8.5 Das Selektionsproblem

Gegeben ist eine Folge $(a_1, \ldots, a_n)$ von n Objekten aus einer totalen Ordnung $(D, <)$ (in Array oder als Liste),
sowie eine Zahl k, $1 \leq k \leq n$.

O.B.d.A.: Alle Einträge verschieden.

Aufgabe: Finde das Element der Folge, das **Rang** k hat, d. h. ein Objekt x in der Liste mit $|\{i \mid a_i \leq x\}| = k$.

Spezialfall: Der **Median** einer Folge mit n Einträgen ist das Element mit Rang $\lceil n/2 \rceil$.
(Median$(\{2, 4, 7, 9\}) = 4$, Median$(\{4, 7, 9\}) = 7$.)

Einfache Lösung: Sortiere, mit Ergebnis $(b_1, \ldots, b_n)$, dann wähle $x = b_k$. – Kosten: $n \log n$ Vergleiche, Zeit $O(n \log n)$.

[1] C. A. R. Hoare (*1934), brit. Informatiker,
erfand Quicksort, Quickselect & Korrektheitskalkül.

„I conclude that there are two ways of constructing a software design: One way is to make it so *simple* that there are *obviously no* deficiencies and the other way is to make it so *complicated* that there are *no obvious* deficiencies. The first method is far more difficult.“

(Dankesrede für den Turingpreis 1980)

„I think Quicksort is the only really interesting algorithm that I've ever developed.“

Quelle: Wikipedia

Zunächst:
Ein **randomisierter Algorithmus** für das Auswahlproblem.

Quickselect (Hoare)

Ansatz: Wie bei Quicksort.

Gegeben: Folge $(a_1, \ldots, a_n)$, Zahl k, $1 \leq k \leq n$.

O.B.d.A.: Die a_i sind verschieden.

Falls $n = 1$, ist nichts zu tun.

Falls $n = 2$, sortiere mit einem Vergleich, Ergebnis (b_1, b_2), gib Element b_k zurück.

Falls $n \geq 3$:

Wähle ein Element x aus $\{a_1, \ldots, a_n\}$ als partitionierendes Element **zufällig**.

Zerlege $(a_1, \ldots, a_n)$ mit $n-1$ Vergleichen in eine Teilfolge $b_1, \ldots, b_{p-1}$, alle $< x$, in das Element x, und eine Teilfolge $c_{p+1}, \ldots, c_n$, alle $> x$.

1. Fall: $k = p$. Das Ergebnis ist x.

2. Fall: $k < p$. Finde (rekursiv) in $(b_1, \ldots, b_{p-1})$ das Element vom Rang k.

3. Fall: $k > p$. Finde (rekursiv) in $(c_{p+1}, \ldots, c_n)$ das Element vom Rang $k - p$.

Prozedur quickSelect(a, b, k)

// Rekursive Prozedur im Quickselect-Algorithmus, $1 \leq a < b \leq n$, $a \leq k \leq b$.
// Vorbed.: Alle Einträge vom Rang $< a$ [$> b$] links [rechts] von `A`$[a..b]$
// Nachbed.: Eintrag vom Rang k in `A`$[k]$,
// kleinere links davon, größere rechts davon.

(1) `s` $\leftarrow$ ein zufälliges Element von $\{a, \ldots, b\}$;
(2) **if** $(a <$ `s`$)$ **then** vertausche `A`$[a]$ und `A[s]`;
(3) **partition**$(a, b,$ `p`$)$; // `p`: Ausgabeparameter
(4) **if** $k = p$ **then return**
(5) **elseif** $k <$ `p` **then quickSelect**$(a,$ `p` $- 1, k)$;
(6) **else quickSelect**$($`p` $+ 1, b, k)$.

Mögliche Anpassungen:
(a) Sortiere z.B. mit Einfügesortieren, wenn $b - a$ sehr klein ist.
(b) Anstelle von Rekursion benutze Iteration, analog zu der halbrekursiven Variante von Quicksort.

Korrektheit: Klar.

Zu analysieren: (Erwartete) Rechenzeit.

Klar: Die Rechenzeit ist proportional zur Anzahl $\boldsymbol{C_k}$ von Vergleichen.

Wir berechnen den Erwartungswert $\mathbf{E}(C_k)$.

Eingabezahlen, sortiert: $b_1 < \cdots < b_n$.

Beobachtung:
Es werden niemals zwei Zahlen mehrfach verglichen.
(Bei Vergleich ist eines das Pivotelement; es kommt im rekursiven Aufruf nicht mehr vor.)

Definiere

$$X_{ij} = \begin{cases} 1 & \text{, falls } b_i \text{ und } b_j \text{ verglichen werden} \\ 0 & \text{, sonst.} \end{cases}$$

Dann gilt:

$$C_k = \sum_{1 \le i < j \le n} X_{ij}.$$

Also (Linearität des Erwartungswertes):

$$\mathbf{E}(C_k) = \sum_{1 \le i < j \le n} \mathbf{E}(X_{ij}).$$

Weil die X_{ij} 0-1-wertig sind:

$$\mathbf{E}(X_{ij}) = \mathbf{Pr}(X_{ij} = 1) = \mathbf{Pr}(b_i \text{ und } b_j \text{ werden verglichen}).$$

Was ist $\mathbf{E}(X_{ij}) = \mathbf{Pr}(b_i$ und b_j werden verglichen)?

Wir stellen uns den Ablauf des Algorithmus mit Auswahl von Pivotelementen und rekursiven Aufrufen für Teilarrays vor.

Man beachte, dass die gebildeten Teilarrays stets einen Abschnitt $\{b_s, b_{s+1}, \ldots, b_t\}$ der sortierten Folge enthalten und dass jedes Element dieser Teilfolge dieselbe Wahrscheinlichkeit hat, als Pivot gewählt zu werden.

Setze $I_{i,j} = \{b_i, \ldots, b_j\}$.

1. Fall: $k \leq i < j$.

Solange kein Eintrag aus $I_{k,j} = \{b_k, b_{k+1}, \ldots, b_j\}$ als Pivot gewählt wird, passiert nichts bezüglich b_i, b_j.

Es kommt auf die Position p des ersten als Pivotelement gewählten Eintrags b_p in $I_{k,j}$ an.

Wenn $p < i$ ist, werden die Einträge in $I_{i,j} = \{b_i, \ldots, b_j\}$ im Weiteren ignoriert.

Wenn $p = i$ oder $p = j$ ist, werden b_i und b_j verglichen.

Wenn $i < p < j$ ist, kommt entweder b_i oder b_j in der Rekursion nicht mehr vor.

Also:

$$\mathbf{Pr}(X_{ij} = 1) = \frac{2}{j - k + 1}.$$

2. Fall: $i < k < j$:

Es kommt darauf an, ob b_i oder b_j vor allen anderen Einträgen in $\{b_i, b_{i+1}, \ldots, b_j\}$ Pivot wird. Also:

$$\mathbf{Pr}(X_{ij} = 1) = \frac{2}{j - i + 1}.$$

3. Fall: $i < j \leq k$:

Es kommt darauf an, ob b_i oder b_j vor allen anderen Einträgen in $\{b_i, b_{i+1}, \ldots, b_k\}$ Pivot wird. Also:

$$\mathbf{Pr}(X_{ij} = 1) = \frac{2}{k - i + 1}.$$

Also:

$$\mathbf{E}(C_k) = 2 \cdot \left(\sum_{k \leq i < j \leq n} \frac{1}{j-k+1} + \sum_{1 \leq i < k < j \leq n} \frac{1}{j-i+1} + \sum_{1 \leq i < j \leq k} \frac{1}{k-i+1} \right).$$

Erste Summe:

$$\sum_{k \leq i < j \leq n} \frac{1}{j-k+1} = \sum_{j=k+1}^{n} \frac{j-k}{j-k+1} < \sum_{j=k+1}^{n} 1 = n-k.$$

Dritte Summe:

$$\sum_{1 \leq i < j \leq k} \frac{1}{k-i+1} = \sum_{i=1}^{k-1} \frac{k-i}{k-i+1} < \sum_{i=1}^{k-1} 1 = k-1.$$

$\Rightarrow$ Beitrag dieser beiden Summen zu $\mathbf{E}(C_k)$ ist höchstens $2(n-1)$.

Die Terme der mittleren Summe

$$S = \sum_{1 \le i < k < j \le n} \frac{1}{j - i + 1}$$

stellen wir in der nachfolgenden $(k-1) \times (n-k)$-Matrix dar (für $k \le n/2$):

$$\begin{pmatrix}
\frac{1}{k+1} & \frac{1}{k+2} & \cdots & \cdots & \frac{1}{n-k+1} & \frac{1}{n-k+2} & \frac{1}{n-k+3} & \cdots & \frac{1}{n-1} & \frac{1}{n} \\
\frac{1}{k} & \frac{1}{k+1} & \frac{1}{k+2} & \cdots & \cdots & \frac{1}{n-k+1} & \frac{1}{n-k+2} & \cdots & \frac{1}{n-2} & \frac{1}{n-1} \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\
\frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{k+1} & \frac{1}{k+2} & \cdots & \cdots & \frac{1}{n-k+1} & \frac{1}{n-k+2} & \frac{1}{n-k+3} \\
\frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{k} & \frac{1}{k+1} & \frac{1}{k+2} & \cdots & \cdots & \frac{1}{n-k+1} & \frac{1}{n-k+2}
\end{pmatrix}$$

Wir betrachten die Diagonalen der Matrix. Auf jeder Diagonalen sind die Einträge konstant, und die Summe der Einträge auf jeder Diagonalen ist kleiner als 1. Es gibt genau $n-2$ Diagonalen, also gilt:

$$\sum_{1 \le i < k < j \le n} \frac{1}{j-i+1} < n-2.$$

Im Falle $k > n/2$ funktioniert das Argument genauso; die Matrix sieht nur etwas anders aus. Man kann jedoch auch o. B. d. A. $k \le n/2$ annehmen, da aus Symmetriegründen $C_k = C_{n-k+1}$ gilt.

Zusammen: $\mathbf{E}(C_k) \le 4n$.

Satz 8.5.1

Algorithmus **Quickselect** löst das Auswahlproblem und hat eine erwartete Vergleichsanzahl von $\leq 4n$ und eine erwartete Laufzeit von $O(n)$.

Mitteilungen:

(a) Eine genauere Analyse ergibt für $\alpha = k/n$ konstant eine erwartete Vergleichsanzahl von $2(1 + H(\alpha)\ln 2 + o(1))n < (3.3863 + o(1)) \cdot n$.

Dabei ist $H(\alpha) = -\alpha \log \alpha - (1-\alpha)\log(1-\alpha)$
die **„binäre Entropie“** der Wahrscheinlichkeitsverteilung $(\alpha, 1-\alpha)$.

$H(\alpha)$ liegt zwischen 0 und 1; das Maximum 1 ist bei $\alpha = \frac{1}{2}$, was der Suche nach dem Median entspricht.

(b) Die beste Schranke für die erwartete Vergleichsanzahl bei einem Algorithmus für das Auswahlproblem, nämlich $\frac{3}{2}n + o(n)$, erreicht ein anderer randomisierter Algorithmus (siehe Vorlesung „Randomisierte Algorithmen“).