Algorithmik

Sebastian Iwanowski
FH Wedel

2. Weitere Such- und Sortieralgorithmen

Algorithmik 2

2.1 Das Auswahlproblem (Order Statistics)

`SELECT (k,A): Element`

Gesucht wird das k-te Element eines n-elementigen unsortierten Arrays A, also das Element $x \in A$ mit der Eigenschaft, dass k Elemente aus A kleiner oder gleich sind.

Einfache Lösungen

1. Sortiere alles und bestimme dann das k-te Element.

   Laufzeit $\Theta(n \log n)$ w.c. und a.c.

2. k = 1 oder k = n:
   Einfacher Durchlauf mit Merken des Minimums bzw. Maximums

   Laufzeit $\Theta(n)$ w.c. und a.c.

Referenzen zum Nacharbeiten und Vertiefen:
Alt, S. 23
Cormen, Kap. 9.1

Algorithmik 2

2.1 Das Auswahlproblem (Order Statistics)

`SELECT (k,A): Element`

Gesucht wird das k-te Element eines n-elementigen unsortierten Arrays A, also das Element $x \in A$ mit der Eigenschaft, dass k Elemente aus A kleiner oder gleich sind.

Randomisierter Algorithmus Laufzeit $\Theta(n^2)$ w.c., $\Theta(n)$ a.c.

1. Wähle zufällig eine Position j aus {1,...,n} und bestimme Element a = A[j].
2. Vertausche die Elemente aus A mit Quicksort-Partition derart, dass a an der richtigen Position steht und alle kleineren links von a und alle größeren rechts von a.
   Teile das Array A in drei hintereinanderliegende Teile $A_<$, $A_=$ und $A_>$ auf.
3. $|A_<| < k \leq |A_<| + |A_=|$ => return a
   $k \leq |A_<|$ => return SELECT (k, $A_<$)
   $k > |A_<| + |A_=|$ => return SELECT (k - ($|A_<|+|A_=|$), $A_>$)

Referenzen zum Nacharbeiten und Vertiefen:
Alt, S. 23 – 26
Cormen, Kap. 9.2

Algorithmik 2

2.1 Das Auswahlproblem (Order Statistics)

SELECT (k,A): Element

Gesucht wird das k-te Element eines n-elementigen unsortierten Arrays A, also das Element $x \in A$ mit der Eigenschaft, dass k Elemente aus A kleiner oder gleich sind.

Deterministischer Algorithmus Laufzeit $\Theta(n)$ w.c. und a.c.

1. Teile A in $\lceil n/5 \rceil$ Teilfolgen der Länge 5 auf.
2. Sortiere jede Teilfolge i und bestimme dann den Median x_i.
3. Bestimme den Median x der Mediane durch SELECT($\lceil n/10 \rceil$,{x_1,...,$x_{\lceil n/5 \rceil}$})
4. Vertausche die Elemente aus A mit Quicksort-Partition derart, dass x an der richtigen Position steht und alle kleineren links von x und alle größeren rechts von x.
   Teile das Array A in drei hintereinanderliegende Teile $A_<$, $A_=$ und $A_>$ auf.
5. $|A_<| < k \leq |A_<| + |A_=|$ => return x
   $k \leq |A_<|$ => return SELECT (k, $A_<$)
   $k > |A_<| + |A_=|$ => return SELECT (k - ($|A_<|+|A_=|$), $A_>$)

Referenzen zum Nacharbeiten und Vertiefen:

Alt, S. 27 – 29
Cormen, Kap. 9.3

Algorithmik 2

2.2 Suchen in Sortierten Feldern

`SELECT (a,A,left,right): Element`

Gesucht wird der Index des Elements a in einem n-elementigen sortierten Array A[1..n] zwischen den Positionen left und right. Falls $a \notin A$, dann wird 0 zurückgegeben.

Vorteil gegenüber dynamischen Datenstrukturen (Kap. 3): weniger Speicherplatz

1. Binärsuche — Laufzeit $\Theta(\log n)$ w.c. und a.c.
2. Interpolationssuche — Laufzeit $\Theta(n)$ w.c. und $\Theta(\log(\log n))$ a.c.
3. Quadratische Binärsuche — Laufzeit $\Theta(n)$ w.c. und $\Theta(\log(\log n))$ a.c.

Referenzen zum Nacharbeiten und Vertiefen:
Alt, S. 30 – 35

Algorithmik 2

2.3 Sortieren in endlichen Universen

- Countingsort

  Einfache Variante (Alt) und Variante für komplexe Daten (Cormen), Laufzeit Θ(n), stabil, d.h. erhält bei Gleichheit der Daten die Eingabereihenfolge

- Bucketsort

  Im Original nur für Real-Zahlen aus [0,1) (Cormen), laufzeitentscheidend ist das Sortieren der Buckets, im Durchschnitt Θ(n), selbst wenn das Sortieren mit quadratischem Algorithmus vorgenommen wird, Beweis in Cormen (Folgefolien)

- Radixsort

  Zum Sortieren von Wörtern mit beschränkter Wortlänge über endlichem Alphabet, Laufzeit Θ(n), benutzt Countingsort (Cormen) oder Bucketsort (Alt) als Unterprozedur

- Verallgemeinerte Anwendungsmöglichkeiten aller Algorithmen

Referenzen zum Nacharbeiten und Vertiefen:
Alt, S. 21 – 22,
Cormen, Kap. 8

ganzen Zahlen innerhalb eines schmalen Bereichs besteht, geht Bucketsort davon aus, dass die Eingabesequenz durch einen zufälligen Prozess erzeugt wird, der die Elemente gleichmäßig und unabhängig voneinander über das Intervall $[0, 1)$ verteilt. (Eine Definition der gleichmäßigen Verteilung finden Sie in Abschnitt C.2.)

Die Idee von Bucketsort besteht darin, das Intervall $[0, 1)$ in n gleichgroße Teilintervalle oder ***Buckets*** aufzuteilen und dann die n Eingabewerte auf die Buckets zu verteilen. Da die Eingaben über $[0, 1)$ gleichmäßig und unabhängig voneinander verteilt sind, erwarten wir nicht, dass viele Zahlen auf das gleiche Bucket entfallen. Um die Ausgabe zu erzeugen, sortieren wir einfach die Zahlen innerhalb jedes Buckets, gehen dann der Reihe nach durch jedes Bucket und listen alle Elemente auf.

Unser Code für Bucketsort setzt voraus, dass die Eingabe ein Feld A der Länge n ist und dass jedes Feldelement $A[i]$ die Bedingung $0 \le A[i] < 1$ erfüllt. Der Code erfordert ein Hilfsfeld $B[0\,..\,n-1]$ von verketteten Listen (Buckets) und setzt voraus, dass es einen Mechanismus für die Verwaltung solcher Listen gibt. (Abschnitt 10.2 beschreibt, wie die grundlegenden Operationen mit verketteten Listen zu implementieren sind.)

```
BUCKET-SORT(A)
1  n ← länge[A]
2  for i ← 1 to n
3        do füge A[i] in die Liste B[⌊nA[i]⌋] ein
4  for i ← 0 to n − 1
5        do sortiere die Liste B[i] durch Sortieren durch Einfügen
6  verbinde die Listen B[0], B[1], ..., B[n − 1]
```

Abbildung 8.4 zeigt, wie Bucketsort auf einem Eingabefeld aus 10 Zahlen arbeitet.

Um zu verstehen, dass dieser Algorithmus korrekt arbeitet, betrachten wir zwei Elemente $A[i]$ und $A[j]$. Wir nehmen ohne Beschränkung der Allgemeinheit $A[i] \le A[j]$ an. Wegen $\lfloor nA[i] \rfloor \le \lfloor nA[j] \rfloor$ wird $A[i]$ entweder in das gleiche Bucket wie $A[j]$ oder in ein Bucket mit kleinerem Index eingefügt. Falls die Elemente $A[i]$ und $A[j]$ in das gleiche Bucket eingefügt wurden, dann muss sie die **for**-Schleife in den Zeilen 4–5 in die richtige Reihenfolge bringen. Falls $A[i]$ und $A[j]$ in verschiedene Buckets eingefügt werden, dann werden sie in Zeile 6 in die richtige Reihenfolge gebracht. Daher arbeitet Bucketsort korrekt.

Um die Laufzeit zu analysieren, stellen wir zunächst fest, dass alle Zeilen außer Zeile 5 im schlechtesten Fall die Zeit $O(n)$ benötigen. Wir müssen nun nur noch die Zeit bestimmen, die durch die n Aufrufe von Sortieren durch Einfügen in Zeile 5 verbraucht wird.

Um die Kosten für die Aufrufe von Sortieren durch Einfügen zu untersuchen, führen wir die Zufallsvariable n_i ein, die die Anzahl der Elemente in Bucket $B[i]$ beschreibt. Da Sortieren durch Einfügen in quadratischer Zeit läuft (siehe Abschnitt 2.2), ist die Laufzeit von Bucketsort

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \,.$$

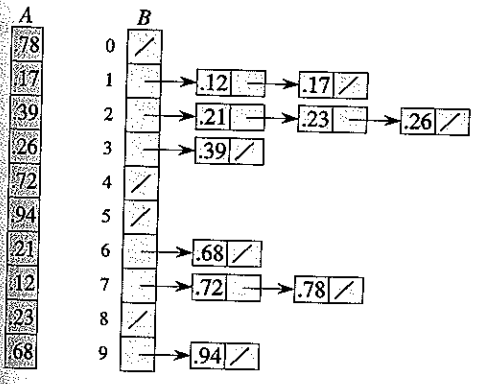

Abbildung 8.4: *Die Arbeitsweise von Bucketsort.* **(a)** *Das Eingabefeld* $A[1\,..\,10]$. **(b)** *Das Feld* $B[0\,..\,9]$ *der sortierten Listen (Buckets) nach Ausführung der Zeile 5 des Algorithmus. Bucket* i *enthält die Werte aus dem halboffenen Intervall* $[i/10, (i+1)/10)$. *Die sortierte Ausgabe besteht aus einer Verknüpfung der Listen* $B[0], B[1], \ldots, B[9]$.

Bilden wir auf beiden Seiten den Erwartungswert und nutzen wir dessen Linearität aus, so erhalten wir

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}\left[O(n_i^2)\right] && \text{(wegen der Linearität des Erwartungswertes)} \\ &= \Theta(n) + \sum_{i=0}^{n-1} O\left(\mathrm{E}\left[n_i^2\right]\right) && \text{(nach Gleichung (C.21))} \,. \end{aligned} \tag{8.1}$$

Wir behaupten, dass

$$\mathrm{E}\left[n_i^2\right] = 2 - 1/n \tag{8.2}$$

für $i = 0, 1, \ldots, n-1$ gilt. Es überrascht nicht, dass jedes Bucket i für $\mathrm{E}\left[n_i^2\right]$ den gleichen Wert hat, denn jeder Wert des Eingabefeldes A hat für jedes Bucket die gleiche Wahrscheinlichkeit, in dieses eingefügt zu werden. Um Gleichung (8.2) zu beweisen, definieren wir die Indikatorfunktionen

$$X_{ij} = \mathrm{I}\{A[j] \text{ wird in Bucket } i \text{ eingefügt}\}$$

für $i = 0, 1, \ldots, n-1$ und $j = 1, 2, \ldots, n$. Damit gilt

$$n_i = \sum_{j=1}^{n} X_{ij} \,.$$

Um $\mathrm{E}\left[n_i^2\right]$ zu berechnen, multiplizieren wir das Quadrat aus und ordnen die Terme um:

$$\begin{aligned}
\mathrm{E}\left[n_i^2\right] &= \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n\\ k\ne j}} X_{ij}X_{ik}\right] \\
&= \sum_{j=1}^{n}\mathrm{E}\left[X_{ij}^2\right] + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n\\ k\ne j}} \mathrm{E}\left[X_{ij}X_{ik}\right] . \qquad (8.3)
\end{aligned}$$

Die letzte Zeile folgt aus der Linearität des Erwartungswertes. Die beiden Summen werten wir separat aus. Die Indikatorfunktion X_{ij} ist mit Wahrscheinlichkeit $1/n$ gleich 1 und mit Wahrscheinlichkeit $1-1/n$ gleich 0; also gilt

$$\begin{aligned}
\mathrm{E}\left[X_{ij}^2\right] &= 1\cdot\frac{1}{n} + 0\cdot\left(1-\frac{1}{n}\right) \\
&= \frac{1}{n} .
\end{aligned}$$

Für $k \ne j$ sind die Variablen X_{ij} und X_{ik} unabhängig, sodass

$$\begin{aligned}
\mathrm{E}\left[X_{ij}X_{ik}\right] &= \mathrm{E}\left[X_{ij}\right]\mathrm{E}\left[X_{ik}\right] \\
&= \frac{1}{n}\cdot\frac{1}{n} \\
&= \frac{1}{n^2}
\end{aligned}$$

gilt. Substituieren wir diese beiden Erwartungswerte in Gleichung (8.3), so erhalten wir

$$\begin{aligned}
\mathrm{E}\left[n_i^2\right] &= \sum_{j=1}^{n}\frac{1}{n} + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n\\ k\ne j}} \frac{1}{n^2} \\
&= n\cdot\frac{1}{n} + n(n-1)\cdot\frac{1}{n^2} \\
&= 1 + \frac{n-1}{n} \\
&= 2 - \frac{1}{n} ,
\end{aligned}$$

womit Gleichung (8.2) bewiesen ist.

Verwenden wir diese Erwartungswerte in Gleichung (8.1), so sehen wir, dass die erwartete Laufzeit von Bucketsort in $\Theta(n) + n\cdot O(2-1/n) = \Theta(n)$ ist. Die erwartete Laufzeit des gesamten Bucketsort-Algorithmus ist also linear.

Selbst wenn die Eingabesequenz nicht einer gleichmäßigen Verteilung genügt, kann Bucketsort in linearer Zeit laufen. Solange die Eingabe die Eigenschaft hat, dass die Summe der Quadrate der Bucketgrößen linear in der Gesamtanzahl der Elemente ist, sagt uns Gleichung (8.1), dass Bucketsort in linearer Zeit laufen wird.

Übungen

8.4-1 Erläutern Sie analog zu den in Abbildung 8.4 gemachten Ausführungen, wie BUCKET-SORT auf dem Feld $A = \langle .79, .13, .16, .64, .39, .20, .89, .53, .71, .42\rangle$ arbeitet.

8.4-2 Was ist die Laufzeit im schlechtesten Fall für den Algorithmus Bucketsort? Durch welche einfache Änderung am Algorithmus kann die lineare erwartete Laufzeit erhalten und die Laufzeit im schlechtesten Fall zu $O(n \lg n)$ gemacht werden?

8.4-3 Sei X eine Zufallsvariable, die beschreibt, wie häufig bei zwei Würfen einer fairen Münze das Ereignis Kopf auftritt. Bestimmen Sie $\mathrm{E}\left[X^2\right]$ und $\mathrm{E}^2[X]$.

*8.4-4** Gegeben seien n Punkte $p_i = (x_i, y_i)$ im Einheitskreis, sodass $0 < x_i^2 + y_i^2 \le 1$ für $i = 1, 2, \ldots, n$ gilt. Angenommen, die Punkte sind gleichmäßig verteilt, d. h. die Wahrscheinlichkeit, einen Punkt in einem Bereich des Kreises zu finden, ist proportional zum Flächeninhalt dieses Bereiches. Entwerfen Sie einen Algorithmus mit der erwarteten Laufzeit $\Theta(n)$, der die n Punkte nach ihrem Abstand $d_i = \sqrt{x_i^2 + y_i^2}$ zum Ursprung sortiert. (*Hinweis:* Entwerfen Sie die Größen der Buckets so, dass sie die gleichmäßige Verteilung der Punkte im Einheitskreis widerspiegeln.)

*8.4-5** Eine ***Verteilungsfunktion*** $P(x)$ einer Zufallsvariablen X ist definiert durch $P(x) = \Pr\{X \le x\}$. Angenommen, n Zufallsvariablen $X_1, X_2, \ldots, X_n$ genügen jeweils einer stetigen Verteilungsfunktion P, die in der Zeit $O(1)$ zu berechnen ist. Zeigen Sie, wie diese Zahlen in linearer erwarteter Zeit berechnet werden können.

Problemstellung

8-1 Untere Schranken für den mittleren Fall bei vergleichenden Sortierverfahren

In dieser Problemstellung beweisen wir eine untere Schranke von $\Omega(n \lg n)$ für die erwartete Laufzeit jedes deterministischen oder randomisierten vergleichenden Sortierverfahrens angewendet auf n paarweise verschiedene Eingabeelemente. Wir