Rapport du TD 1 d'Algorithmique

Stéphane Zimmermann / Claire Guillet

On s'intéresse dans ces exercices à des ensembles de n entiers, tous distincts, rangés dans un tableau $T[1], \ldots, T[n]$. Les algorithmes considérés effectuent des affectations et leur seul critère de décision est la comparaison.

Exercice 1

1- **Un algorithme naïf** pour le calcul du maximum de n entiers :

---

début
  max $\leftarrow T[1]$
  pour i **de** *1* **à** n **faire**
    si $T[i] > max$ **alors** max $\leftarrow T[i]$
  fin
  Retourner max
fin

Complexité en nombre de comparaisons : quel que soit le cas, on effectue $n-1$ comparaisons dans l'algorithme, chaque élément après le premier étant comparé une fois.

- dans le pire des cas : n (si le tableau est ordonné de manière croissante),
- dans le meilleur des cas : 1 (si le maximum est en première position),
- en moyenne : pour le calcul de la complexité en moyenne, il va falloir faire appel à un calcul plus poussé :
  - **Rappel :** *complexité en moyenne d'un algorithme A de données de taille n :*

$$moy_A(n) = \sum_{\text{n données de taille m}} p(d)\text{coût}_A(d)$$

    où $p(d)$ est la probabilité que d soit une entrée de l'algorithme A.

- les données sont les permutations de [1, ... ,n] stockées dans T. Les permutations étant totalement aléatoires et n'étant régies par aucune loi mathématique, on peut les supposer équiprobables.
- le coût d'un algorithme A(T) est son nombre d'affectation. Ainsi, pour chaque cas de $P_{n,k}$, l'algorthime effectue k affectations. On obtient donc ainsi que le coût d'un algorithme A(T) est de $kP_{n,k}$.
- d'où :

$$moy_A(n) = \frac{1}{n!} \sum_{T\ permutations\ de\ [1,...,n]} cout_A(T) = \frac{1}{n!} \sum_{k=1}^{n} kP_{n,k}$$

- trouvons $P_{n,k}$:
  * supposons le max en $T[n]$: pour avoir k affectations en tout, il faut : $P_{n-1,k-1}$
  * sinon, les affectations sont réalisées avant et on a alors $n-1$ permutations possibles: $P_{n-1,k}$
  * Donc $P_{n,k} = P_{n-1,k-1} + (n-1)P_{n-1,k-1}$
- cas limites :
  * $P_{n,0} = 0$
  * si $k > n$: $P_{n,k} = 0$
  * $\forall n \in \mathbb{N}$: $P_{n,n} = 1$
- Utilisons les séries génératrices : $G_n(z) = \sum_{k=1}^{n} P_{n,k} z^k$

$$\begin{aligned}
G_{n+1}(z) &= \sum_{k=1}^{n+1} P_{n,k} z^k \\
&= \sum_{k=1}^{n+1} (P_{n,k-1} + nP_{n,k}) z^k \\
&= \sum_{k=1}^{n+1} P_{n,k-1} z^{k-1} + n \sum_{k=1}^{n+1} P_{n,k} z^k \\
&= \sum_{k=0}^{n} P_{n,k} z^k + n \sum_{k=1}^{n+1} P_{n,k} z^k
\end{aligned}$$

Or $P_{n,0} = 0$ et $P_{n,n+1} = 0$ d'où :

$$G_{n+1}(z) = z\sum_{k=0}^{n} P\_n, kz^k + n\sum_{k=1}^{n} P_{n,k}z^k$$

d'où :

$$G_{n+1}(z) = (z+n)G_n(z)$$

Or $G_1(z) = P_{1,1}z = z$ d'où $G_n(z) = z(z+1)\ldots(z+n-1)$

– de plus :

$$\begin{aligned}
G'_n(z) &= \sum_{k=1}^{n} kP_{n,k}z^{k-1} \\
\text{d'où : } G'_n(1) &= \sum_{k=1}^{n} kP_{n,k} \\
\text{de même : } G_n(1) &= \sum_{k=1}^{n} P_{n,k} = n!
\end{aligned}$$

– On obtient donc :

$$\begin{aligned}
moy_A(n) &= \frac{G'_n(1)}{G_n(1)} \\
&= [lnG(z)]'(1) \\
&= [\sum_{i=0}^{n-1} ln(z+i)]'(1) \\
&= 1 + \frac{1}{2} + \cdots + \frac{1}{n} \\
&= H_n \text{ (la suite harmonique à l'ordre } n\text{)} \\
\\
moy_A(n) &= O(ln\, n)
\end{aligned}$$

2-**Optimalité en terme de comparaisons**

Une comparaison est un match entre deux entiers, le gagnant étant le plus grand. Si le max est connu, c'est que tout les autres entiers ont perdu au moins une fois un match contre lui, ce qui demande en tout $n-1$ comparaisons. Notre algorithme en faisant toujours $n-1$, il est optimal pour la complexité en nombre de comparaisons.

Exercice 2 - Plus petit et plus grand

1- **Un algorithme naïf** pour le calcul du maximum et du minimum de n entiers :

---

début
  max $\leftarrow T[1]$
  $i_{max} \leftarrow 1$
  min $\leftarrow T[1]$
  pour i **de** *2* **à** n **faire**
    si $T[i] > max$ **alors** max $\leftarrow T[i]$; $i_{max} \leftarrow i$
  fin
  pour i **de** *2* **à** n **faire**
    si $i \neq i_{max}$ **et** $min < T[i]$ **alors** min $\leftarrow T[i]$
  fin
  Retourner (max,min)
fin

Complexité en nombre de comparaisons : On effectue $n-1$ comparaisons pour trouver le maximum, et $n-2$ pour trouver le minimum. On a donc une complexité en $2n-3$.

2- **groupement des éléments par paires**

Algorithme :

---

début
  pour i **de** *1* **à** $\lfloor \frac{n}{2} \rfloor$ **faire**
    si $T[2i-1] > T[2i]$ **alors** échange $T[2i-1]$ et $T[2i]$
  fin
  max $\leftarrow T[2]$
  pour i **de** *2* **à** $\lfloor \frac{n}{2} \rfloor$ **faire**
    si $T[2i] > max$ **alors** max $\leftarrow T[2i]$
  fin
  min $\leftarrow T[1]$
  pour i **de** *2* **à** $\lfloor \frac{n}{2} \rfloor$ **faire**
    si $T[2i-1] < min$ **alors** min $\leftarrow T[2i-1]$
  fin
  si n *impair* **alors** **si** $T[n] > max$ **alors** max $\leftarrow T[n]$
  sinon si $T[n] < min$ **alors** min $\leftarrow T[n]$
fin

Complexité associée :

- *si n est pair*, on va avoir $\frac{n}{2}$ comparaisons pour faire les paires, $\frac{n}{2} - 1$ comparaisons pour trouver le maximum et autant pour le mimimum. Dans ce cas là, la complexité vaut donc :

$$\frac{3n}{2} - 2$$

- *si n est impair*, le nombre de comparaisons est inférieur ou égal à $3\lfloor \frac{n}{2} \rfloor$, donc inférieur ou égal à $\lceil \frac{3n}{2} \rceil - 2$.

3- **méthode de l'***adversaire*

Cette méthode va être utilisée pour fournir une borne inférieure sur le nombre de comparaisons à effectuer et ainsi montrer l'optimalité de l'algorithme précédent.

Définitions :
- Novice : élément qui n'a jamais été comparé
- Gagnant : élément qui a gagné tout ses matchs
- Perdant : élément qui a perdu tout ses matchs
- Moyen : élément qui a gagné et perdu

Soit A notre algorithme. On note (i, j, k, l) respectivement le nombre de novices, gagnants, perdants et moyens (avec $i + j + k + l = n$).

Au début de l'algorithme, on a $(i, j, k, l) = (n, 0, 0, 0)$. A la fin, $(i, j, k, l) = (0, 1, 1, n - 1)$.

Le but de l'adversaire est de maximiser la durée de l'algorithme tout en conservant une stratégie cohérente. On peut alors donner les opérations suivantes :

comparaison	choix	i	j	k	l
N:N		$i-2$	$j+1$	$k+1$	/
N:G	$G > N$	$n-1$	/	$k+1$	/
N:P	$P < N$	$n-1$	$j+1$	/	/
N:M	$M > N$	$i-1$	/	$k+1$	/
	$N > M$ [1]	$i-1$	$j+1$	/	/
G:G		/	$j+1$	/	$l+1$
G:P	$G > P$	/	/	/	/
G:M	$G > M$	/	/	/	/
P:P		/	/	$k-1$	$l+1$
P:M	$P < M$	/	/	/	/
M:M		/	/	/	/

/ représentant le cas où l'entier reste inchangé.

Cherchons à donner une borne inférieure sur le nombre de tests à effectuer. Il faut que chaque novice soit comparé au moins une fois. La comparaison la plus efficace est alors N:N car elle diminue le nombre de novice de deux, et il en faut au moins $\lceil \frac{n}{2} \rceil$ pour atteindre le bon nombre de novices.

De plus, il faut à la fin que l'on ait $n-2$ moyens, ce qui nécessite au moins $n-2$ autres comparaisons.

D'où au minimum

$$n - 2 + \left\lceil \frac{n}{2} \right\rceil = \left\lceil \frac{3n}{2} \right\rceil - 2 \text{ comparaisons.}$$

La borne inférieure sur le nombre de tests à effectuer est égale à la complexité de l'algorithme précédent. On montre ainsi que ce dernier est optimal.

[1] ici, les deux cas sont équivalents.