Übungen zur Vorlesung
Einführung in die Algorithmik WS 15/16

Hasso-Plattner-Institut
Potsdam
13. Dezember 2015

Moritz Eissenhauer

Lösung zu Übungsblatt 7

Lösung zu Aufgabe 3

a)

Die Amorisierten Kosten seien wie folgt:

$$\begin{aligned} &1 \text{ für } 0 \to 1 \\ &2 \text{ für } 1 \to 2 \\ &0 \text{ für } 2 \to 0 \end{aligned} \tag{1}$$

Wobei $0 \to 1$ die Änderung eines Trit von 0 auf 1 sei.
Die Tatsächlichen Kosten der i-ten Inkrementierung ergeben sich aus $t_i = e_i + z_i + n_i$. e_i ist die Anzahl der Tritänderung von 0 auf 1, z_i von 1 auf 2 und n_i von 2 auf 0. Mit den oben angegebenen Kosten erhält man für die amorisierten Kosten der i-ten Inkrementierung $a_i = 1e_i + 2z_z + 0n_i$.
Das Guthaben entspricht immer der Anzahl der 2en, weil genau dann ein Guthaben hinzugefügt wird wenn eine neue 2 entsteht, und genau dann ein Guthaben abgezogen wird wenn eine 2 zur 0 wird. Es steht also immer mindestens genug Guthaben zu verfügung um alle 2en zu 0en zu ändern. Die Gesamtkosten um n mal zu inkrementieren sind damit abschätzbar durch:

$$T(n) = \sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} a_i \tag{2}$$

Bei jeder Inkrementierung wird aber höchstens ein Trit von von 1 auf 2 geändert oder ein Trit von 0 auf 1. Das heißt $\forall i \leq n : a_i \leq 2$. Für die Gesamtkosten ergibt sich somit $T(n) \leq \sum_{i=1}^{n} 2 = 2n \in \mathcal{O}(n)$.

b)

Die Addition von 11_2 is Äquivalent zu erst 10_2 und dann 1_2 addieren. Die Amorisierten Kosten um 10_2 zu addieren sind höchstens gleich hoch wie die um 1_2 zu addieren (man ignoriert einfach die letzte stell und wendet genau das gleiche Verfahren an). Die Kosten das i-te mal 3 zu addieren sind also $a_{i,3} = a_{i,2} + a_{i,1} = 2a_{i,1} = 4$ wobei $a_{i,1}$ die kosten sind um das i-te mal 1 zu addieren.

Lösung zu Aufgabe 4

a)

Wähle man wähle $\Phi(i)$ als:

$$\Phi(i) = c_i - \lfloor \frac{|A_i|}{2} \rfloor \tag{3}$$

c_i ist die länge des dynamischen Arrays am **anfang** der i-ten Einfügeoperation. $|A_i|$ ist die länge von A am **ende** der i-ten Einfügeoperation. Da initial $c_i = 0$ und $|A_i| = 1$ gilt $\Phi(0) = 0$. Sei

$\Delta\Phi(i) = \Phi(i) - \Phi(i-1)$, dann gilt für die amorisierten Kosten $a_i = t_i + \Delta\Phi(i)$. Wenn man nur nacheinander einfügt bleibt $c_i \geq \lfloor \frac{|A_i|}{2} \rfloor$ und $\Phi(i)$ ist somit immer positivy. Daraus folgt, dass wir die tatsächlichen Kosten abschätzen können mit:

$$T(n) = \sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} a_i \tag{4}$$

Angenommen die i-te Operation muss k_i Elemente kopieren. Dann sind die tatsächlihen Kosten $t_i = k_i + 1$ weil auch noch das neue Element geschrieben werden muss. Wenn aber ein neues, doppelt so großes Array angelegt wird dann wird auch das Potential um k_i kleiner, da dann $c_i = \frac{|A_i|}{2} \rfloor$ (die Elemente passen jetzt alle in die linke Hälfte des Array). Für Das Potential gilt:

$$\begin{aligned} &\Phi(i) \leq \Phi(i-1) + 1 - k_i \\ \Leftrightarrow &\Delta\Phi(i) \leq 1 - k_i \end{aligned} \tag{5}$$

Die amorisiertn Kosten sind damit

$$\begin{aligned} a_i &= t_i + \Delta\Phi(i) \\ &\leq k_i + 1 + 1 - k_i \\ &= 2 \end{aligned} \tag{6}$$

und für die Gesamtkosten gilt damit

$$T(n) \leq \sum_{i=1}^{n} a_i \leq 2n \in \mathcal{O}(n) \tag{7}$$

b)

Wenn 2^k viele Elemente eingefügt wurden, dann braucht das Einfügen $e = 2^k c_c + c_e$ Kosten, wobei c_c die Kosten zum kopieren und c_e die zum einfügen sind. Wnn man dannach gleich wieder löscht, dann sind die Kosten $l = 2^k c_c + c_d$ wobei c_d die kosten zum Löschen sind. Wenn t_i die kosten für das i-te Einfügen sind, dann sind die Kosten genau dann maximal wenn

$$T_n(k) = \sum_{i=1}^{2^k} t_i + \frac{(n - 2^k)}{2}(e + l) \tag{8}$$

maximal ist. Wenn man k findet dann ist die Sequenz 2^k mal append gefolgt von $\frac{(n-2^k)}{2}$ mal append und removeLast abgewechselt. [1]

[1] Irgendwas muss ich übersehen haben, das es einfacher macht