ADS: Algorithmen und Datenstrukturen

Peter F. Stadler & Konstantin Klemm

Bioinformatics Group, Dept. of Computer Science & Interdisciplinary Center for Bioinformatics, **University of Leipzig**

8. Februar 2010

Quick-Sort I

[Hoare, 1962]; auch als *Partition-Exchange-Sort* bezeichnet

Sortieren durch Austauschen mittels Divide-and-Conquer:

- Bestimmung eines Pivot-Elementes $x \in L$
- Zerlegung der Liste in zwei Teillisten L_1 und L_2 durch Austauschen, so daß linke (rechte) Teilliste L_1 (L_2) nur Schlüssel enthält, die kleiner (größer oder gleich) als x sind

$$< - - - L_1 - - - > x < - - - L_2 - - - >$$

- Rekursive Sortierung der Teillisten, bis nur noch Listen der Länge 1 verbleiben

Quick-Sort II

Realisierung der Zerlegung:

- Duchlauf des Listenbereiches von links über Indexvariable i, solange bis $L[i] \geq x$ vorliegt
- Duchlauf des Listenbereiches von rechts über Indexvariable j, solange bis $L[j] \leq x$ vorliegt
- Austausch von $L[i]$ und $L[j]$
- Fortsetzung der Durchläufe bis $i > j$ gilt

Quick-Sort Algorithmus

Algorithmus `qsort(A,l,r)` sortiert die `A[l]` bis `A[r]`:

```
i=l; j=r                      // Initialisieren:
if(r<=l) return               // Fertig?
piv = A[(l+r)/2]              // Pivot-Element
do                            // Schleife
     while(A[i]<piv) i++      // Links suchen
     while(A[j]>piv) j--      // Rechts suchen
     if(i<=j) exchange(A[i],A[j])
              i++; j--
while (i<=j)                  // Schleifenende
qsort(A,l,j)                  // Rekursiver Aufruf
qsort(A,i,r)
```

Quick-Sort Beispiel

Berechne `Q(A,l,r)`

$$67, 58, 23, {\color{green}44}, 91, 11, 30, 54$$

$$\underbrace{{\color{red}30, 11, 23}}_{{\color{red}Q(A,l,j)}}, {\color{green}44}, \underbrace{{\color{blue}91, 58, 67, 54}}_{{\color{blue}Q(A,i,r)}}$$

Quick-Sort: Eigenschaften

- *In-situ*-Verfahren
- nicht stabil
- Kosten am geringsten, wenn Teillisten stets gleichlang sind, d.h. wenn das Pivot-Element dem mittleren Schlüsselwert (Median) entspricht
- Halbierung der Teillisten bei jeder Zerlegung
  $\rightarrow$ Kosten $O(n \log n)$
- Worst-Case: Liste der Länge k wird in Teillisten der Längen 1 und $k-1$ zerlegt (z.B. bei bereits sortierter Eingabe und Wahl des ersten oder letzten Elementes als Pivot-Element)
  $\rightarrow$ Kosten $O(n^2)$

Quick-Sort: Eigenschaften

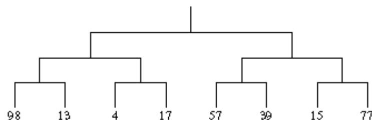
Wahl des Pivot-Elementes ist von entscheidender Bedeutung.
Sinnvolle Methoden sind:

- mittleres Element
- Mittlerer Wert von k Elementen (z.B. $k = 3$) $\rightarrow$ bei fast allen Eingabefolgen können Kosten in der Größenordnung $O(nlogn)$ erzielt werden
- Zahlreiche Variationen von Quick-Sort
  (z.B. Behandlung von Duplikaten, Umschalten auf elementares Sortierverfahren für kleine Teillisten)

Turnier-Sortierung

Maximum- bzw. Minimum-Bestimmung einer Sortierung analog zur Siegerermittlung bei Sportturnieren mit KO-Prinzip

- paarweise Wettkämpfe zwischen Spielern/Mannschaften
- nur Sieger kommt weiter
- Sieger des Finales ist Gesamtsieger
- Zugehörige Auswahlstruktur ist ein binärer Baum



Turnier-Sortierung: Zweiter Sieger

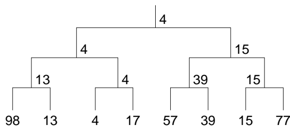
- Aber: Der zweite Sieger (das zweite Element der Sortierung) ist nicht automatisch der Verlierer im Finale
- Stattdessen: Neuaustragung des Wettkampfes auf dem Pfad des Siegers (ohne seine Beteiligung)
- Pfad für Wurzelelement hinabsteigen und Element jeweils entfernen
- Neubestimmung der Sieger

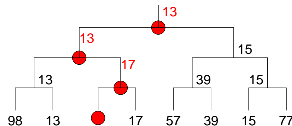
Turnier-Sortierung: Algorithmus

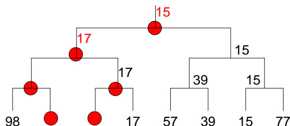
Algorithmus `TOURNAMENT_SORT`
Spiele ein KO-Turnier und erstelle dabei einen binären Auswahlbaum

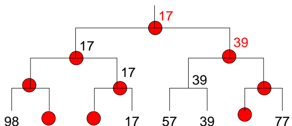
```
FOR I := 1 TO n DO
   Gib Element an der Wurzel aus
   Steige Pfad des Elementes an der Wurzel hinab und loesche es
   Steige Pfad zurueck an die Wurzel und spiele ihn dabei neu aus
END
```

Turnier-Sortierung: Beispiel weiter









Turnier-Sortierung: Aufwand

Anzahl Vergleiche (der einfachheit halber $n = 2^k$)
Initialen Auswahlbaumes $2^{k-1} + 2^{k-2} + ... + 1 = 2^k - 1 = n - 1$
pro Schleifendurchlauf Absteigen und Aufsteigen im Baum über k
Stufen: $2k = 2 \log_2 n$ Vergleiche
Gesamtaufwand $= n - 1 + 2k(n - 2) = O(n \log n)$
Platzbedarf für Auswahlbaum
$2^k + 2^{k-1} + 2^{k-2} + ... + 1 = 2k + 1 - 1 = 2n - 1$

Heap-Sort

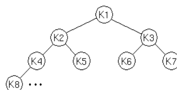
(Williams, 1964)
Reduzierung des Speicherplatzbedarfs gegenüber Turnier-Sort, indem “Löcher” im Auswahlbaum umgangen werden
Heap (Halde):

- binärer Auswahlbaum mit der Eigenschaft, dass sich das größte Element jedes Teilbaumes in dessen Wurzel befindet
- $\rightarrow$ Baumwurzel enthält größtes Element

Heap-Sort: Absinken

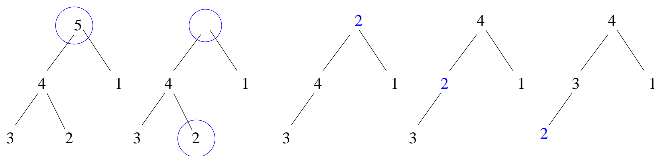
Vorgehensweise:

1. Repräsentation der Schlüsselfolge $K_1, K_2, K_3, \ldots$ in einem Binärbaum minimaler Höhe:



2. **Herstellen der Heap-Eigenschaft:** ausgehend von Blättern durch "Absinken" von Knoteninhalten, welche kleiner sind als für einen der direkten Nachfolgerknoten
3. Ausgabe des Wurzelelementes
4. Ersetzen des Wurzelelementes mit dem am weitesten rechts stehenden Element der untersten Baumebene
5. Wiederherstellen der Heap-Eigenschaft durch "Absenken" des Wurzelelementes
6. Ausgabe der neuen Wurzel usw. (solange bis Baum nur noch 1 Element enthält)

Heap-Sort: Beispiel



Heap-Sort: Algorithmus

Effiziente Realisierbarkeit mit Arrays

- Wurzelelement = erstes Feldelement $A[1]$
- direkter Vaterknoten von $A[i]$ ist $A[i/2]$
- Heap-Bedingung: $A[i] \leq A[i/2]$

Algorithmus

1. Aufbau des Heaps durch Absenken aller Vaterknoten $A[n/2]$ bis $A[1]$
2. Vertauschen von $A[1]$ mit $A[n]$ (statt Entfernen der Wurzel)
3. Reduzierung der Feldlänge n um 1
4. Falls $n > 1$:
    - Absenken der neuen Wurzel $A[1]$
    - Gehe zu 2

Heap Sort Beispiel

Ausgangsliste 57, 16, 62, 30, 80, 7, 21, 78, 41
Heap-Darstellung 80, 78, 62, 57, 16, 7, 21, 30, 41

80	78	62	57	16	7	21	30	41
41	78		57					80
78	57		41				30	|80
30							|78	80

Heap-Sort: Aufwand I

- Für $2^{k-1} \leq n \leq 2^k$ gilt:
  - Baum hat die Höhe k mit $k = 1, 2, \dots$
  - Anzahl der Knoten auf Stufe i $(0 \leq i \leq k-1)$ ist $2i$
- Kosten der Prozedur Sink
  - Schleife wird höchstens $h - 1$ mal ausgeführt (h ist Höhe des Baumes mit Wurzelelement i)
  - Zeitbedarf $O(h)$ mit $h \leq k$

Heap-Sort: Aufwand II

- Kosten zur Erstellung des anfänglichen Heaps
  - Absenken nur für Knoten mit nicht-leeren Unterbäumen (Stufe $k-2$ bis Stufe 0)
  - Kosten für einen Knoten auf der Stufe $k-2$ betragen höchstens c Einheiten, die Kosten für die Wurzel $(k-1)c$ Einheiten.
  - max. Kosten insgesamt

$$2^{k-2} \times 1c + 2^{k-3} \times 2c + \ldots + 2^0 \times (k-1)c =$$

$$= \sum_i c \cdot i \cdot 2^{k-i-1} = c2^{k-1} \sum_i 2^{-i} i < cn \sum_i 2^{-i}$$

$$< 2nc$$

  - Gesamtkosten $O(n)$
- Kosten der Sortierung: $n-1$ Aufrufe von Sink mit Kosten von höchstens $O(k) = O(\log_2 n)$
- maximale Gesamtkosten von HEAPSORT: $O(n) + O(n \log_2 n) = O(n \log_2 n)$

Sortieren durch Streuen und Sammeln

(Distribution-Sort, Bucket-Sort)
Lineare Sortierkosten $O(n)$
Kein allgemeines Sortierverfahren, sondern beschränkt auf kleine und zusammenhängende Schlüsselbereiche $1..m$
Die Verteilung der Schlüsselwerte wird für alle Werte bestimmt und daraus die relative Position jedes Eingabewertes in der Sortierfolge bestimmt
Vorgehensweise

- Hilfsfeld COUNT [1..m]
- Bestimmen der Häufigkeit des Vorkommens für jeden der m möglichen Schlüsselwerte ("Streuen")
- Bestimmung der akkumulierten Häufigkeiten in COUNT
- "Sammeln" von $i = 1$ bis m: falls COUNT[i]>0 wird i-ter Schlüsselwert COUNT[i]-mal ausgegeben

Distribution Sort: Beispiel

Beispiel: $m = 10$; Eingabe 4 0 1 1 3 5 6 9 7 3 8 5 ($n = 12$)
Streuen:

0	1	.	3	4	5	6	7	8	9
.	1	.	3	.	5	.	.	.	.

Sammel: 0 1 1 3 3 4 5 5 6 7 8 9
Kosten:

- keine Duplikate:
  Streuen $C_1 n$, Sammeln $C_2 m$
- mit d Duplikaten:
  Streuen $C_1 n$, Sammeln $C_2(m + d)$

Distribution-Sort / Fachverteilen

Verallgemeinerung: Sortierung von k-Tupeln gemäß lexikographischer Ordnung ("Fachverteilen")
Lexikographische Ordnung: $A = \{a_1, a_2, ..., a_n\}$ sei Alphabet mit gegebener Ordnung $a_1 < a_2 < ... < a_n$ die sich wie folgt auf A^* fortsetzt:

$$v \leq w \iff w = vu \text{ oder}$$
$$v = ua_1u' \text{ und } w = ua_ju'' \text{ mit}$$
$$u, u', u'' \in A^* \text{ und } a_i, a_j \in A \text{ in } i < j.$$

Die antisymmetrische Relation $\leq$ heißt lexikographische Ordnung.
Sortierung erfolgt in k Schritten:
- Sortierung nach der letzten Stelle
- Sortierung nach der vorletzten Stelle

...
- Sortierung nach der ersten Stelle Beispiel:

Sortierung von Postleitzahlen