ETH zürich

D&A - Übungsstunde 4

Diese Folien basieren auf denjenigen der Vorlesung, wurden aber durch den Assistenten Adel Gavranović adaptiert und erweitert

Übersicht

Heutiges Programm

Follow-up
Feedback zu [code]expert
Algorithmen und Laufzeiten
Amortisierte Analyse
Codeanalyse
Dynamisch alloziierter Speicher
Tipps zu [code]expert
Outro
Code-Beispiel: Dynamischer Vektor

n.ethz.ch/~agavranovic

- Link zum Material für die Übungsstunden
- Webseite des Assistenten
- Mail an Assistenten

1. Follow-up

Follow-up aus vorheriger Übungsstunde

- Die Laufzeit für die Codes aus der Ersten Übungsstunde habe ich *noch immer* nicht...
- *Cookies?*
- Theorem zur Mastermethode hatte möglicherweise einen Fehler
  - wir wissen Bescheid
  - ihr werdet keinen Punkte deswegen verlieren

2. Feedback zu [code]expert

Allgemeines zu [code]expert

- Löscht die auskommentierten Zeilen raus bevor ihr submitted

Task "Improving Sorting"

- Wieso haben so viele einfach den Code unverändert submitted?

Task "Langstrasse Sprint"

- Entschuldigung für die vielen unnötigen anderen Funktionen
- Nur `find_closest_sliding_window()` musste implementiert werden

Task "Throwing eggs"

- Strategie für beliebig viele Eier und n Etagen?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?
  - Von unten anfangen. n Versuche.

Eier werfen

Strategie mit zwei Eiern

- 1. Ansatz. Intervalle gleicher Länge: Unterteile n in k Intervalle.
  Maximale Anzahl Versuche: $f(k) = k + n/k - 1$
  Minimiere maximale Anzahl Versuche: $f'(k) = 1 - n/k^2 = 0 \Rightarrow k = \sqrt{n}$.
  $n = 100 \Rightarrow 19$ Versuche. $\Theta(\sqrt{n})$
- 2. Ansatz: Beziehe ersten Wurfversuch in die Berechnung ein mit kleiner werdenden Intervallen. Wähle kleinstes s mit
  $s + s - 1 + s - 2 + ... + 1 = s(s+1)/2 \geq n$. Wenn $n = 100$ dann $s = 14$.
  Maximale Anzahl Versuche: $s \in \Theta(\sqrt{n})$

Asymptotisch sind beide Methoden gleich gut. Praktisch ist der zweite Ansatz vorzuziehen.

Fragen zu [code]expert eurerseits?

3. Algorithmen und Laufzeiten

Wahl des Algorithmus

Unter einer sehr grossen Anzahl (n) anwesender Studierender soll ein Preis vergeben werden. Die Person mit der Median Leginummer gewinnt. Es kommt zum Streit, welcher Algorithmus zur Ermittlung des Medians verwendet werden soll. Welche Aussage ist richtig?

Um im schlechtesten Fall eine Laufzeit von $\mathcal{O}(n \log(n))$ zu erhalten, verwenden wir…

- …Mergesort
- …BubbleSort
- …Sortiern durch Auswählen
- …Quicksort

Laufzeit des Algorithmus

Unter einer sehr grossen Anzahl (n) anwesender Studierender soll ein Preis vergeben werden. Die Person mit der Median Leginummer gewinnt. Es kommt zum Streit, welcher Algorithmus zur Ermittlung des Medians verwendet werden soll. Welche Aussage ist richtig?

Wir verwenden Quickselect ▸ Wikipedia mit zufälliger Auswahl des Pivots. Damit haben wir…

- …im schlechtesten Fall eine Laufzeit von $\mathcal{O}(n \log(n))$
- …im schlechtesten Fall eine Laufzeit von $\mathcal{O}(n)$
- …eine erwartete Laufzeit von $\mathcal{O}(\log(n))$
- …eine erwartete Laufzeit von $\mathcal{O}(n)$

4. Amortisierte Analyse

Amortisierte Laufzeitanalyse

Drei Methoden

- Aggregierte Analyse
- Kontomethode
- Potentialmethode

Beispiel: einfaches Wörterbuch

Unterstützt Operationen Insert und Find.

Idee:

- Familie von Arrays A_i mit Grösse 2^i
- Jedes Array ist entweder ganz leer oder ganz voll und speichert seine Elemente in sortierter Reihenfolge
- Zwischen den Arrays besteht keine weitere Beziehung

Daten $\{1, 8, 10, 18, 20, 24, 36, 48, 50, 75, 99\}$, $n = 11$

$$
\begin{aligned}
A_0&: \quad [50] \\
A_1&: \quad [8, 99] \\
A_2&: \quad \emptyset \\
A_3&: \quad [1, 10, 18, 20, 24, 36, 48, 75]
\end{aligned}
$$

Beispiel: einfaches Wörterbuch

Daten $\{1, 8, 10, 18, 20, 24, 36, 48, 50, 75, 99\}$, $n = 11$

$$
\begin{aligned}
A_0&: \quad [50] \\
A_1&: \quad [8, 99] \\
A_2&: \quad \emptyset \\
A_3&: \quad [1, 10, 18, 20, 24, 36, 48, 75]
\end{aligned}
$$

Algorithmus **`Find`**: Durchlaufen aller Arrays, jeweils binäre Suche
Worst-case Laufzeit : $\Theta(\log^2 n)$,

$$\log 1 + \log 2 + \log 4 + \cdots + \log 2^k = \sum_{i=0}^{k} \log_2 2^i = \frac{k \cdot (k+1)}{2} \in \Theta(\log^2 n).$$

$(k = \lfloor \log_2 n \rfloor)$

Beispiel: einfaches Wörterbuch

Algorithmus **`Insert(x)`**:

- Neues Array $A'_0 \leftarrow [x]$, $i \leftarrow 0$
- Solange $A_i \neq \emptyset$, setze $A'_{i+1} = \text{Merge}(A_i, A'_i)$, $A_i \leftarrow \emptyset$, $i \leftarrow i + 1$
- Setze $A_i \leftarrow A'_i$

Insert(11)

$$
\begin{array}{ll} A_0: & [50] \\ A_1: & [8, 99] \\ A_2: & \emptyset \\ A_3: & [1, 10, 18, \ldots, 75] \end{array}
\quad
\begin{array}{ll} A'_0: & [11] \\ A'_1: & [11, 50] \\ A'_2: & [8, 11, 50, 99] \end{array}
\quad \Rightarrow \quad
\begin{array}{ll} A_0: & \emptyset \\ A_1: & \emptyset \\ A_2: & [8, 11, 50, 99] \\ A_3: & [1, 10, 18, \ldots, 75] \end{array}
$$

Kosten Insert

Im Folgenden: $n = 2^k$, $k = \log_2 n$

Annahme: Erzeugen neues Array A'_i der Länge 2^i (und, für $i > 0$ anschliessendes Zusammenführen von A'_{i-1} und A_{i-1}) hat Kosten $\Theta(2^i)$

Im schlechtesten Fall erzeugt das Einfügen eines Elements $\log_2 n$ solche Operationen.

$\Rightarrow$ **Worst-case Kosten Insert**:

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1 \in \Theta(n).$$

Aggregatanalyse

Level	Kosten	Beispiel Array
0	1	$[*]$
1	2	$[*, *]$
2	4	$[*, *, *, *]$
3	8	$\emptyset$
4	16	$[*, *, *, *, *, *, *, *, *, *, *, *, *, *, *, *]$

Beobachtung: startet man mit einem leeren Container, so gelangt die Einfügesequenz jedes Mal auf Level 0, jedes zweite Mal auf Level 1 (mit Kosten 2), jedes vierte Mal das Level 2 (mit Kosten 4) etc.

- Gesamtkosten: $1 \cdot \frac{n}{1} + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + \cdots + 2^k \cdot \frac{n}{2^k} = (k+1)n$
  Dies ist in $\Theta(n \log n)$, weil $k = \log_2 n$.
- **Amortisierte Kosten pro Operation**: $\Theta((n \log n)/n) = \Theta(\log n)$.

Kontomethode

- Jedes Element i ($1 \leq i \leq n$) zahlt $a_i = \log_2 n$ Münzen, wenn es in die Datenstruktur eingefügt wird.
- Damit bezahlt das Element das Anlegen des ersten Arrays und jeden weiteren Merge-Schritt, welcher auftreten kann, bis das Element im Array A_k angekommen ist.
- Das Konto weist damit immer genügend Kredit auf, um alle Merge-Operationen zu bezahlen.

$\Rightarrow$ **Amortisierte Kosten** für das Einfügen $\mathcal{O}(\log n)$

Potentialmethode

Wir wissen von der Kontomethode, dass jedes Element auf dem Weg nach oben $\log n$ Münzen benötigt, d.h. dass ein **Element auf dem Level** i **noch** $k - i$ **Münzen** haben sollte. Wir verwenden das **Potential**

$$\Phi_j = \sum_{0 \leq i \leq k : A_i \neq \emptyset} (k - i) \cdot 2^i$$

Potentialmethode

Für die **Änderung des Potentials** $\Phi_j - \Phi_{j-1}$ müssen wir nur die unteren l Levels betrachten, welche zum Zeitpunkt $j-1$ belegt sind (Analogie zum binären Zähler). Sei also l der kleinste Index, so dass A_l leer ist.

Nach dem Zusamenführen der Arrays $A_0 \dots A_{l-1}$ sind Arrays $A_i, 0 \leq i < l$ leer und das Level A_l ist nun belegt. Also:

$$\Phi_j - \Phi_{j-1} = (k-l) \cdot 2^l - \sum_{i=0}^{l-1} (k-i) \cdot 2^i$$

Reale Kosten:

$$t_j = \sum_{i=0}^{l} 2^i = 2^{l+1} - 1$$

Potentialmethode

$$
\begin{aligned}
\Phi_j - \Phi_{j-1} &= (k-l)\cdot 2^l - \sum_{i=0}^{l-1}(k-i)\cdot 2^i \\
&= (k-l)\cdot 2^l - k\cdot(2^l-1) + \sum_{i=0}^{l-1} i\cdot 2^i \\
&= (k-l)\cdot 2^l - k\cdot(2^l-1) + l\cdot 2^l - 2^{l+1} + 2 \\
&= k - 2^{l+1} + 2 \\
\Phi_j - \Phi_{j-1} + t_j &= k - 2^{l+1} + 2 + 2^{l+1} - 1 = k + 1 \in \Theta(\log n)
\end{aligned}
$$

$\sum i \cdot \lambda^i$

Immer der gleiche Trick:

$$\begin{aligned}
\lambda \cdot \sum_{i=0}^{n} i \cdot \lambda^i - \sum_{i=0}^{n} i \cdot \lambda^i &= \sum_{i=0}^{n} i \cdot \lambda^{i+1} - \sum_{i=0}^{n} i \cdot \lambda^i = \sum_{i=1}^{n+1} (i-1) \cdot \lambda^i - \sum_{i=0}^{n} i \cdot \lambda^i \\
&= n \cdot \lambda^{n+1} + \sum_{i=1}^{n} (i-1) \cdot \lambda^i - i \cdot \lambda = n \cdot \lambda^{n+1} - \sum_{i=1}^{n} \lambda^i \\
&= n \cdot \lambda^{n+1} - \frac{\lambda^{n+1} - 1}{\lambda - 1} + 1 \\
(\lambda - 1) \cdot \sum_{i=0}^{n} i \cdot \lambda^i &= n \cdot \lambda^{n+1} - \frac{\lambda^{n+1} - 1}{\lambda - 1} + 1
\end{aligned}$$

Für $\lambda = 2$:

$$\sum_{i=0}^{n} i \cdot 2^i = n \cdot 2^{n+1} - 2^{n+1} + 1 + 1 = (n-1) \cdot 2^{n+1} + 2$$

5. Codeanalyse

Quiz

```
void g(unsigned n){
  for (unsigned k = 1; k != n ; ++k){
    unsigned prev = k-1;
    for (unsigned num = k; num != 0; num /= 2){
      if (num % 2 != prev % 2)
        f();
      prev /= 2;
    }
  }
}
```

Was macht dieser Code überhaupt?
Er ruft `f()` auf, für jedes Bit, dass beim Schritt $k-1 \to k$ geflippt wird.

Quiz

```
void g(unsigned n){
  for (unsigned k = 1; k != n ; ++k){
    unsigned prev = k-1;
    for (unsigned num = k; num != 0; num /= 2){
      if (num % 2 != prev % 2)
        f();
      prev /= 2;
    }
  }
}
```

Anzahl Aufrufe von `f()` $\in \Theta(n)$
(Zählerbeispiel aus der Vorlesung)

6. Dynamisch alloziierter Speicher

Wiederholung dynamisch alloziierter Speicher

Sehr wichtig: Jedes `new` braucht sein `delete` und nur eins!

Deshalb “Rule of three”:

- constructor
- copy constructor
- destructor

Faule Variante: „Rule of two”:

- niemals kopieren (unsicher)
- mache copy constructor privat (sicher) oder deleted

7. Tipps zu [code]expert

Tipps für nächste [code]expert -Aufgaben

Mini-Tutorial zu LaTeX

- Jetzt gibt's in den Aufgaben ein Mini-Tutorial zu LaTeX!
  - z. B. in der Aufgabe "Compare Sorting Algorithms"
  - *Bitte* nutzt es

Aufgabe "Compare Sorting Algorithms"

- Formattiert eure Antwort bitte dementsprechen hübsch

Tipps für nächste [code]expert -Aufgaben

Aufgabe "Stable and In-Situ Sorting"

- *"in their unmodified form"*

Aufgabe "Amortized Analysis"

- Ottman/Widmayer, Kapitel 3.3
- Cormen et al, Kapitel 17

Tipps für nächste [code]expert -Aufgaben

Aufgabe "Double Ended Queue"

- Zeitintensiv (fangt früh an...)
- Dynamische Datentypen und Speicherverwaltung
- Thema 25 im Kurs "Informatik"
- Material aus meiner Zeit als Informatik-TA könnten helfen ▸ Link (insb. ab Woche 11)
- *"By the way: the name Double Ended Queue may be misleading because it suggests to be implemented with a linked list. This would make it hard, if not impossible, to fulfill the requirements stated above. Rather* ***think of something like a vector*** *and extend it with* `push_front()`*."*

8. Outro

Allgemeine Fragen?

Bis zum nächsten Mal

Schönes Wochenende!

9. Code-Beispiel: Dynamischer Vektor

Vorbereitung für Deque-Aufgabe