Script **generated by TTT**

Title: TÃ¤ubig: GAD (04.06.2013)

Date: Tue Jun 04 14:26:55 CEST 2013

Duration: 80:23 min

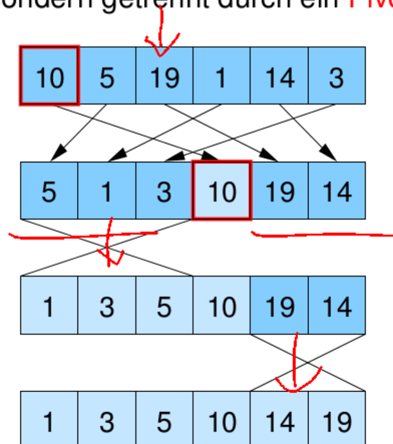
Pages: 21

Übersicht

6 Sortieren
- Einfache Verfahren
- MergeSort
- Untere Schranke
- QuickSort
- Selektieren
- Schnelleres Sortieren
- Externes Sortieren

QuickSort

Idee:
Aufspaltung in zwei Teilmengen, aber nicht in der Mitte der Sequenz wie bei MergeSort, sondern getrennt durch ein Pivotelement



QuickSort: Algorithmus

quickSort(Element[] a, int ℓ, int r) {
  // $a[\ell \dots r]$: zu sortierendes Feld
  if ($\ell < r$) {
    $p = a[r]$; // Pivot
    int $i = \ell - 1$; int $j = r$;
    do { // spalte Elemente in $a[\ell, \dots, r-1]$ nach Pivot p
      do { $i{+}{+}$ } while ($a[i] < p$);
      do { $j{-}{-}$ } while ($j \geq \ell \wedge a[j] > p$);
      if ($i < j$) swap($a[i]$, $a[j]$);
    } while ($i < j$);
    swap ($a[i]$, $a[r]$); // Pivot an richtige Stelle
    quickSort(a, ℓ, $i - 1$);
    quickSort(a, $i + 1$, r);
  }
}

QuickSort: Laufzeit

Problem:

- im Gegensatz zu MergeSort kann die Aufteilung in Teilprobleme unbalanciert sein (also nur im Optimalfall eine Halbierung)
- im worst case quadratische Laufzeit
(z.B. wenn Pivotelement immer kleinstes oder größtes aller Elemente ist)

Lösungen:

- wähle zufälliges Pivotelement:
Laufzeit $O(n \log n)$ mit hoher Wahrscheinlichkeit
- berechne Median (mittleres Element):
mit Selektionsalgorithmus, später in der Vorlesung

QuickSort

Laufzeit bei zufälligem Pivot-Element

- Zähle Anzahl Vergleiche (Rest macht nur konstanten Faktor aus)
- $\bar{C}(n)$: erwartete Anzahl Vergleiche bei n Elementen

Satz

Die erwartete Anzahl von Vergleichen für QuickSort ist

$$\bar{C}(n) \leq 2n \ln n \leq 1.39 n \log_2 n$$

QuickSort

Beweis.

- Betrachte sortierte Sequenz $\langle e'_1, \ldots, e'_n \rangle$
- nur Vergleiche mit Pivotelement
- Pivotelement ist nicht in den rekursiven Aufrufen enthalten

$\Rightarrow$ e'_i und e'_j werden höchstens einmal verglichen und zwar dann, wenn e'_i oder e'_j Pivotelement ist

QuickSort

Beweis.

- Zufallsvariable $X_{ij} \in \{0, 1\}$
- $X_{ij} = 1 \quad \Leftrightarrow \quad e'_i$ und e'_j werden verglichen

$$\bar{C}(n) = \mathbb{E}\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} \mathbb{E}\left[X_{ij}\right]$$
$$= \sum_{i<j} 0 \cdot \Pr\left[X_{ij} = 0\right] + 1 \cdot \Pr\left[X_{ij} = 1\right]$$
$$= \sum_{i<j} \Pr\left[X_{ij} = 1\right]$$

QuickSort

Lemma

$$\Pr[X_{ij} = 1] = 2/(j - i + 1)$$

Beweis.

- Sei $M = \{e'_i, \ldots, e'_j\}$
- Irgendwann wird ein Element aus M als Pivot ausgewählt.
- Bis dahin bleibt M immer zusammen.
- e'_i und e'_j werden genau dann *direkt* verglichen, wenn eines der beiden als Pivot ausgewählt wird
- Wahrscheinlichkeit:

$$\Pr[e'_i \text{ oder } e'_j \text{ aus } M \text{ ausgewählt}] = \frac{2}{|M|} = \frac{2}{j-i+1}$$

□

QuickSort

Beweis.

$$\begin{aligned}
\bar{C} &= \sum_{i<j} \Pr\left[X_{ij} = 1\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \quad = \quad 2 \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{1}{k} \\
&\leq 2 \sum_{i=1}^{n-1} \sum_{k=2}^{n} \frac{1}{k} \quad = \quad 2(n-1) \sum_{k=2}^{n} \frac{1}{k} \quad = \quad 2(n-1)(H_n - 1) \\
&\leq 2(n-1)(1 + \ln n - 1) \quad \leq \quad 2n \ln n \;=\; 2n \ln(2) \log_2(n)
\end{aligned}$$

$\ln n \leq H_n \leq \ln n + 1$

□

QuickSort

Verbesserte Version ohne Check für Array-Grenzen

```
qSort(Element[] a, int ℓ, int r) {
    while (r − ℓ ≥ n0) {
        j = pickPivotPos(a, ℓ, r);
        swap(a[ℓ],a[j]);     p = a[ℓ];
        int i = ℓ;     int j = r;
        repeat {
            while (a[i] < p) do i + +;
            while (a[j] > p) do j − −;
            if (i ≤ j) { swap(a[i], a[j]);   i + +;   j − −; }
        } until (i > j);
        if (i < (ℓ + r)/2) { qSort(a, ℓ, j);   ℓ = i; }
        else               { qSort(a, i, r);   r = j; }
    }
    insertionSort(a, ℓ, r);
}
```


Übersicht

6 Sortieren
- Einfache Verfahren
- MergeSort
- Untere Schranke
- QuickSort
- Selektieren
- Schnelleres Sortieren
- Externes Sortieren

Rang-Selektion

- Bestimmung des kleinsten und größten Elements ist mit einem einzigen Scan über das Array in Linearzeit möglich
- Aber wie ist das beim k-kleinsten Element, z.B. beim $\lfloor n/2 \rfloor$-kleinsten Element (Median)?

Problem:

Finde k-kleinstes Element in einer Menge von n Elementen

Rang-Selektion

- Bestimmung des kleinsten und größten Elements ist mit einem einzigen Scan über das Array in Linearzeit möglich
- Aber wie ist das beim k-kleinsten Element, z.B. beim $\lfloor n/2 \rfloor$-kleinsten Element (Median)?

Problem:

Finde k-kleinstes Element in einer Menge von n Elementen

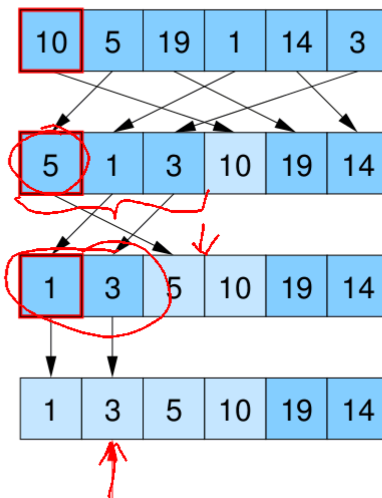
Naive Lösung: Sortieren und k-tes Element ausgeben

$\Rightarrow$ Zeit $O(n \log n)$

Geht das auch schneller?

QuickSelect

Ansatz: ähnlich zu QuickSort, aber nur eine Seite betrachten



QuickSelect

Methode analog zu QuickSort

```
Element quickSelect(Element[] a, int l, int r, int k) {
  // a[l...r]: Restfeld,   k: Rang des gesuchten Elements
  if (r == l) return a[l];
  int z = zufällige Position in {l,...,r};   swap(a[z], a[r]);
  Element v = a[r];   int i = l − 1;   int j = r;
  do {   // spalte Elemente in a[l,...,r − 1] nach Pivot v
    do i + + while (a[i] < v);
    do j − − while (a[j] > v && j ≠ l);
    if (i < j) swap(a[i], a[j]);
  } while (i < j);
  swap (a[i], a[r]);   // Pivot an richtige Stelle
  if (k < i) return quickSelect(a, l, i − 1, k);
  if (k > i) return quickSelect(a, i + 1, r, k);
  else return a[k];   // k == i
}
```

QuickSelect
Alternative Methode

```
Element select(Element[] s, int k) {
  assert(|s| ≥ k);
  Wähle p ∈ s zufällig (gleichverteilt);

  Element[] a := {e ∈ s : e < p};
  if (|a| ≥ k)
    return select(a,k);

  Element[] b := {e ∈ s : e = p};
  if (|a| + |b| ≥ k)
    return p;

  Element[] c := {e ∈ s : e > p};
  return select(c,k − |a| − |b|);
}
```


QuickSelect
Alternative Methode

Beispiel

s	k	a b c
$\langle 3,1,4,1,5,9,2,6,5,3,5,8,9\rangle$	7	$\langle 1,1\rangle\ \langle 2\rangle\ \langle 3,4,5,9,6,5,3,5,8,9\rangle$
$\langle 3,4,5,9,6,5,3,5,8,9\rangle$	4	$\langle 3,4,5,5,3,5\rangle\ \langle 6\rangle\ \langle 9,8,9\rangle$
$\langle 3,4,5,5,3,5\rangle$	4	$\langle 3,4,3\rangle\ \langle 5,5,5\rangle\ \langle\rangle$

In der sortierten Sequenz würde also an 7. Stelle das Element 5 stehen.

Hier wurde das mittlere Element als Pivot verwendet.

QuickSelect

teilt das Feld jeweils in 3 Teile:

- a Elemente kleiner als das Pivot
- b Elemente gleich dem Pivot
- c Elemente größer als das Pivot

$T(n)$: erwartete Laufzeit bei n Elementen

Satz

Die erwartete Laufzeit von QuickSelect ist linear: $T(n) \in O(n)$.

QuickSelect

Beweis.

- Pivot ist gut, wenn weder a noch c länger als $2/3$ der aktuellen Feldgröße sind:

schlecht	gut	schlecht

$\Rightarrow$ Pivot ist gut, falls es im mittleren Drittel liegt

$$p = \Pr[\text{Pivot ist gut}] = 1/3$$

QuickSelect

Beweis.

- Pivot ist gut, wenn weder a noch c länger als $2/3$ der aktuellen Feldgröße sind:

$\Rightarrow$ Pivot ist gut, falls es im mittleren Drittel liegt

$$p = \Pr[\text{Pivot ist gut}] = 1/3$$

Erwartete Zeit bei n Elementen

- linearer Aufwand außerhalb der rekursiven Aufrufe: cn
- Pivot gut (Wsk. $1/3$): Restaufwand $\leq T(2n/3)$
- Pivot schlecht (Wsk. $2/3$): Restaufwand $\leq T(n-1) < T(n)$

QuickSelect

Beweis.

$$
\begin{aligned}
T(n) &\leq cn + p \cdot T(n \cdot 2/3) + (1-p) \cdot T(n) \\
p \cdot T(n) &\leq cn + p \cdot T(n \cdot 2/3) \\
T(n) &\leq cn/p + T(n \cdot 2/3) \\
&\leq cn/p + c \cdot (n \cdot 2/3)/p + T(n \cdot (2/3)^2) \\
&\dots \text{ wiederholtes Einsetzen} \\
&\leq (cn/p)(1 + 2/3 + 4/9 + 8/27 + \dots) \\
&\leq \frac{cn}{p} \cdot \sum_{i \geq 0} (2/3)^i \\
&\leq \frac{cn}{1/3} \cdot \frac{1}{1-2/3} = 9cn \in O(n)
\end{aligned}
$$

□