Synthèse du cours 3 : la recherche du k-ème élément

28 septembre 2021

François Laroussinie

NB : Ces synthèses ont pour but de compléter les notes prises en cours. Elles ne les remplacent pas ! En particulier, la plupart des preuves n'y figurent pas. Rappel : il faut programmer les algorithmes vus en cours.

1 La recherche du k-ème élément

Input : un tableau $T[\ldots]$ de taille n contenant des valeurs distinctes, et un entier $1 \leq k \leq n$.
Output : la valeur v telle que le nombre de valeurs de T strictement inférieures à v est $k-1$ et le nombre de valeurs de T strictement supérieures à v est $n-k$.

1.1 Algorithme naïf

Le premier algorithme est basé sur la fonction `indiceMin` qui renvoie l'indice du plus petit élément d'un tableau.

Procédure `Algo1`(T,k)
begin
  pour chaque $i = 1 \ldots k$ **faire**
    $\mathsf{im} = \mathtt{indiceMin}(T)$;
    si $i < k$ **alors**
      $T[\mathsf{im}] := \infty$;
    fin
  fin
  return $T[\mathsf{im}]$;
end

Complexité en $O(k \cdot n)$.

1.2 Algorithme à base de tri

Le second algorithme est basé sur la fonction `trier` qui trie le tableau (dans l'ordre croissant).

Procédure `Algo2`(T,k)
begin
  `trier`(T) ;
  return $T[k]$;
end

Complexité en $O(n \cdot \log n)$.

1.3 Algorithme utilisant les tas

Le troisième algorithme utilise des tas [1] : `MiseEnTas` est transforme un tableau de valeurs en tas (complexité en $O(n)$) et `ExtraireMin` extrait (et renvoie) le min d'un tas (complexité en $O(\log n)$)

Procédure `Algo3`(T,k)
begin
  `MiseEnTas`(T) ;
  pour chaque $i = 1 \ldots k$ **faire**
    v = `ExtraireMin`(T) ;
  fin
  return v ;
end

Complexité en $O(n + k \cdot \log n)$.

1.4 Algorithme quickselect

Le quatrième algorithme (appelé *quickselect*) est basé sur *quicksort* : on pivote (*ie* on choisit un pivot p et on place toutes les valeurs inférieures à p au début du tableau et toutes celles supérieures à p à la fin, et p à sa place) et en fonction du rang du pivot, on continue la recherche dans la partie gauche, ou la partie droite ou on s'arrête si le pivot est le k-ème élément. La fonction `indicePivot` pivote le tableau et renvoie l'indice du pivot (placé à sa place).

Procédure `Algo4`(T, k, bg, bd)
begin
  ip := `indicePivot`(T, bg, bd) ;
  rgpivot := ip $- bg + 1$;
  si $k <$ rgpivot **alors**
    return `Algo4`($T, k, bg,$ ip $- 1$) ;
  else if $k >$ rgpivot **then**
    return `Algo4`($T, k -$ rgpivot, ip $+ 1, bd$) ;
  else
    return T[ip] ;
  end
end

La complexité dans le pire des cas est en $O(n^2)$ mais en **moyenne** elle est linaire (donc en $O(n)$).

La complexité en moyenne ; On note $\mathcal{C}(n, k)$ le coût moyen (en nombre de comparaisons) de la recherche du k-ème élément dans un tableau de taille n (ou plus exactement une zone de taille n) par le quickselect. On suppose que le *rang* du pivot est équiprobable : il peut prendre

1. Voir la note séparée.

les valeurs 1, 2,…,n et chacune a une probabilité de $\frac{1}{n}$. On peut écrire (si l'algorithme de pivotage vérifie certaines propriétés) :

$$\mathcal{C}(n,k) \;=\; \underbrace{\frac{1}{n}\Big(\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i) \;+\; \sum_{i=k+1}^{n}\mathcal{C}(i-1,k)\Big)}_{i \text{ correspond à la pos. du pivot.}} \;+\; \underbrace{n-1}_{\text{pivotage}}$$

La première somme correspond à la poursuite de la recherche dans la partie droite du tableau et la seconde à la recherche dans la partie gauche… La moyenne pour tous les k est alors : $\mathcal{C}(n) = \dfrac{1}{n}\displaystyle\sum_{k=1}^{n}\mathcal{C}(n,k)$.

D'où :

$$\mathcal{C}(n) \;=\; \frac{1}{n^2}\Big(\underbrace{\sum_{k=1}^{n}\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i)}_{A} \;+\; \underbrace{\sum_{k=1}^{n}\sum_{i=k+1}^{n}\mathcal{C}(i-1,k)}_{B}\Big) \;+\; n-1$$

Si on étudie les sommes A et B (à faire !), on observe que :

— A : $\displaystyle\sum_{k=1}^{n}\sum_{i=1}^{k-1}\mathcal{C}(n-i,k-i) = \sum_{i=1}^{n-1}\sum_{k=1}^{i}\mathcal{C}(i,k)$.

— B : $\displaystyle\sum_{k=1}^{n}\sum_{i=k+1}^{n}\mathcal{C}(i-1,k) \;=\; \sum_{k=1}^{n}\sum_{i=k}^{n-1}\mathcal{C}(i,k) \;=\; \sum_{i=1}^{n-1}\sum_{k=1}^{i}\mathcal{C}(i,k)$.

On en déduit :

$$\begin{aligned}\mathcal{C}(n) &= \frac{2}{n^2}\sum_{i=1}^{n-1}\Big(\sum_{k=1}^{i}\mathcal{C}(i,k)\Big) \quad + \quad n-1\\ &= \frac{2}{n^2}\sum_{i=1}^{n-1}\Big(i\cdot\mathcal{C}(i)\Big) \quad + \quad n-1\end{aligned} \tag{1}$$

On résout…

$$\begin{array}{rcc}
n^2\cdot\mathcal{C}(n) & = & 2\displaystyle\sum_{i=1}^{n-1} i\cdot\mathcal{C}(i) \quad + \; n^2(n-1)\\
(n-1)^2\cdot\mathcal{C}(n-1) & = & 2\displaystyle\sum_{i=1}^{n-2} i\cdot\mathcal{C}(i) \quad + \; (n-1)^2(n-2)\\
\hline
n^2\cdot\mathcal{C}(n)-(n-1)^2\cdot\mathcal{C}(n-1) & = & 2(n-1)\mathcal{C}(n-1)+n^2(n-1)-(n-1)^2(n-2)\\
n^2\cdot\mathcal{C}(n) & = & ((n-1)^2+2n-2)\mathcal{C}(n-1)+n^3-n^2+2n^2-4n+2-n^3+2n^2-n\\
n^2\cdot\mathcal{C}(n) & = & (n^2-2n+1+2n-2)\mathcal{C}(n-1)+3n^2-5n+2\\
n^2\cdot\mathcal{C}(n) & = & (n^2-1)\mathcal{C}(n-1)+3n^2-5n+2
\end{array}$$

D'où on déduit :

$$\frac{n^2\cdot\mathcal{C}(n)}{n(n+1)} \;=\; \frac{(n+1)(n-1)}{n(n+1)}\mathcal{C}(n-1) \;+\; \frac{3n^2-5n+2}{n(n+1)}$$

Et donc :

$$\frac{n}{n+1}\cdot\mathcal{C}(n) \;=\; \frac{n-1}{n}\mathcal{C}(n-1) \;+\; \frac{3n^2-5n+2}{n(n+1)}$$

On pose : $s_n = \frac{n}{n+1}\cdot\mathcal{C}(n)$, et on a : $s_n = s_{n-1} + \frac{3n^2-5n+2}{n(n+1)}$, d'où :

$$s_n \;=\; s_{n-1} + \frac{3n-5}{n+1} + \frac{2}{n} - \frac{2}{n+1}$$

Avec $s_1 = \frac{\mathcal{C}(1)}{2} = 0$. Mais on peut aussi prendre de manière équivalente (mais plus « régulière » pour des indices allant de 1 à n pour le calcul ci-dessous) la convention : $s_0 = 0$ (car il donne aussi $s_1 = 0$). Et finalement, pour $n \geq 1$, on a :

$$\begin{aligned} s_n &= \sum_{i=1}^{n} \underbrace{\frac{3i-5}{i+1}}_{3-\frac{8}{i+1}} + \frac{2}{i} - \frac{2}{i+1} \\ &= 3n - \underbrace{8\sum_{i=1}^{n}\frac{1}{i+1}}_{O(\ln n+1)} \quad + \quad \underbrace{2\sum_{i=1}^{n}(\frac{1}{i} - \frac{1}{i+1})}_{=2(1-\frac{1}{n+1})} \end{aligned} \tag{2}$$

Et on a donc : $\mathcal{C}(n) = O(n)$.

1.5 Algorithme optimal

Cet algorithme a été proposé par Blum, Floyd, Pratt, Rivest et Tarjan en 1973. Il procède comme suit :

— Si n est petit (par exemple ≤ 50), utiliser l'algo 4.

— Sinon :

1. diviser les n éléments en $\lfloor\frac{n}{5}\rfloor$ blocs B_i de 5 éléments ;
2. trouver le médian m_i de chaque B_i ;
3. trouver le médian M de $[m_1, \ldots, m_{\lfloor\frac{n}{5}\rfloor}]$;
4. pivoter avec M comme pivot ;
5. continuer dans la bonne partie du tableau (comme dans select)...

Il contient deux appels récursifs : le premier pour trouver le médian M des m_i, et le second sur un des sous-tableaux (à moins que le pivot M soit le k-ème élément recherché). Ce choix de pivot assure que le sous-tableau où se poursuit éventuellement la recherche est de taille au plus $\frac{7\cdot n}{10} + 2$, et cela permet de montrer qu ela complexité dans le pire cas est en $O(n)$.