Script generated by TTT

Title: Seidl: GAD (18.05.2016)

Date: Wed May 18 13:23:27 CEST 2016

Duration: 46:41 min

Pages: 16

Master-Theorem

Lösung von Rekursionsgleichungen

Satz (vereinfachtes Master-Theorem)

Seien a, b, c, d positive Konstanten und $n = b^k$ mit $k \in \mathbb{N}$.

Betrachte folgende Rekursionsgleichung:

$$r(n) = \begin{cases} a & \text{falls } n = 1, \\ cn + d \cdot r(n/b) & \text{falls } n > 1. \end{cases}$$

Dann gilt:

$$r(n) = \begin{cases} \Theta(n) & \text{falls } d < b, \\ \Theta(n \log n) & \text{falls } d = b, \\ \Theta(n^{\log_b d}) & \text{falls } d > b. \end{cases}$$

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

- Level 0: 1 Problem der Größe $n = b^k$
- Level i: d^i Probleme der Größe $n/b^i = b^{k-i}$
- Level k: d^k Probleme der Größe $n/b^k = b^{k-k} = 1$, hier jedes mit Kosten a, also Kosten ad^k
- $d = b$: Kosten $ad^k = ab^k = an \in \Theta(n)$ auf Level k, $cnk = cn \log_b n \in \Theta(n \log n)$ für den Rest
- $d < b$: Kosten $ad^k < ab^k = an \in O(n)$ auf Level k,

Rest: $$cn \sum_{i=0}^{k-1} \left(\frac{d}{b}\right)^i = cn \frac{1 - (d/b)^k}{1 - d/b} < cn \frac{1}{1 - d/b} \in O(n)$$

$$> cn \in \Omega(n) \quad \Rightarrow \quad \Theta(n)$$

Master-Theorem

Lösung von Rekursionsgleichungen

Satz (vereinfachtes Master-Theorem)

Seien a, b, c, d positive Konstanten und $n = b^k$ mit $k \in \mathbb{N}$.

Betrachte folgende Rekursionsgleichung:

$$r(n) = \begin{cases} a & \text{falls } n = 1, \\ cn + d \cdot r(n/b) & \text{falls } n > 1. \end{cases}$$

Dann gilt:

$$r(n) = \begin{cases} \Theta(n) & \text{falls } d < b, \\ \Theta(n \log n) & \text{falls } d = b, \\ \Theta(n^{\log_b d}) & \text{falls } d > b. \end{cases}$$

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

- $d > b$: $n = b^k$, also $k = \log_b n = log_b d \cdot \log_d n$

$$d^k = d^{\log_b n} = d^{log_d n \cdot \log_b d} = n^{\log_b d}$$

Kosten $an^{\log_b d} \in \Theta\left(n^{\log_b d}\right)$ auf Level k,

Rest: $$cb^k \frac{(d/b)^k - 1}{d/b - 1} = c\frac{d^k - b^k}{d/b - 1} = cd^k \frac{1 - (b/d)^k}{d/b - 1} \in \Theta\left(d^k\right) \in \Theta\left(n^{\log_b d}\right)$$

Master-Theorem

Lösung von Rekursionsgleichungen

Satz (vereinfachtes Master-Theorem)

Seien a, b, c, d positive Konstanten und $n = b^k$ mit $k \in \mathbb{N}$.

Betrachte folgende Rekursionsgleichung:

$$r(n) = \begin{cases} a & \text{falls } n = 1, \\ cn + d \cdot r(n/b) & \text{falls } n > 1. \end{cases}$$

Dann gilt:

$$r(n) = \begin{cases} \Theta(n) & \text{falls } d < b, \\ \Theta(n \log n) & \text{falls } d = b, \\ \Theta(n^{\log_b d}) & \text{falls } d > b. \end{cases}$$

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

- $d > b$: $n = b^k$, also $k = \log_b n = log_b d \cdot \log_d n$

$$d^k = d^{\log_b n} = d^{log_d n \cdot \log_b d} = n^{\log_b d}$$

Kosten $an^{\log_b d} \in \Theta\left(n^{\log_b d}\right)$ auf Level k,

Rest: $$cb^k \frac{(d/b)^k - 1}{d/b - 1} = c\frac{d^k - b^k}{d/b - 1} = cd^k \frac{1 - (b/d)^k}{d/b - 1} \in \Theta\left(d^k\right) \in \Theta\left(n^{\log_b d}\right)$$

Untere Schranke

MergeSort hat Laufzeit $O(n \log n)$ im worst case.

InsertionSort kann so implementiert werden, dass es im best case lineare Laufzeit hat.

Gibt es Sortierverfahren mit Laufzeit besser als $O(n \log n)$ im worst case, z.B. $O(n)$ oder $O(n \log \log n)$?

$\Rightarrow$ nicht auf der Basis einfacher Schlüsselvergleiche

Entscheidungen: $x_i < x_j \quad \rightarrow \quad$ ja / nein

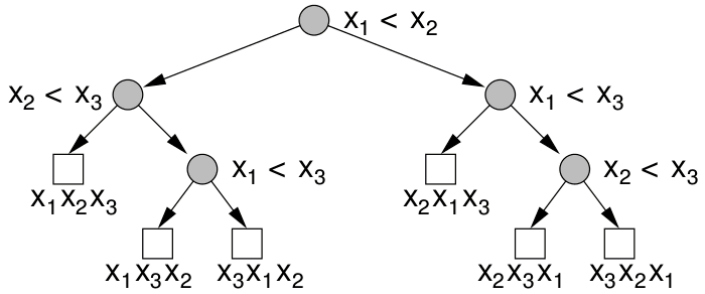
Satz

Jeder vergleichsbasierte Sortieralgorithmus benötigt im worst case mindestens $n \log n - O(n) \in \Theta(n \log n)$ Vergleiche.

Untere Schranke

Vergleichsbasiertes Sortieren

Entscheidungsbaum mit Entscheidungen an den Knoten:



Untere Schranke

Vergleichsbasiertes Sortieren

muss insbesondere auch funktionieren, wenn alle n Schlüssel verschieden sind

$\Rightarrow$ Annahme: alle verschieden

Wieviele verschiedene Ergebnisse gibt es?
$\Rightarrow$ alle Permutationen:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+O\left(\frac{1}{n}\right)\right) \geq \frac{n^n}{e^n}\sqrt{2\pi n}$$

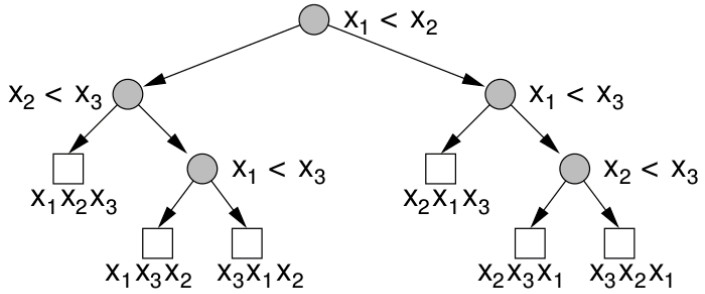
Binärbaum der Höhe h hat höchstens 2^h Blätter bzw.
Binärbaum mit b Blättern hat mindestens Höhe $\log_2 b$

$$\Rightarrow h \geq \log_2(n!) \geq n\log n - n\log e + \tfrac{1}{2}\log(2\pi n)$$

Untere Schranke

Vergleichsbasiertes Sortieren

Entscheidungsbaum mit Entscheidungen an den Knoten:



Untere Schranke

Vergleichsbasiertes Sortieren

muss insbesondere auch funktionieren, wenn alle n Schlüssel verschieden sind

$\Rightarrow$ Annahme: alle verschieden

Wieviele verschiedene Ergebnisse gibt es?
$\Rightarrow$ alle Permutationen:

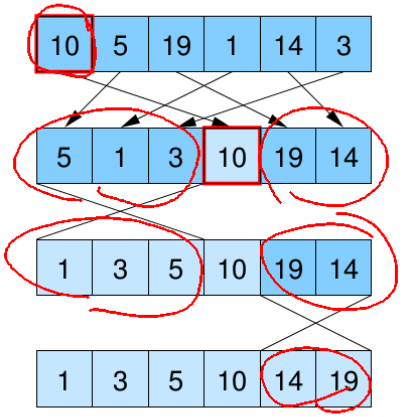
$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+O\left(\frac{1}{n}\right)\right) \geq \frac{n^n}{e^n}\sqrt{2\pi n}$$

Binärbaum der Höhe h hat höchstens 2^h Blätter bzw.
Binärbaum mit b Blättern hat mindestens Höhe $\log_2 b$

$$\Rightarrow h \geq \log_2(n!) \geq n\log n - n\log e + \tfrac{1}{2}\log(2\pi n)$$

QuickSort

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



Implementierung: effizient und in-place

```
void quickSort(Element[ ] a, int ℓ, int r) {
    // a[ℓ ... r]: zu sortierendes Feld
    if (ℓ < r) {
        p = a[r];   // Pivot
        int i = ℓ − 1;   int j = r;
        do {   // spalte Elemente in a[ℓ, ..., r − 1] nach Pivot p
            do { i++ } while (a[i] < p);
            do { j−− } while (j ≥ ℓ ∧ a[j] > p);
            if (i < j) swap(a, i, j);
        } while (i < j);
        swap (a, i, 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)

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 mit zufällig ausgewähltem Pivotelement ist

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