

Bachelorarbeit

Andreas Windorfer

11. Mai 2020

Zusammenfassung

Inhaltsverzeichnis

1	Fazit	4
2	Einführung der amortisierten Laufzeitanalyse	4

1 Fazit

2 Einführung der amortisierten Laufzeitanalyse

Sei $i \in \{0, \dots, m\}$. Bei der **amortisierten Laufzeitanalyse** wird eine Folge von m Operationen betrachtet. Hierbei kann es sich m mal um die gleiche Operation handeln, oder auch um verschiedene. Die **tatsächlichen Kosten** t_i stehen für die exakten Kosten zum ausführen der i -ten Operation. Durch aufaddieren der tatsächlichen Kosten jeder einzelnen Operation erhält man **tatsächlichen Gesamtkosten**. Stehen für die Laufzeit der Operationen jeweils nur obere Schranken zur Verfügung, kann man mit diesen genau so vorgehen, um eine obere Schranke für die Gesamtlaufzeit zu erhalten. So erzeugte obere Schranken können jedoch unnötig hoch sein. Die Idee bei einer amortisierten Analyse ist es, bereits eingesparte Zeit durch schnell ausgeführte Operationen, den noch folgenden Operationen zum Verbrauchen zur Verfügung zu stellen. Dabei wird insbesondere der aktuelle Zustand der zugrunde liegenden Datenstruktur vor und nach einer Operation betrachtet. Hier soll die amortisierte Laufzeitanalyse verwendet werden um im folgenden Abschnitt eine niedrigere obere Schranke als $O(\log(n))$ für `einfügenFixup` zu finden. Es gibt drei Methoden zur amortisierten Analyse, hier wird die **Potentialfunktionmethode** verwendet.

Potentialfunktionmethode Eine Potentialfunktion $\Phi(D)$ ordnet einem Zustand einer Datenstruktur D eine natürliche Zahl, **Potential** genannt, zu. Es bezeichnet $\Phi(D)_i$ das Potential von D nach Ausführung der i -ten Operation. t_i steht für die **tatsächlichen Kosten** zum durchführen der i -ten Operation. Dabei handelt es sich um die exakten Kosten die beim Die **amortisierten Kosten** a_i einer Operation berücksichtigen die von der Operation verursachte Veränderung am Potential, $a_i = t_i + \Phi(D)_i - \Phi(D)_{i-1}$. Um die **amortisierten Gesamtkosten** A zu berechnen bildet man die Summe der amortisierten Kosten aller Operationen.

$$A = \sum_{i=1}^m a_i = \sum_{i=1}^m (t_i + \Phi(D)_i - \Phi(D)_{i-1}) = \Phi(D)_m - \Phi(D)_0 + \sum_{i=1}^m t_i$$

Folgendes gilt für die Summe der t_i :

$$\begin{aligned} \sum_{i=1}^m t_i &= \sum_{i=1}^m (a_i - \Phi(D)_i + \Phi(D)_{i-1}) = \Phi(D)_0 - \Phi(D)_m + \sum_{i=1}^m a_i \\ \Rightarrow \left(\Phi(D)_m \geq \Phi(D)_0 \Rightarrow \sum_{i=1}^m a_i \geq \sum_{i=1}^m t_i \right) \end{aligned}$$

Ist das Potenzial nach Ausführung der Operationsfolge also nicht kleiner als zum Beginn, dann sind die amortisierten Gesamtkosten eine obere Schranke für die tatsächlichen Gesamtkosten. Die wesentliche Aufgabe ist es nun eine Potentialfunktion zu finden, bei der die amortisierten Gesamtkosten möglichst niedrig sind und für die gilt $\Phi(D)_m \geq \Phi(D)_0$. Dies wird jetzt noch an einem einfachen Beispiel demonstriert, bevor eingefügen/fixup betrachtet wird.

Potentialfunktionmethode am Beispiel eines Stack Der Stack verfügt wie gewöhnlich über eine Operation *push* zum Ablegen eines Elementes auf dem Stack und über *pop* zum Entfernen des oben liegenden Elementes. Zusätzlich gibt es eine Operation *popAll*, die so oft *pop* aufruft, bis der Stack leer ist. Sei n die Anzahl der Elemente, die maximal im Stack enthalten sein kann. *push* und *pop* können in konstanter Zeit durchgeführt werden und wir berechnen jeweils eine Kosteneinheit. Für die Laufzeit von *popAll* gilt $O(n)$, da *pop* bis zu n mal aufgerufen wird. Für die Gesamtlaufzeit einer Folge von m Operationen kann sicher $O(mn)$ angegeben werden. Mit einer amortisierten Analyse wird nun aber $O(m)$ für *popAll* gezeigt. Als Φ verwenden wir eine Funktion, welche die aktuelle Anzahl der im Stack enthaltenen Elemente zurück gibt. Φ_0 setzen wir auf 0, das heißt wir starten mit einem leeren Stack. *push* erhöht also das Potential um eins, während *pop* es um eins vermindert. Nun werden die amortisierten Kosten bestimmt.

$$\begin{aligned} a_{push} &= t_{push} + \Phi i - \Phi i - 1 &= 2 \\ a_{pop} &= t_{pop} + \Phi i - \Phi i - 1 &= 0 \\ a_{popAll} &= n \cdot a_{pop} &= 0 \end{aligned}$$

Alle drei Operationen haben konstante amortisierte Kosten. Auf jedem Fall gilt $\Phi_m \geq \Phi_0 = 0$. Damit gilt für die Ausführungszeit der Folge $O(m)$. Bei diesem einfachen Beispiel ist sofort klar, warum es funktioniert. Aus einem zu Beginn leeren Stack kann nur entfernt werden, was zuvor eingefügt wurde. *push* zahlt für die Operation, welche das eingefügte Element eventuell wieder entfernt, gleich mit, bleibt bei den Kosten aber konstant. Deshalb

kann pop amortisiert kostenlos durchgeführt werden, wodurch einer der beiden Faktoren zur Berechnung der Kosten von $popAll$ zu 0 wird.

Literatur