y rapportent. Les exercices sur les automates se permettent alors d'aborder à nouveau l'algorithmique des graphes.
- Le dernier chapitre reprend les thématiques de logique propositionnelle et de calculabilité accessible avec le modeste programme de CPGE.

⋆ ⋆ ⋆ ⋆ ⋆

L'initiateur de ce livre est indéniablement Ismael Belghiti ; sans son enthousiasme et sa ténacité (depuis sa propre intégration), ce livre n'aurait jamais pu voir le jour. Nous avons longuement réfléchi puis travaillé sur ce projet. Jill-Jênn Vie a rejoint l'équipe afin de multiplier les regards et les intentions. De nombreuses personnes se sont investies dans le pénible travail de relecture : leurs remarques et conseils ont été précieux ; nous remercions particulièrement Alice Contat, Marion Scoazec, Yixin Shen, Jonathan Dong, Hugo Manet, Raphaël Monat. Les auteurs assument toutefois la totalité des erreurs ou coquilles ayant échappé à leur vigilance.

Roger Mansuy
Paris, le 11 novembre 2016

Table des matières



Chapitre 1

Algorithmique

Pour les questions d'algorithmique des exercices qui suivent, nous avons séparé l'expression en français ou en pseudo-code et l'analyse (terminaison, correction, complexité).

▷ Les conventions pour le pseudo-code sont assez classiques. Voici par exemple le pseudo-code pour la recherche du maximum dans un tableau de longueur n :

```
MaxTableau(t)
  iMax ← 0
  pour i allant de 1 à n − 1 faire
    si t[i] > t[iMax] alors
      iMax ← i
  Renvoyer(t[iMax])
```

▷ Afin d'estimer les complexités, nous utilisons les notations O de domination et Ω au sens de Landau. Rappelons que $u_n = O(v_n)$ si

$$\exists M > 0,\ \exists N \in \mathbb{N},\ \forall n \geqslant N, \qquad |u_n| \leqslant M|v_n|$$

et $u_n = \Omega(v_n)$ si $v_n = O(u_n)$.

Exercice 1

Rhô de Pollard

Soit E un ensemble fini, $f : E \to E$ une fonction et $u \in E^{\mathbb{N}}$ une suite telle que $u_{n+1} = f(u_n)$ pour tout $n \in \mathbb{N}$.

a. Montrer qu'il existe un entier $i \leqslant |E|$ tel que $(u_n)_{n \geqslant i}$ est périodique.

Soit r le plus petit entier vérifiant cette propriété et T la plus petite période de $(u_n)_{n\geqslant r}$.

b. Montrer que, si deux entiers $i < j$ vérifient $u_i = u_j$, alors $i \geqslant r$ et $j - i$ est multiple de T.

c. Soit v la suite définie, pour tout n, par $v_n = u_{2n}$. Caractériser l'ensemble $A = \{n \in \mathbb{N},\ u_n = v_n\}$.

d. En déduire un algorithme pour calculer T de complexité temporelle $O(r + T)$ et de complexité spatiale $O(\log(r+T))$. On suppose qu'une application de f s'effectue en temps et espace constant.

e. Notons i le plus petit élément non nul de A. En comparant la suite u avec la suite $w = (u_{i+n})_n$, montrer qu'on peut également déterminer l'entier r avec la même complexité que celle obtenue pour déterminer T.

f. Soit $h : \{1, \dots, n+1\} \to \{1, \dots, n\}$. Déterminer un algorithme, de complexité temporelle $O(n)$ et de complexité spatiale $O(\log n)$, qui renvoie deux entiers distincts a, $b \in \{1, \dots, n+1\}$ tels que $h(a) = h(b)$.

Solution

a. D'après le principe des tiroirs, deux des $|E| + 1$ éléments $u_0, \dots, u_{|E|}$ sont égaux car ils appartiennent à un ensemble de taille $|E|$. Il existe donc $0 \leqslant i < j \leqslant |E|$ tels que $u_i = u_j$. En posant $t = j - i$, on obtient donc que $f^t(u_i) = u_i$. D'après ce qui précède, pour tout $i' \geqslant i$:

$$u_{i'+t} = f^{i'-i}(f^t(f^i(u_0))) = f^{i'-i}(f^t(u_i)) = f^{i'-i}(u_i) = u_{i'}.$$

La suite $(u_n)_{n\geqslant i}$ est donc périodique de période t.

b. Soit $i < j$ deux entiers tels que $u_i = u_j$. D'après le raisonnement précédent, la suite $(u_n)_{n\geqslant i}$ est périodique de période $t = j - i$. Par minimalité de r, $i \geqslant r$. Soit $k \in \mathbb{N}$ tel que $r + kT \geqslant i$. En utilisant la T-périodicité de $(u_n)_{n\geqslant r}$ puis la t-périodicité de $(u_n)_{n\geqslant i}$ et à nouveau la T-périodicité de $(u_n)_{n\geqslant r}$,

$$\forall n \geqslant r, \qquad u_{n+t} = u_{n+kT+t} = u_{n+kT} = u_n.$$

Ainsi, $j - i$ est une période de $(u_n)_{n\geqslant r}$ donc un multiple de T.

c. Remarquons tout d'abord que $0 \in A$. Soit $n > 0$ tel que $v_n = u_n$, c'est-à-dire que $u_{2n} = u_n$. Puisque $n < 2n$, d'après la question précédente, $n \geqslant r$ et $2n - n = n$ est multiple de T. Réciproquement, si n est un multiple de T supérieur ou égal à r, alors $(u_p)_{p\geqslant r}$ admet n pour période, d'où $u_{2n} = u_n$. En conclusion, A est la réunion du singleton $\{0\}$ et de l'ensemble des multiples de T supérieurs ou égaux à r.

d. ▷ Soit i le plus petit élément non nul de A. Vu la question précédente, i est le plus petit multiple non nul de T supérieur ou égal à r ; ainsi, $i \leqslant r+T$. On peut donc déterminer i avec la complexité demandée en calculant en parallèle les termes successifs de u et de v par application de f, jusqu'à trouver le premier rang non nul où ces suites coïncident.

▷ Soit j le plus petit entier strictement supérieur à i vérifiant $u_i = u_j$. Puisque $i \geqslant r$, $u_i = u_{i+T}$ d'où $j \leqslant i+T$ par minimalité de j. Par ailleurs, d'après **b.**, $j-i$ est multiple de T, d'où $j-i = T$. On peut donc obtenir T comme le plus petit nombre (non nul) de fois qu'il faut appliquer f à u_i pour retomber sur u_i, ce qui donne bien une complexité temporelle linéaire en T. Pour faire ce calcul, on n'utilise qu'un nombre constant de variables comprises entre 1 et $r+T$, donc $O(\log(r+T))$ bits de mémoire.

Ces calculs fournissent un algorithme qui permet de déterminer T avec une complexité temporelle en $O(r+T)$ et, de façon remarquable, une complexité spatiale en $O(\log(r+T))$.

> *Cette méthode est appelée « algorithme du lièvre et de la tortue » ; la suite v représentant le lièvre et la suite u représentant la tortue.*

e. ▷ Soit $n \in \mathbb{N}$. Si $u_n = w_n$, alors $u_n = u_{n+i}$ d'où $n \geqslant r$ d'après la question **b.**. Réciproquement, pour $n \geqslant r$, $u_n = u_{n+i}$ car $(u_n)_{n\geqslant r}$ est T-périodique et i est multiple de T d'après **c.**. Les termes des suites u et w sont donc distincts jusqu'au rang r puis égaux au-delà.

▷ Dans la question précédente, on a exhibé un algorithme pour calculer $w_0 = u_i$ avec la complexité attendue. Pour obtenir r, il suffit donc de calculer en parallèle les termes successifs des suites u et w jusqu'à trouver un rang où elles coïncident, ce qui prend une complexité temporelle $O(r)$ et une complexité spatiale $O(\log(r+T))$.

f. Pour se ramener à la situation précédente, on pose $f = h$, on considère la suite u correspondant à $u_0 = n+1$ puis on définit r et T comme ci-dessus.

La suite $(u_n)_{n\geqslant 0}$ n'est pas périodique car $u_0 = n+1$ n'est pas dans l'image de f et n'apparaît donc qu'une fois dans u, d'où $r \geqslant 1$. En posant $a = u_{r-1}$ et $b = u_{r+T-1}$, on obtient $h(a) = h(u_{r-1}) = u_r$, $h(b) = h(u_{r+T-1}) = u_{r+T} = u_r$ car $(u_n)_{n\geqslant r}$ est T-périodique. Ainsi, $h(a) = h(b)$.

Supposons par l'absurde que $a = b$. En posant $i = r-1$ et $j = r+T-1$, on obtient $i < j$ et $u_i = a = b = u_j$ d'où, d'après **b.**, $i \geqslant r$, ce qui est absurde. Ainsi, $a \neq b$.

Enfin, comme $T = O(n)$, les algorithmes des questions **d.** et **e.** permettent de calculer r et T (et donc a et b) avec une complexité temporelle $O(n)$ et une complexité spatiale $O(\log n)$.

Exercice 2

Bornes optimales

a. Écrire un algorithme qui prend en entrée un tableau de $n \geqslant 2$ nombres distincts de taille n et qui renvoie l'indice de son maximum avec au plus $n-1$ comparaisons.

b. Soit un algorithme résolvant le problème précédent. Montrer qu'il existe un tableau de taille n pour lequel cet algorithme requiert au moins $n-1$ comparaisons.

c. Écrire maintenant un algorithme qui permet de déterminer l'indice du maximum et l'indice du minimum, en utilisant au plus $\lceil 3n/2 \rceil - 2$ comparaisons.

d. Montrer de même que, pour tout algorithme résolvant ce problème, il existe un tableau de taille n pour lequel cet algorithme requiert au moins $\lceil 3n/2 \rceil - 2$ comparaisons.

e. On considère maintenant le problème consistant à trouver les deux plus grands éléments du tableau de taille n fourni en entrée. Exhiber un algorithme utilisant au plus $n + \lceil \log_2 n \rceil - 2$ comparaisons.

f. Montrer que cette borne est atteinte.

Solution

L'exercice détaille des bornes explicites (et pas seulement asymptotiques) pour des problèmes algorithmiques classiques. Il illustre également des techniques pour trouver des instances (des valeurs des arguments) réalisant ces bornes.

a. Pour trouver l'indice du maximum, on parcourt le tableau en comparant à chaque étape l'élément courant avec le plus grand élément rencontré jusqu'alors.

```
IndiceMax(t)
  iMax ← 0
  pour i allant de 1 à n − 1 faire
    si t[i] > t[iMax] alors
      iMax ← i
  Renvoyer(iMax)
```

Cet algorithme utilise une unique comparaison par itération, soit $n-1$ comparaisons au total.

b. Soit t un tableau de longueur n et i_{min} et i_{max} les indices respectifs du minimum et du maximum de ce tableau. Considérons le graphe non-orienté G sur l'ensemble de sommets $S = \{0, \ldots, n-1\}$ dont les arêtes sont exactement les paires d'indices $\{i, j\}$ correspondant aux comparaisons faites lors de l'exécution de l'algorithme avec l'argument t. Si l'algorithme effectue strictement moins de $n-1$ comparaisons, le graphe G n'est pas connexe ; notons C l'ensemble (non vide) des sommets qui ne sont pas dans la composante connexe de i_{max}. Construisons alors le tableau t' obtenu à partir de t en augmentant de $t[i_{max}] - t[i_{min}] + 1$ les valeurs des éléments dont l'indice est dans C.

▷ Les éléments de t' sont deux à deux distincts ; en effet, les valeurs de t' d'indice dans C sont obtenues par translation de celles de t et t' coincide avec t sur $\bar{C}$. Par conséquent, les valeurs (distinctes) de t' sur C sont minorées par $t[i_{max}] + 1$ tandis que celles sur $\bar{C}$ sont majorées par $t[i_{max}]$. t' est donc une instance valide du problème.

▷ Par construction, les comparaisons qu'effectue l'algorithme avec l'argument t' sont identiques à celles qu'il effectue avec t : le résultat est donc l'indice i_{max} y compris pour l'argument t' : contradiction.

> *On a établi une propriété plus forte que celle de l'énoncé : l'algorithme utilise en effet au moins $n-1$ comparaisons pour tout argument.*

c. ▷ Une manière efficace de procéder consiste à considérer les éléments par paires. Pour chaque paire, on commence par comparer les deux éléments entre eux. Ensuite, on compare le plus grand des deux au maximum courant et le plus petit des deux au minimum courant.

▷ Dénombrons le nombre de comparaisons effectuées par cet algorithme.

- Si n est pair, alors la condition $i == n-1$ n'est jamais réalisée (car i croît de 2 en 2 en partant de 2, ne prenant ainsi que des valeurs paires), et l'algorithme fait donc une comparaison avant la boucle puis exactement 3 comparaisons par itération, soit un total de
$$\left\lceil \frac{3(n-2)}{2} \right\rceil + 1 = \left\lceil \frac{3n}{2} \right\rceil - 2.$$
- Si n est impair, l'algorithme effectue les mêmes comparaisons que pour le cas $n-1$ puis deux comparaisons pour la condition finale $i == n-1$, soit un total de
$$\left\lceil \frac{3(n-1)}{2} \right\rceil - 2 + 2 = \left\lceil \frac{3(n-1)}{2} \right\rceil = \left\lceil \frac{3n}{2} \right\rceil - 2.$$

```
IndiceMaxMin(t)
  si t[0] < t[1] alors
    iMin ← 0, iMax ← 1
  sinon
    iMin ← 0, iMax ← 1
  i ← 2
  tant que i < n faire
    si i == n − 1 alors
      si t[i] > t[iMax] alors
        iMax ← i
      sinon si t[i] < t[iMin] alors
        iMin ← i
    sinon
      si t[i] < t[i + 1] alors
        iMinPaquet ← i, iMaxPaquet ← i + 1
      sinon
        iMinPaquet ← i + 1, iMaxPaquet ← i
      si t[iMinPaquet] < t[iMin] alors
        iMin ← iMinPaquet
      sinon si t[iMaxPaquet] > t[iMax] alors
        iMax ← iMaxPaquet
      i ← i + 2
  Renvoyer(iMax, iMin)
```

d. Afin de démontrer le résultat demandé, nous allons exhiber une stratégie adversaire, c'est-à-dire un algorithme qui répond aux requêtes de comparaison de l'algorithme, construisant itérativement un tableau qui requiert l'utilisation d'au moins $\lceil 3n/2 \rceil - 2$ comparaisons.

> *Les valeurs des éléments du tableau ne sont pas essentielles, il suffit de construire l'ordre relatif des éléments pour pouvoir répondre aux différentes comparaisons. Pour construire un tableau, il suffit d'ordonner les différents éléments de celui-ci.*

Au début, l'adversaire marque tous les éléments à la fois d'une marque $+$ et d'une marque $-$.

Lorsque l'algorithme demande la comparaison de deux éléments x et y du tableau, l'adversaire répond de la façon suivante (en éliminant les cas symétriques) :

i. Si x et y ont tous deux à la fois les marques $+$ et $-$, alors l'adversaire répond de façon quelconque, celui jugé le plus petit perdant sa marque $+$, et celui jugé le plus grand perdant sa marque $-$.