

MÉTHODES ALÉATOIRES POUR LE PROBLÈME DE TRI ET RANG

Résumé : En informatique, il est courant de manipuler des listes de nombres. Leur classement en ordre croissant, la recherche du k -ième élément (le plus petit, avec $k \in \mathbb{N}^*$) figurent parmi les procédures les plus usuelles. Nous allons présenter dans ce texte des algorithmes aléatoires permettant d'effectuer ces opérations, puis nous aborderons l'étude de leurs performances.

Mots clefs : Algorithme aléatoire, bornes de Tchebychev, martingale à temps discret.

- *Il est rappelé que le jury n'exige pas une compréhension exhaustive du texte. Vous êtes laissé(e) libre d'organiser votre discussion comme vous l'entendez. Des suggestions de développement, largement indépendantes les unes des autres, vous sont proposées en fin de texte. Vous n'êtes pas tenu(e) de les suivre. Il vous est conseillé de mettre en lumière vos connaissances à partir du fil conducteur constitué par le texte. Le jury appréciera que la discussion soit accompagnée d'exemples traités sur ordinateur.*

1. L'algorithme RandQS

Supposons donné un ensemble S de n réels distincts, avec $n \in \mathbb{N}^*$ relativement grand, que l'on cherche à classer en ordre croissant. Si l'on savait trouver $y_0 \in S$ tel qu'en posant $S_1 = \{x \in S : x < y_0\}$ et $S_2 = \{x \in S : x > y_0\}$, $\text{card}(S_1)$ soit presque égal à $\text{card}(S_2)$ (au plus à une unité près), on procéderait de la manière suivante : par comparaison de tous les éléments de $S \setminus \{y_0\}$ avec y_0 , on construit S_1 et S_2 comme ci-dessus. Puis on itère ceci, en utilisant des éléments y_1 et y_2 respectivement presque "médián" de S_1 et de S_2 etc. Ainsi à chaque étape on dispose d'une partition "ordonnée" de S (après la première étape il s'agit de $S_1, \{y_0\}, S_2$, puis, avec des notations naturelles, de $S_{1,1}, \{y_1\}, S_{1,2}, \{y_0\}, S_{2,1}, \{y_2\}, S_{2,2} \dots$) et à l'étape suivante les sous-ensembles non réduits à un singleton voient leur taille divisée par 2 environ. Au bout d'un nombre d'étapes majoré par $\ln(n)/\ln(2)$, la partition ordonnée ne contient que des singletons, ce qui forme le classement de S voulu. À chaque étape le nombre de comparaisons effectuées est au plus n , de sorte que leur nombre total est en $O(n \ln(n))$. Dans ce texte, on mesurera le temps de calcul (auquel on se référera aussi sous le terme de complexité) d'un algorithme par le nombre de comparaisons qu'il effectue, car pour les algorithmes de classement raisonnables, l'ensemble des opérations de comparaisons représente généralement la partie plus coûteuse en termes de temps de calcul. Le problème de la procédure précédente est que l'on ne sait pas a priori choisir un élément presque médian d'un ensemble. Notons que l'on garderait une complexité en $O(n \ln(n))$ si l'on savait choisir un élément $y_0 \in S$ tel qu'avec les notations précédentes, $\text{card}(S_1)/\text{card}(S) \in [\rho, 1 - \rho]$ avec $0 < \rho \leq 1/2$ fixé (c'est-à-dire ne dépendant pas de $\text{card}(S)$). Ceci n'est pas plus évident à effectuer, cependant si y_0

est choisi aléatoirement de manière uniforme sur S , il a de “bonnes chances” de couper S en S_1 et S_2 de tailles comparables. Cette remarque est à la base de l’algorithme **RandQS** (pour *Random Quicksort* en anglais) de type récursif (c’est-à-dire s’appelant lui-même) :

Entrée : un ensemble S de nombres réels distincts.

Sortie : les éléments de S rangés en ordre croissant.

1. Si S ne contient qu’un élément, renvoyer celui-ci.

2. Choisir un élément y_0 de manière uniforme sur S .

3. Comparer chaque élément de $S \setminus \{y_0\}$ avec y_0 pour construire $S_1 = \{x \in S : x < y_0\}$ et $S_2 = \{x \in S : x > y_0\}$.

4. Donner en sortie la version rangée de S_1 , obtenue en rappelant **RandQS**(S_1), suivie de y_0 , puis de la version rangée de S_2 , obtenue en rappelant **RandQS**(S_2).

Le nombre de comparaisons effectuées par cet algorithme est une variable aléatoire C et on s’intéresse à son espérance $\mathbb{E}[C]$.

Proposition 1. *En notant $n = \text{card}(S)$, on a $\mathbb{E}[C] \leq 2n \sum_{1 \leq k \leq n} 1/k$, cette dernière quantité étant équivalente à $2n \ln(n)$ pour n grand. Ainsi en moyenne, la complexité de **RandQS** est de l’ordre de celle des procédures hypothétiques envisagées précédemment.*

Preuve : ◀ Pour $1 \leq i \leq n$, notons $S^{(i)}$ l’élément de rang i de S ($S^{(1)}$ étant le plus petit). Pour $1 \leq i < j \leq n$, soit $C_{i,j}$ la variable de Bernoulli qui vaut 1 si $S^{(i)}$ et $S^{(j)}$ sont comparés au cours de l’algorithme et 0 sinon. En notant $p_{i,j}$ la probabilité $\mathbb{P}[C_{i,j} = 1]$, on a

$$\mathbb{E}[C] = \sum_{1 \leq i < j \leq n} \mathbb{E}[C_{i,j}] = \sum_{1 \leq i < j \leq n} p_{i,j}$$

Pour évaluer $p_{i,j}$, avec $1 \leq i < j \leq n$ donnés, associons à une réalisation de l’algorithme un arbre binaire (où chaque sommet admet au plus deux fils) : en reprenant les notations du début du texte, on place y_0 à la racine de l’arbre, puis son fils gauche (respectivement droit) est y_1 (resp. y_2), le fils gauche de y_1 est ensuite l’élément $y_{1,1}$ tiré dans $S_{1,1}$ etc. Lisons successivement en partant de la racine les lignes horizontales ainsi créées, de gauche à droite. On obtient une “permutation” $\pi = (y_0, y_1, y_2, y_{1,1}, y_{1,2}, y_{2,1}, y_{2,2}, \dots)$ de S . On vérifie ensuite que $C_{i,j} = 1$ si et seulement si parmi les éléments $S^{(i)}, S^{(i+1)}, \dots, S^{(j-1)}, S^{(j)}$, le premier qui apparaît dans π est $S^{(i)}$ ou $S^{(j)}$. En effet, par exemple si $S^{(i)}$ est le premier élément de $\{S^{(i)}, S^{(i+1)}, \dots, S^{(j-1)}, S^{(j)}\}$ apparaissant dans π , il va être comparé à $S^{(i+1)}, \dots, S^{(j-1)}, S^{(j)}$, car tous ces éléments se trouvaient avant le tirage de $S^{(i)}$ dans le même sous-ensemble que $S^{(i)}$ de la partition ordonnée construite jusque là. Ainsi on a dans ce cas $C_{i,j} = 1$. Par contre si le premier élément de $\{S^{(i)}, S^{(i+1)}, \dots, S^{(j-1)}, S^{(j)}\}$ que l’on rencontre dans π est $S^{(k)}$, avec $i < k < j$, alors par comparaison avec cet élément, $S^{(i)}$ et $S^{(j)}$ vont se retrouver dans des sous-ensembles distincts de la partition ordonnée construite à ce moment là et ne pourront jamais être comparés, d’où $C_{i,j} = 0$.

Il en découle que $p_{i,j} = 2/(j - i + 1)$. D’où en fin de compte

$$\mathbb{E}[C] = \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1} \leq 2 \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq n} \frac{2}{j}$$

►

2. Bornes de Chebyshev

Pour l'utilisateur, l'information fournie par l'espérance $\mathbb{E}[C]$ sur le temps de fonctionnement de l'algorithme **RandQS** n'est souvent pas suffisante : il voudrait savoir de plus avec quelle probabilité C est susceptible de beaucoup s'écarter de sa moyenne.

Un des moyens les plus simple pour évaluer ces déviations est d'avoir recours aux bornes de Chebyshev, que nous rappelons ci-dessous. Soit X un variable aléatoire réelle admettant un moment d'ordre 2. On note $\mu_X = \mathbb{E}[X]$ sa moyenne et $\sigma_X = \sqrt{\mathbb{E}[(X - \mu_X)^2]}$ son écart type. On dispose alors de la

Proposition 2. *Pour tout $a > 0$, on a*

$$\mathbb{P}[|X - \mu_X| \geq a\sigma_X] \leq \frac{1}{a^2}$$

Nous allons maintenant chercher à appliquer ces bornes à C . Commençons par préciser la proposition 1 en prouvant le

Lemme 3. *Pour $n \in \mathbb{N}^*$, notons C_n le nombre de comparaisons effectuées par l'algorithme **RandQS** sur une entrée formée d'un ensemble S de cardinal n . On a pour n grand,*

$$\mathbb{E}[C_n] \sim 2n \ln(n)$$

Preuve : ◀ On va procéder par récurrence sur n pour évaluer $\mathbb{E}[C_n]$. Introduisons les notations suivantes : K est le rang de l'élément y_0 choisi (ainsi par exemple si y_0 est le plus petit élément de S , $K = 1$), T_1 (respectivement T_2) est le nombre de comparaisons effectuées dans S_1 (resp. dans S_2). On a donc

$$C_n = n - 1 + T_1 + T_2$$

Cependant, conditionnellement à $K = k$, pour $k \in \{1, 2, \dots, n\}$, T_1 est indépendant de T_2 et T_1 (resp. T_2) a même loi que C_{k-1} (resp. C_{n-k}). De plus par hypothèse, K est uniformément distribué sur $\{1, 2, \dots, n\}$. On en déduit que

$$\begin{aligned} \mathbb{E}[T_1] &= \mathbb{E}[\mathbb{E}[T_1|K]] \\ &= \frac{1}{n} \sum_{1 \leq k \leq n} \mathbb{E}[T_1|K = k] \\ &= \frac{1}{n} \sum_{1 \leq k \leq n} \mathbb{E}[C_{k-1}] \end{aligned}$$

et cette quantité est aussi égale à $\mathbb{E}[T_2]$. En posant $e_n = \mathbb{E}[C_n]$, on a abouti de la sorte à la récurrence

$$e_n = n - 1 + \frac{2}{n} \sum_{1 \leq k \leq n-1} e_k$$

On en déduit que $(n+1)e_{n+1} - ne_n = 2n + 2e_n$, d'où

$$\frac{e_{n+1}}{n+2} = \frac{e_n}{n+1} + \frac{2n}{(n+1)(n+2)}$$

puis

$$(1) \quad \begin{aligned} e_n &= (n+1) \sum_{k=0}^{n-1} \frac{2k}{(k+1)(k+2)} \\ &= 2n \ln(n) + O(n) \end{aligned}$$

►

Intéressons-nous ensuite au moment d'ordre 2 de C_n .

Lemme 4. *On a pour $n \in \mathbb{N}^*$ grand*

$$\mathbb{E}[C_n^2] \sim (2n \ln(n))^2$$

Esquisse de preuve : ◀ On procède de la même manière que dans la preuve précédente, dont on reprend les notations. En particulier on a $C_n^2 = (n-1)^2 + 2(n-1)(T_1 + T_2) + T_1^2 + 2T_1T_2 + T_2^2$. Posons $f_n = \mathbb{E}[C_n^2]$, en conditionnant à nouveau par rapport à K , on obtient

$$\forall n \geq 1, \quad f_n = \frac{2}{n} \sum_{k=0}^{n-1} f_k + r_n$$

avec $r_n = (n-1)^2 + \frac{4(n-1)}{n} \sum_{k=0}^{n-1} e_k + \frac{2}{n} \sum_{k=0}^{n-1} e_k e_{n-1-k}$. Cette récurrence se traite comme précédemment, mais il faut montrer que pour n grand,

$$\sum_{k=2}^n \frac{kr_k - (k-1)r_{k-1}}{k(k-1)} \sim 4n(\ln(n))^2$$

ce qui s'obtient notamment en tenant compte de (1). ►

Il en découle que

Proposition 5. *Pour tout $\epsilon > 0$, on a*

$$\lim_{n \rightarrow \infty} \mathbb{P}[C_n > (1+\epsilon)2n \ln(n)] = 0$$

Ainsi pour n grand, C_n est relativement proche de sa moyenne avec une grande probabilité. Signalons qu'en utilisant des bornes de Chernoff pour les sommes de variables de Bernoulli indépendantes, il est possible d'obtenir des renseignements quantitatifs plus précis sur le comportement des algorithmes considérés.

3. L'algorithme Find

Considérons plutôt le problème consistant à trouver le k -ième élément de S avec $1 \leq k \leq n = \text{card}(S)$. Pour le résoudre, on introduit l'algorithme **Find** qui s'inspire de **RandQS** :

Entrée : un couple (S, k) formé d'un ensemble S de réels et d'un entier $1 \leq k \leq \text{card}(S)$.

Sortie : le k -ième élément de S .

1. Choisir un élément y_0 de manière uniforme sur S .
2. Comparer chaque élément de S avec y_0 pour construire $S_1 = \{x \in S : x < y_0\}$ (et ce faisant calculer $n_1 = \text{card}(S_1)$) et $S_2 = \{x \in S : x > y_0\}$.
3. Si $n_1 = k - 1$, le k -ième élément de S est y_0 . Si $n_1 < k - 1$ (respectivement $n_1 > k - 1$), itérer la procédure avec $(S_2, k - n_1 - 1)$ (resp. (S_1, k)).

Intéressons-nous au nombre aléatoire T d'itérations effectuées par cet algorithme à l'étape 4.

Proposition 6. On a $\mathbb{E}[T] \leq 4(1 + \ln(\text{card}(S)))$.

Pour démontrer cette borne, on va faire appel à un résultat général sur les chaînes de Markov $(Y_p)_{p \in \mathbb{N}}$ à valeurs dans \mathbb{N}^* , homogènes en temps et qui sont strictement décroissantes tant qu'elles n'ont pas atteint 1. Supposons que l'on connaisse une fonction $g : [1, +\infty[\rightarrow \mathbb{R}_+^*$ croissante telle que pour tout temps $p \geq 0$ et tout entier $m \geq 2$,

$$(2) \quad \mathbb{E}[Y_{p+1} | Y_p = m] \leq m - g(m)$$

Soit $\tau = \min\{p \geq 0 : Y_p = 1\}$ le temps d'atteinte de 1.

Lemme 7. Si $Y_0 = n \in \mathbb{N}^*$, on dispose de l'estimation

$$\mathbb{E}[\tau] \leq \int_1^n \frac{1}{g(t)} dt$$

Esquisse de preuve : ◀ Pour $p \in \mathbb{N}$, soit \mathcal{F}_p la tribu engendrée par Y_0, \dots, Y_p et posons

$$M_p = p + \int_1^{Y_p} \frac{1}{g(t)} dt$$

Le processus $(M_p)_{p \in \mathbb{N}}$ est une surmartingale positive par rapport à la filtration $(\mathcal{F}_p)_{p \in \mathbb{N}}$, au sens où

$$\forall p \in \mathbb{N}, \quad \mathbb{E}[M_{p+1} | \mathcal{F}_p] \leq M_p$$

En appliquant le théorème d'arrêt de Doob au temps d'arrêt borné τ , il ressort que $\mathbb{E}[M_\tau] \leq \mathbb{E}[M_0]$, ce qui donne le résultat voulu. ▶

On voudrait appliquer ce lemme avec la suite de variables aléatoires $(Y_p)_{p \in \mathbb{N}}$ donnant la taille des ensembles résiduels à traiter obtenus à l'étape 3 de l'algorithme **Find**. Plus précisément, $Y_0 = n = \text{card}(S)$, puis selon les trois alternatives de l'étape 3 de l'algorithme **Find**, on a $Y_1 = 1$ (pour l'ensemble résiduel $\{y_0\}$), $Y_1 = n - n_1 - 1$ (pour l'ensemble résiduel S_2) ou $Y_1 = n_1$ (pour l'ensemble résiduel S_1) etc. Malheureusement, la chaîne $(Y_p)_{p \in \mathbb{N}}$ n'est pas markovienne, car à tout instant $p \in \mathbb{N}$, pour obtenir Y_{p+1} , outre Y_p , on a besoin de connaître le rang de l'élément que l'on cherche, disons K_p (ainsi $K_0 = k$, puis $K_1 = 1$ ou $K_1 = k - n_1 - 1$ ou $K_1 = k$, suivant les trois alternatives possibles à l'étape 3 de l'algorithme **Find**). En introduisant les tribus $\mathcal{F}_p = \sigma(Y_q, K_q : 0 \leq q \leq p)$, on peut voir que le lemme précédent s'étend à cette situation, si l'on remplace (2) par l'hypothèse que $\mathbb{E}[Y_{p+1} | \mathcal{F}_p] \leq Y_p - g(Y_p)$ pour tout $p \geq 0$. Or

avec ces notations, on calcule que pour tous $m \geq 2$ et $1 \leq l \leq m$,

$$\begin{aligned} \mathbb{E}[Y_{p+1}|Y_p = m, K_p = l] &= \frac{1}{m} \left(-l^2 + (m+1)l + \frac{(m-1)(m-2)}{2} \right) \\ &\leq \frac{3m^2}{4} - \frac{m}{2} + \frac{3}{4} \end{aligned}$$

En conséquence, $\mathbb{E}[Y_{p+1}|\mathcal{F}_p] \leq Y_p - g(Y_p)$ avec $g(m) = \max(1, m-1)/4$ pour tout $m \geq 1$. et la proposition 2 s'en déduit. Plus généralement, s'il existe $0 < \rho < 1$ tel que $\forall p \geq 0, \forall m \geq 1, \mathbb{E}[Y_{p+1} - 1|Y_p = m] \leq \rho(m-1)$, on peut montrer que $\mathbb{E}[\sum_{0 \leq p \leq \tau} Y_p - 1] \leq (n-1)/(1-\rho)$. Ainsi la complexité de **Find** est majorée par $4n$.

Suggestions pour le développement

- ▶ *Soulignons qu'il s'agit d'un menu à la carte et que vous pouvez choisir d'étudier certains points, pas tous, pas nécessairement dans l'ordre, et de façon plus ou moins fouillée. Vous pouvez aussi vous poser d'autres questions que celles indiquées plus bas. Il est très vivement souhaité que vos investigations comportent une partie traitée sur ordinateur et, si possible, des représentations graphiques de vos résultats.*
- *Modélisation.* L'affirmation sur la part prépondérante des comparaisons dans le temps de calcul des algorithmes considérés vous paraît-elle justifiée? Quelles sont les valeurs minimales et maximales que peut prendre la variable C ? Pourquoi les problèmes de classement et de recherche d'un élément médian sont-ils en fait très liés?
- *Développements mathématiques.*
On pourra compléter les démonstrations des divers résultats énoncés dans le texte. En particulier a-t-on que T_1 est indépendant de T_2 dans la preuve du lemme 3? Quelles sont les hypothèses de validité du théorème d'arrêt de Doob invoqué dans le texte? Notamment le lemme 7 est-il encore valable si $(Y_p)_{p \in \mathbb{N}}$ est seulement décroissante (pas nécessairement strictement).
- *Etude numérique.* Certains des algorithmes présentés pourront être implémentés en ayant recours à des procédures récursives. On pourra alors essayer de vérifier expérimentalement les résultats énoncés sur leurs performances, à l'aide de variables externes convenablement définies (pour compter les comparaisons utilisées par l'algorithme **RandQS**, il est commode de définir une procédure effectuant cette opération tout en augmentant un compteur global). Comment feriez-vous pour illustrer numériquement la proposition 2 ou les lemmes 3 ou 7?