Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

# Übungsblatt 02

## Algorithmen I - Sommersemester 2024

**Abgabe im ILIAS bis 03.05.2024, 18:00 Uhr**

Die Abgabe erfolgt alleine oder zu zweit als eine PDF-Datei über das Übungsmodul in der Gruppe eures Tutoriums im ILIAS. Beschriftet die Abgabe deutlich mit Matrikelnummer(n) und Name(n). Bei Abgaben zu zweit reicht es, wenn eine Person die Abgabe im ILIAS hochlädt.

- Achtet bei handschriftlichen Abgaben auf Lesbarkeit.
- Achtet darauf, ob Algorithmen in Worten oder Pseudocode beschrieben werden sollen.
- Es werden immer asymptotisch möglichst effiziente Algorithmen erwartet, wenn nicht anders angegeben.
- Wenn Korrektheits- oder Laufzeitanalysen gefordert sind, behandelt diese separat von der Algorithmenbeschreibung.

**Gesamtpunkte**: 20

Hinweis: Schaut euch gerne die Algo-Code Aufgabe zu dynamischen Arrays an. Sie eignet sich besonders gut, um sich mehr mit amortisierten Analysen auseinanderzusetzen.
`https://algo-code.iti.kit.edu/array/dynamic-array/`

## Aufgabe 1 - Knoten lösen (5 Punkte)

In dieser Aufgabe wollen wir das asymptotische Verhalten von Laufzeitfunktionen mittels ihres Rekursionsbaums bestimmen. Analysiere dafür den Rekursionsbaum, indem du folgende Fragen für jede rekursive Funktion beantwortest:

- Wie viele Lagen hat der Rekursionsbaum?
- Wie viele Knoten befinden sich auf Lage $i$?
- Wie groß ist der Funktionsparameter auf Lage $i$?
- Wie viel Zeit kostet ein Knoten auf Lage $i$?

Stelle mithilfe deiner Antworten eine Summe auf, die die Gesamtlaufzeit der Funktion angibt. Bestimme anhand der Summe das asymptotisch Verhalten der Funktion.

a) $T_1(n) = \begin{cases} 1 & \text{falls } n = 1, \\ 27 \cdot T_1(\frac{n}{3}) + 3 \cdot n^3 & \text{sonst.} \end{cases}$ (1.5 Punkte)

b) $T_2(n) = \begin{cases} 3 & \text{falls } n = 1, \\ 32 \cdot T_2(\frac{n}{4}) + n \cdot (\log(n) + 1) & \text{sonst.} \end{cases}$ (2 Punkte)

c) $T_3(n) = \begin{cases} 1 & \text{falls } n = 1, \\ 11 \cdot T_3(\frac{n}{7}) + n \cdot \sqrt{n} & \text{sonst.} \end{cases}$ (1.5 Punkte)

## Lösung 1

a)
- Auf jeder Lage dritteln wir die Größe der Knoten, daher haben wir insgesamt $\log_3(n)$ Lagen.
- Auf jeder Lage ver-27-fachen wir die Knoten der Lage davor, auf Lage $i$ haben wir daher $27^i$ Knoten.
- Da wir die Knotengröße auf jeder Lage dritteln, erhalten wir auf Lage $i$ eine Größe von $\frac{n}{3^i}$.
- Zusätzlich hat ein Knoten mit Größe $k$ immer Kosten von $3 \cdot k^3$

Damit haben wir eine Laufzeit von

$$\begin{aligned} \sum_{i=0}^{\log_3(n)} 27^i \cdot 3 \cdot \left(\frac{n}{3^i}\right)^3 &= 3 \cdot n^3 \cdot \sum_{i=0}^{\log_3(n)} \frac{27^i}{(3^3)^i} \\ &= 3 \cdot n^3 \cdot \sum_{i=0}^{\log_3(n)} 1 \\ &= 3 \cdot n^3 \cdot (\log_3(n) + 1) \in \Theta(n^3 \cdot \log(n)) \end{aligned}$$

b) Analog erhalten wir
- $\log_4(n)$ Lagen.
- $32^i$ Knoten auf Lage $i$.
- Knotengröße $\frac{n}{4^i}$ auf Lage $i$.
- Zusätzliche Kosten pro Knoten mit Größe $k$ von $k \cdot (\log(k) + 1)$

Damit haben wir eine Laufzeit von

$$\begin{aligned} \sum_{i=0}^{\log_4(n)} 32^i \cdot \frac{n}{4^i} \cdot \left(\log\left(\frac{n}{4^i}\right) + 1\right) &= n \cdot \sum_{i=0}^{\log_4(n)} \frac{32^i}{4^i} \cdot \left(\log\left(\frac{n}{4^i}\right) + 1\right) \\ &= n \cdot \sum_{i=0}^{\log_4(n)} 8^i \cdot \left(\log\left(\frac{n}{4^i}\right) + 1\right) \end{aligned}$$

Wir können die Summe hier asymptotisch durch ihren größten Summanden abschätzen

$$\begin{aligned}
&\in n \cdot \Theta\left(8^{\log_4(n)} \cdot \left(\log\left(\frac{n}{4^{\log_4(n)}}\right) + 1\right)\right) \\
&= n \cdot \Theta\left(8^{\frac{\log_8(n)}{\log_8(4)}} \cdot \left(\log\left(\frac{n}{n}\right) + 1\right)\right) \\
&= n \cdot \Theta(8^{\frac{3}{2} \cdot \log_8(n)} \cdot (\log(1) + 1)) \\
&= \Theta(n^{\frac{5}{2}})
\end{aligned}$$

c) Analog erhalten wir

- $\log_7(n)$ Lagen.
- $11^i$ Knoten auf Lage $i$.
- Knotengröße $\frac{n}{7^i}$ auf Lage $i$.
- Zusätzliche Kosten pro Knoten mit Größe $k$ von $k \cdot \sqrt{k}$

Damit haben wir eine Laufzeit von

$$\begin{aligned}
\sum_{i=0}^{\log_7(n)} 11^i \cdot \frac{n}{7^i} \cdot \sqrt{\frac{n}{7^i}} &= n \cdot \sqrt{n} \cdot \sum_{i=0}^{\log_7(n)} \frac{11^i}{7^i} \cdot \sqrt{\frac{1}{7^i}} \\
&= n \cdot \sqrt{n} \cdot \sum_{i=0}^{\log_7(n)} \left(\frac{11}{7}\right)^i \cdot \left(\sqrt{\frac{1}{7}}\right)^i \\
&= n \cdot \sqrt{n} \cdot \sum_{i=0}^{\log_7(n)} \left(\sqrt{\frac{121}{49}}\right)^i \cdot \left(\sqrt{\frac{1}{7}}\right)^i \\
&= n \cdot \sqrt{n} \cdot \sum_{i=0}^{\log_7(n)} \left(\sqrt{\frac{121}{49}} \cdot \frac{1}{7}\right)^i
\end{aligned}$$

Wir nutzen, dass die geometrische Reihe gegen eine Konstante konvergiert und erhalten:

$$\in \Theta(n \cdot \sqrt{n})$$

## Aufgabe 2 - Angriff der Klonbiber (10 Punkte)

Um seinen Geheimplan in die Tat umsetzen zu können, benötigt Dr. Meta eine große Menge an Bibern. Leider steht ihm zur Zeit nur eine kleine Zahl der flauschigen Tiere zur Verfügung. Dr. Meta ist es allerdings gelungen, eine Klonmaschine zu entwickeln, die es ihm erlaubt, seine Biber zu vervielfachen. Aufgrund technischer Schwierigkeiten kann die Maschine aktuell jeden Biber nur ein Mal klonen, daher unterscheiden wir zwischen originalen (das heißt nicht geklonten) Bibern und den geklonten Exemplaren. Wir bezeichnen die Anzahl der originalen Biber, die sich zu einem bestimmten Zeitpunkt

in der Maschine befinden, mit $\alpha$ und analog die Anzahl der geklonten Biber mit $\beta$. Die Maschine unterstützt die folgenden Operationen (die Kosten hängen von dem Zustand der Maschine vor Aufruf der Operation ab):

- LOAD: Lädt einen bereits existierenden originalen Biber in die Maschine, benötigte Energie 1.
- CLONE: Klont die Biber in der Maschine, indem jeder originale Biber durch zwei geklonte ersetzt wird. Bereits geklonte Biber werden nicht geklont, benötigte Energie $2 \cdot \alpha$.
- EMPTY: Entfernt alle Biber aus der Maschine, benötigte Energie $4 \cdot \alpha + \beta$.

Da die Stromkosten in dem letzten Jahr stark gestiegen sind, macht sich Dr. Meta allerdings Sorgen wegen des hohen Energieverbrauchs der CLONE-Operation.

a) Gib für die folgenden beiden Folgen von Operationen die asymptotischen Gesamtkosten abhängig von $k$ an:

(i) $k$·LOAD
CLONE
EMPTY

(1 Punkt)

(ii) $k$·LOAD
CLONE
$2 \cdot k$·LOAD
CLONE
EMPTY

(1 Punkt)

b) Zeige, dass der amortisierte Energieverbrauch für eine beliebige Abfolge von Operationen auf der zu Beginn leeren Maschine bei $\Theta(1)$ pro Operation liegt. Nutze dafür Aggregation, Charging oder die Kontomethode. (2 Punkte)

Dr. Meta ist zwar höchst erfreut über deine Ergebnisse, ist allerding mit den von dir genutzten Methoden nicht zufrieden. Er ist nicht von der Überzeugung abzubringen, dass nur mit der Potentialmethode das Potential der Maschine voll ausgeschöpft werden kann. Du sollst daher den Beweis mit dieser Methode wiederholen. Die hilfreichen Biber haben bereits etwas an dem Beweis herumgetüftelt. Dabei haben sie allerdings einiges durcheinander gebracht.

c) Bringe die untenstehenden Beweischnipsel in die richtige Reihenfolge. Außerdem haben die Bibern wohl einige Teile eines anderen Beweises mit den Schnipseln vermischt. Entferne die nicht benötigten Abschnitte. (3 Punkte)

Hinweis: Die Musterlösung enthält 14 Aussagen.

(1) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 0 - 2 \cdot \alpha = -2 \cdot \alpha$.

(2) Wir nutzen die Potentialfunktion $\Phi(M) = \beta - \alpha$. Da die Anzahl der originalen bzw. geklonten Biber in der Maschine nicht-negativ ist, ist auch die Potentialfunktion gültig.

(3) Somit liegen die amortisierten Kosten bei $1 + 4 = 5 \in \Theta(1)$.

(4) Somit liegen die amortisierten Kosten bei $4 \cdot \alpha + \beta + \alpha - \beta = 5 \cdot \alpha \in \Theta(\alpha)$.

(5) Damit zeigen wir, dass die amortisierten Kosten jeder Operation konstant sind, indem wir die tatsächlichen Kosten einer Operation zusammen mit ihrer Potentialänderung betrachten.

(6) Wir laden einen originalen Biber ein und haben damit eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = (4 \cdot (\alpha + 1) + \beta) - (4 \cdot \alpha + \beta) = 4$.

(7) Wir nutzen die Potentialfunktion $\Phi(M) = 2 \cdot \alpha$. Da die Anzahl der originalen Biber in der Maschine nicht-negativ ist, ist auch die Potentialfunktion gültig.

(8) Somit liegen die amortisierten Kosten bei $2 \cdot \alpha - 2 \cdot \alpha = 0 \in \Theta(1)$.

(9) Durch diese Operation werden alle $\alpha$ originalen Biber und alle $\beta$ geklonten Biber aus der Maschine entfernt.

(10) Somit liegen die amortisierten Kosten bei $2 \cdot \alpha - 2 \cdot \alpha = 0 \in \Theta(1)$.

(11) CLEAN:

(12) Wir laden einen originalen Biber ein und haben damit eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 1$.

(13) Somit liegen die amortisierten Kosten bei $4 \cdot \alpha + \beta - (4 \cdot \alpha + \beta) = 0 \in \Theta(1)$.

(14) LOAD:

(15) EMPTY:

(16) Wir nutzen die Potentialfunktion $\Phi(M) = 4 \cdot \alpha + \beta$. Da die Anzahl der originalen bzw. geklonten Biber in der Maschine nicht-negativ ist, ist auch die Potentialfunktion gültig.

(17) Für die Maschine $M$ bezeichen wir mit $M_{\text{vor}}$ den Zustand vor und mit $M_{\text{nach}}$ den Zustand nach einer Operation.

(18) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = (0 + (2 \cdot \alpha + \beta)) - (4 \cdot \alpha + \beta) = -2 \cdot \alpha$.

(19) Durch diese Operation ersetzen wir die $\alpha$ originalen Biber durch $2 \cdot \alpha$ geklonte Biber.

(20) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 0 - (4 \cdot \alpha + \beta) = -(4 \cdot \alpha + \beta)$.

(21) CLONE:

(22) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 0 - (\beta - \alpha) = \alpha - \beta$.

Während du den Beweis zusammengesetzt hast, hat Dr. Meta an der Maschine herumgeschraubt. Nun erzeugt CLONE aus jedem originalen Biber drei geklonte Biber. Allerdings entfernt EMPTY nur noch die geklonten Biber und $\frac{4}{5}$ der originalen Biber aus der Machine. Somit bleiben $\frac{1}{5}$ der originalen Biber nach der Ausführung der Operation in der Maschine[1]. LOAD und die Energiekosten der Operationen bleiben unverändert.

d) Wende nun das Gelernte an und zeige mit der Potentialmethode, dass der amortisierte Energieverbrauch jeder Operation in der von uns betrachteten Abfolge noch immer konstant ist. (3 Punkte)

Hinweis: Falls du Probleme hast eine Potentialfunktion zu finden: Du kannst die Funktion aus b) leicht abwandeln. Überlege dir dafür, was schief geht, wenn du die alte Potentialfunktion nutzt und passe diese dann an, um den Fehler zu beheben.

## Lösung 2

a) (i) $k$-maliges Anwenden von LOAD hat Kosten $k$, danach haben wir $k$ originale und 0 geklonte Biber in der Maschine, daher hat CLONE Kosten $2 \cdot k$ und wir erhalten 0 originale und $2 \cdot k$ gelonte Biber, womit EMPTY Kosten $2 \cdot k$ hat und wir erhalten insgesamte Kosten $\Theta(k)$.

(ii) $k$-maliges Anwenden von LOAD hat Kosten $k$, danach haben wir $k$ originale und 0 geklonte Biber in der Maschine, daher hat CLONE Kosten $2 \cdot k$ und wir erhalten 0 originale und $2 \cdot k$ gelonte Biber. Nach $2 \cdot k$-maligem Anwenden von LOAD und Kosten $k$, haben wir $2 \cdot k$ originale und $2 \cdot k$ gelonte Biber in der Maschine. Daher hat CLONE Kosten $4 \cdot k$ und hinterlässt die Maschine mit 0 originale und $6 \cdot k$ gelonten Biber. Somit hat EMPTY Kosten $6 \cdot k$ und wir erhalten insgesamte Kosten $\Theta(k)$

b) • Wir beginnen mit der Kontomethode:

Hier liegen die Kosten für LOAD bei einem Token, die für CLONE bei $2 \cdot \alpha$ und für EMPTY bei $4 \cdot \alpha + \beta$ Token. Wir zahlen nun für LOAD vier weitere Token ein, wodurch die Kosten weiterhin konstant bleiben. Wir werden hiervon später zwei Token für CLONE und zwei für EMPTY nutzen, um die Kosten auf 0 Token zu senken.

Wir betrachten nur Folgen die eine oder keine EMPTY-Operation enthalten und mit dieser enden. Alle anderen Folgen lassen sich in mehrere solche Folgen, die je mit einer leeren Maschine beginnen, unterteilen.

Vor jedem Aufruf von CLONE haben wir mindestens $4 \cdot \alpha$ mehr Token auf dem Konto als nach dem letzten Klonen. Dies gilt, da nach CLONE alle originalen Biber zu geklonten wurden und somit $\alpha$ LOAD Operation, die je vier

[1] Einfachheitshalber nehmen wir an, dass EMPTY nur aufgerufen wird, falls die Division ohne Rest abläuft.

überschüssigen Token einzahlen, nötig sind, um $\alpha$ originale Biber in der Maschine zu haben. CLONE verwendet nun $2 \cdot \alpha$ dieser Token, da sich $\alpha$ originale Biber in der Maschine befinden. Nach dem Klonen verbleiben auf dem Konto noch $2 \cdot \alpha$ der neuen Token, was einem Token pro neu erzeugten, geklonten Biber entspricht.

Vor der EMPTY-Operation haben wir $\alpha$ originale und $\beta$ geklonte Biber in der Maschine und die Kosten von EMPTY liegen daher bei $4 \cdot \alpha + \beta$. Wir nutzen nun die $\beta$ von CLONE übriggelassenen Token für die geklonten Biber. Außerdem nutzen wir die vier durch LOAD eingezahlten Token für jeden originalen Biber. Beachte, dass alle originalen Biber, die sich aktuell in der Maschine befinden, nach der letzten CLONE-Operation durch LOAD eingeladen wurden. Daher wurden die zwei für CLONE reservierten Token nicht genutzt und wir können diese Token zusätzlich für EMPTY verwenden.

Somit wird unser Konto nie negativ und daher hat jede Operation amortisiert konstante Kosten.

- Als nächstes nutzen wir Aggregation: Wie oben erklärt haben wir die folgenden Kosten:
  - LOAD: 1
  - CLONE: $2 \cdot \alpha$
  - EMPTY: $4 \cdot \alpha + \beta$

  Wir betrachten erneut nur Folgen die eine oder keine EMPTY-Operation enthalten und mit dieser enden. Alle anderen Folgen lassen sich in mehrere solche Folgen, die je mit einer leeren Maschine beginnen, unterteilen.

  Somit besteht unsere Folge beliebig vielen Teilfolgen, die $k_i$ LOAD- gefolgt von einer CLONE-Operation beinhalten. Hierbei ist $i$ die Nummer der Teilfolge. Die letzte Teilfolge, kann auch kein CLONE enthalten. Nach der letzten Teilfolge wird EMPTY aufgerufen. Dabei hat jedes LOAD Kosten von 1 und jedes Clone Kosten von $2 \cdot k_i$. Dies gilt, da nach jedem CLONE keine originalen Biber verbleiben und somit vor dem nächsten CLONE nur die neueingeladenen originalen Biber in der Maschine sind. Das finale EMPTY, wenn vorhanden, hat Kosten von weniger als $\sum_i 4 \cdot {k_i}^2$, da aus jedem eingeladenen Biber zwei geklonte werden, die je Kosten 1 haben. Für jeden originalen Biber haben wir Kosten 4. Für die originalen Biber sind die Kosten höher und daher finden wir hier diese obere Schranke.

  Somit haben wir insgesamt maximal $\sum_i (k_i + 1) + 1$ Operationen. Wir summieren nun alle Kosten auf und teilen durch die Anzahl der Operationen, um

[2] $\sum_i$ bezeichnet hier die Summe über alle $i$, die eine Teilfolge nummerieren.

die durchschnittlichen Kosten zu erhalten.

$$\begin{aligned}\frac{\sum_i (k_i \cdot 1 + 2 \cdot k_i) + \sum_i (4 \cdot k_i)}{\sum_i (k_i + 1) + 1} &= \frac{\sum_i (3 \cdot k_i) + \sum_i (4 \cdot k_i)}{\sum_i (k_i + 1) + 1} \\ &= 3 \cdot \frac{\sum_i k_i}{\sum_i (k_i + 1) + 1} + 4 \cdot \frac{\sum_i k_i}{\sum_i (k_i + 1) + 1} \\ &\leq 3 + 4 \in \Theta(1)\end{aligned}$$

Hier ist direkt ersichtlich, dass die durchschnittlichen, und somit auch die amortisierten, Kosten konstant sind. Fallen die letzte CLONE- bzw. die EMPTY-Operation weg, ist die Gesamtzahl der Operationen um zwei kleiner. Dies ändert bis auf einen konstanten Faktor nichts am Ergebnis

- Als letztes nutzen wir Charging: Wie oben erklärt haben wir die folgenden Kosten:
  - LOAD: 1
  - CLONE: $2 \cdot \alpha$
  - EMPTY: $4 \cdot \alpha + \beta$

  Dies entspricht der Anzahl Token jeder Operation. Wir verschieben nun Token von CLONE und EMPTY auf die LOAD-Operationen, sodass jede Operation konstant viele Token hat. Verschiebe dafür zwei Token von CLONE auf jedes LOAD zwischen dem vorherigen und dem aktuell betrachteten CLONE. Da CLONE alle originalen Biber verwendet, entspricht die Anzahl der originalen Biber in der Maschine bei Ausführung von CLONE der Anzahl der seit dem letzten CLONE durch LOAD eingefügten Biber. Verschiebe außerdem zwei Token von EMPTY auf jedes LOAD, auf das ein CLONE folgt, und vier auf die anderen LOAD. Die vier Token entsprechen hierbei den Kosten von EMPTY für den von LOAD eingeladnen originalen Biber und die zwei Token den Kosten von eins pro geklontem Biber, der aus dem originalen Biber entstanden ist.

  Die beiden Operationen haben somit null Token. LOAD hat dann $1+0+4 = 5$ bzw. $1 + 2 + 2 = 5$ Token.

  Da jede Operation nur konstant viele Token zugeteilt bekommen hat und die Gesamtzahl der Token gleich geblieben ist, sind die amortisierte Kosten einer Operation nach Charging konstant.

c) Die korrekte Reihenfolge ist (Die ersten beiden Aussagen können vertauscht werden. Die Operationen können in beliebiger Reihenfolge analysiert werden):

(17) Für die Maschine $M$ bezeichen wir mit $M_{\text{vor}}$ den Zustand vor und mit $M_{\text{nach}}$ den Zustand nach einer Operation.

(16) Wir nutzen die Potentialfunktion $\Phi(M) = 4 \cdot \alpha + \beta$. Da die Anzahl der originalen bzw. geklonten Biber in der Maschine nicht-negativ ist, ist auch die Potentialfunktion gültig.

(5) Damit zeigen wir, dass die amortisierten Kosten jeder Operation konstant sind, indem wir die tatsächlichen Kosten einer Operation zusammen mit ihrer Potentialänderung betrachten.

(14) LOAD:

(6) Wir laden einen originalen Biber ein und haben damit eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = (4 \cdot (\alpha + 1) + \beta) - (4 \cdot \alpha + \beta) = 4$.

(3) Somit liegen die amortisierten Kosten bei $1 + 4 = 5 \in \Theta(1)$.

(21) CLONE:

(19) Durch diese Operation ersetzen wir die $\alpha$ originale Biber durch $2 \cdot \alpha$ geklonte Biber.

(18) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = (0 + (2 \cdot \alpha + \beta)) - (4 \cdot \alpha + \beta) = -2 \cdot \alpha$.

(8) Somit liegen die amortisierten Kosten bei $2 \cdot \alpha - 2 \cdot \alpha = 0 \in \Theta(1)$.

(15) EMPTY:

(9) Durch diese Operation werden alle $\alpha$ originalen Biber und alle $\beta$ geklonte Biber aus der Maschine entfernt.

(20) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 0 - (4 \cdot \alpha + \beta) = -(4 \cdot \alpha + \beta)$.

(13) Somit liegen die amortisierten Kosten bei $4 \cdot \alpha + \beta - (4 \cdot \alpha + \beta) = 0 \in \Theta(1)$.

Nicht benötigt werden:

(11) CLEAN:

(2) Wir nutzen die Potentialfunktion $\Phi(M) = \beta - \alpha$. Da die Anzahl der originalen bzw. geklonten Biber in der Maschine nicht-negativ ist, ist auch die Potentialfunktion gültig.

(7) Wir nutzen die Potentialfunktion $\Phi(M) = 2 \cdot \alpha$. Da die Anzahl der originalen Biber in der Maschine nicht-negativ ist, ist auch die Potentialfunktion gültig.

(12) Wir laden einen originalen Biber ein und haben damit eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 1$.

(22) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 0 - (\beta - \alpha) = \alpha - \beta$.

(4) Somit liegen die amortisierten Kosten bei $4 \cdot \alpha + \beta + \alpha - \beta = 5 \cdot \alpha \in \Theta(\alpha)$.

(1) Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = 0 - 2 \cdot \alpha = -2 \cdot \alpha$.

(10) Somit liegen die amortisierten Kosten bei $2 \cdot \alpha - 2 \cdot \alpha = 0 \in \Theta(1)$.

d) Für die Maschine M bezeichen wir erneut mit $M_{\text{vor}}$ den Zustand vor und mit $M_{\text{nach}}$ den Zustand nach einer Operation.

Wir nutzen die Potentialfunktion $\Phi(M) = 5 \cdot \alpha + \beta$. Damit zeigen wir, dass die amortisierten Kosten jeder Operation konstant sind, indem wir die tatsächlichen Kosten einer Operation zusammen mit ihrer Potentialänderung betrachten.

LOAD

Wir laden einen originalen Biber ein und haben damit eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = (5 \cdot (\alpha+1) + \beta) - (5 \cdot \alpha + \beta) = 5$. Somit liegen die amortisierten Kosten bei $1 + 5 = 6 \in \Theta(1)$.

CLONE

Durch diese Operation ersetzen wir die $\alpha$ originale Biber durch $3 \cdot \alpha$ geklonte Biber. Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = (0 + (3 \cdot \alpha + \beta)) - (5 \cdot \alpha + \beta) = -2 \cdot \alpha$. Somit liegen die amortisierten Kosten bei $2 \cdot \alpha - 2 \cdot \alpha = 0 \in \Theta(1)$.

EMPTY

Durch diese Operation werden alle $\beta$ geklonte Biber und $\frac{4}{5}$ der originalen Biber aus der Maschine entfernt. Dadurch haben wir eine Potentialänderung von $\Phi(M_{\text{nach}}) - \Phi(M_{\text{vor}}) = (5 \cdot \frac{1}{5} \cdot \alpha + 0) - (5 \cdot \alpha + \beta) = -(4 \cdot \alpha + \beta)$. Somit liegen die amortisierten Kosten bei $4 \cdot \alpha + \beta - (4 \cdot \alpha + \beta) = 0 \in \Theta(1)$.

## Aufgabe 3 - Quite sum problem (5 Punkte)

Wir wollen in dieser Aufgabe zwei Arrays $A$ und $B$ der Länge $n$ von Ganzzahlen betrachten.

Für ein halboffenes Intervall $[i, j)$ mit $i \in \{0, \ldots, n-1\}$ und $j \in \{0, \ldots, n\}$, und Array $X \in \{A, B\}$ bezeichnen wir mit $\text{sum}_X(i, j) = \sum_{k=i}^{j-1} X[k]$ die Summe der Einträge zwischen $i$ und $j$. Leere Summen sind gleich 0, also gilt z.B. auch $\text{sum}_X(0, 0) = 0$. In dieser Aufgabe gelten für alle Vorkommen von $i, j$ und $X$ die obigen Einschränkungen.

Wir wollen nun bestimmen, für welche $i, j$ die Summen $\text{sum}_A(i, j)$ und $\text{sum}_B(i, j)$ gleich sind. Unser Algorithmus soll den größten solchen Bereich finden. Dafür bezeichnen wir mit $j - i$ die Länge des Bereichs zwischen $i$ und $j$.

a) Betrachte die beiden folgenden Arrays und bestimme das längsten Intervall $[i, j)$ mit $\text{sum}_A(i, j) = \text{sum}_B(i, j)$. Gebe hierfür die Grenzen $i, j$ und die Länge des Intervalls an. (1 Punkt)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 2 | 5 | −3 | 5 | 0 | 6 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| −1 | 3 | 4 | 0 | 1 | 2 | 3 |

Die beiden Arrays mit Indizes.

Um das Problem zu lösen, wollen wir Präfixsummen als Hilfsmittel verwenden. Eine Präfixsumme ist die Summe $\text{sum}_X(0, j)$ für das Intervall $[0, j)$ und Array $X$.

b) Beschreibe in Worten einen Algorithmus, der in Zeit $\mathcal{O}(n)$ für alle Indizes die jeweilige Präfixsumme eines Arrays berechnet. Begründe warum dein Algorithmus die geforderte Laufzeit erreicht. (1.5 Punkte)

c) Nach Ausführung deines Algorithmus aus c) stehen dir die Präfixsummen eines Arrays für beliebige Intervalle $[0, j)$ beginnend bei 0 zur Verfügung. Wie kannst du mithilfe dieser Präfixsummen die Summe $\text{sum}_X(i, j)$ berechnen? (1 Punkt)

d) Beschreibe nun einen Algorithmus der in Zeit $\mathcal{O}(n^2)$ das längste Intervall mit gleicher Summe findet. (1.5 Punkte)

## Lösung 3

a) Wir finden ein Intervall der Länge 4 mit den Grenzen $i = 2$ und $j = 6$.

b) Wir gehen über das Array und summieren die bisher gesehenen Einträge auf, indem wir den aktuellen Eintrag auf das bisherige Zwischenergebnis addieren. In jedem Schritt erhalten wir somit die aktuelle Zwischensumme und speichern diese ab.

   Dies erfolgt in Linearzeit, da wir nur einmal über das Array gehen und jede Addition konstante Zeit benötigt.

c) Es gilt $\text{sum}_X(i, j) = \text{sum}_X(0, j) - \text{sum}_X(0, i)$.

d) Wir nutzen zuerst den Algorithmus aus c), um alle Präfixsummen der Arrays $A$ und $B$ zu berechnen. Anschließend können wir für jedes Array und jedes Intervall $[i, j)$ die Summe $\text{sum}_X(i, j)$ berechnen. Als nächstes verwerfen wir alle Intervalle mit $\text{sum}_A(i, j) \neq \text{sum}_B(i, j)$. Mit einer finalen Iteration über die Intervalle können wir das Intervall mit der maximalen Länge finden.

   Laufzeitanalyse (nicht gefordert):
   Der Algortihmus aus c) läuft in Linearzeit. Anschließend betrachten wir quadratisch viele Intervalle. Wir können den Faktor zwei, der durch die zwei Arrays entsteht, asymptotisch ignorieren. So wohl die Berechnung der Summe, als auch die Entscheidung, ob das Intervall verworfen wird, können in konstanter Zeit pro Intervall getroffen werden. Das Finden des Maximums erfolgt durch einen Durchlauf über alle quadratisch vielen Intervalle. Somit ist die Gesamtlaufzeit quadratisch.

## Aufgabe 4 - Zum Knobeln (0 Punkte + 3 Bonus)

Wir wollen jetzt einen Linearzeitalgorithmus für das Problem aus Aufgabe 3 finden. Im Folgenden steht dir eine magische Blackbox $M$ zur Verfügung, die eine Folge von

Werten speichern kann. Du kannst in jeweils konstanter Zeit Elemente in diese Blackbox einfügen und überprüfen, ob ein gegebenes Element in der Blackbox vorhanden ist.[3]

a*) In welchem Verhältnis stehen $\text{sum}_A(0,i) - \text{sum}_B(0,i)$ und $\text{sum}_A(0,j) - \text{sum}_B(0,j)$ zu einander, wenn:

- $\text{sum}_A(i,j) = \text{sum}_B(i,j)$?
- $\text{sum}_A(i,j) \neq \text{sum}_B(i,j)$?

(1 Punkt)

b*) Gebe einen Linearzeit Algorithmus an, der das längste zusammenhängende Intervall $[i,j)$ mit $\text{sum}_A(i,j) = \text{sum}_B(i,j)$ bestimmt. Beschreibe diesen Algorithmus in Worten. (2 Punkte)

## Lösung 4

Anmerkung: Wir speichern $(key, value)$ pairs in der Blackbox. Dafür definieren wir zwei Tupel $(a_1,b_1),(a_2,b_2)$ als gleich, wenn $a_1 = a_2$ ($b_1, b_2$ sind für die Gleichheit nicht von Bedeutung). Wenn wir überprüfen, ob ein Element $(a,b_1)$ in der Blackbox ist, kann uns die Blackbox ein Tupel $(a,b_2)$ zurückgeben, wobei $b_1 \neq b_2$. Dies erlaubt es uns, die Blackbox wie eine Hashmap zu verwenden obwohl die ursprüngliche Lösung dies nicht vorgesehen hat.

a) In Bereichen mit gleicher Summe sind die beiden Differenzen gleich. In Bereichen mit verschiedener Summe sind die beiden Differenzen verschieden.

b) Wir gehen von links nach rechts über die beiden Arrays und bestimmen in jedem Schritt die Differenz $x = \text{sum}_A(0,k) - \text{sum}_B(0,k)$, für Schrittzahl $k \in \{0,\dots,n\}$. Dabei prüfen wir für jedes $k$ ob $(x,k)$ bereits in $M$ ist. Ist dies nicht der Fall, fügen wir $(x,k)$ ein. In dem anderen Fall wird uns ein $(x,k')$ zurückgegeben und nach a) gilt dann $\text{sum}_A(k',k) = \text{sum}_B(k',k)$. Indem wir uns stets den größten bisher gefundenen Bereich merken, finden wir das Maximum.

Skizze der Laufzeitanalyse (Nicht gefordert):
Wir gehen einmal über das Array und führen dabei nur Operationen aus, die konstante Zeit benötigen. Wir können hierfür die Präfixsummen, wie in A3 gesehen berechnen.

Skizze des Korrektheitsbeweises (Nicht gefordert):
Es gilt für Intervall $[i,j)$:

$$\begin{aligned}
&\text{sum}_A(i,j) = \text{sum}_B(i,j)\\
&\Leftrightarrow \text{sum}_A(0,j) - \text{sum}_A(0,i) = \text{sum}_B(0,j) - \text{sum}_B(0,i)\\
&\Leftrightarrow \text{sum}_A(0,j) - \text{sum}_B(0,j) = \text{sum}_A(0,i) - \text{sum}_B(0,i)
\end{aligned}$$

[3] Klausurvorbereitung für den Rest des Semesters: Überlege dir bei Einführung einer neuen Datenstruktur, ob du mit dieser eine solche Blackbox implementieren kannst bzw. was die Laufzeiten der Operationen wären.

# Fachschaftsveranstaltungen für Erstis

Du interessierst dich für die Arbeit der Fachschaft und möchtest dich vielleicht gerne selbst engagieren? Schau einfach vorbei am **25. April um 19:00 Uhr** beim **Semesterauftakttreffen**. Hier zeigen wir dir, wie die Fachschaft organisiert ist, was ihre Aufgaben sind und wie du dich bei uns einbringen kannst.