Übungsblatt 3

Maurice Donner Jan Hubrich Adrian Müller

May 27, 2022

Aufgabe 1

a)
Es gibt insgesamt 4 operationen, die mehr als 1 kosten:

Operation	Kosten
$4^4 = 256$	256
$4^3 = 64$	64
$4^2 = 16$	16
$4^1 = 4$	4

Danach bleiben 252 Operationen mit Kosten 1 Die Gesamtkosten sind also $T(256) = 252 + 256 + 64 + 16 + 4 = 592$

b) Die Kosten T(n) können wie oben gezeigt in 2 Teile aufgeteilt werden, die nur von der Potenz m abhängen: Die viererpotenzen und die Operationen dazwischen.

Die Gesamtkosten ergeben sich also zu

$$T_1(m) + T_2(m) \quad \text{mit} \quad T_1(m) = \sum_{i=1}^{m} 4^i \quad \text{und} \quad T_2(m) = 4^m - m$$

Erweitert man die Summe in T_1, so erhält man

$$T_1 = \sum_{i=0}^{m} 4^i - 1$$

Nutzen wir nun die Formel für endliche Partialsummen einer geometrischen Reihe:

$$a_0 \sum_{k=0}^{n} q^k = a_0 \frac{q^{n+1} - 1}{q - 1}$$

So erhalten wir

$$T_1 = \frac{4^{m+1} - 4}{3}$$

Insgesamt erhalten wir also

$$T_1(m) + T_2(m) = \frac{4 \cdot 4^m - 4}{3} + \frac{3 \cdot 4^m - 3m}{3} = \frac{1}{3}\left(12 \cdot 4^m - 3m - 4\right)$$

Die kosten des Algorithmus betragen also

$$T(n) = 4 - \log(n) - 4/3$$

Aufgabe 2

1. Um einen unbeschränkten Stack mittels einer einfach verketteten liste zu bauen definieren wir wie in der Vorlesung das Item, und einen Pointer, der auf Items zeigt:

```
class Handle = Pointer to Item
class Item of Element
    e : Element
    next := this : Handle
```

Beim Erstellen eines neuen Elements, zeigt der Pointer `next` auf das Element selbst.
Den Stack kann man dann durch die Implementierung von `pushFront` (Element hinzufügen) und `popFront` (Element löschen) realisieren:

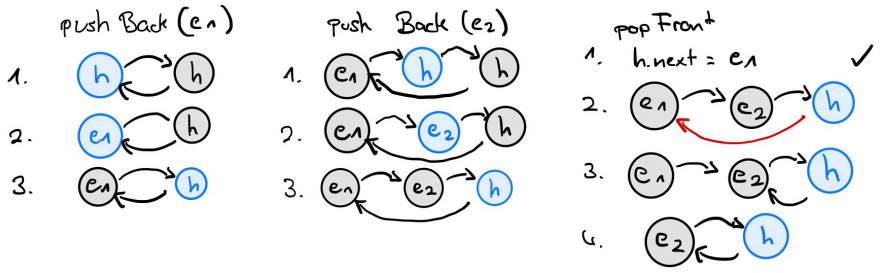
```
class Stack of Element
    h : new Item // Dummy-header
    proc pushFront(e : Element)
        tmp := h.next : Handle
        h.next := new Item
        h.next.e = e
        h.next.next = tmp
    proc popFront
        assert h.next != h
        tmp := h.next : Handle
        h.next = h.next.next
        delete tmp
```

Die neuen Elemente des Stacks werden zwischen `h` und dem nächst-neuem Element gespeichert.
Die Operationen verbrauchen konstante Zeit, da sie unabhängig von der Anzahl der in der Liste gespeicherten Elemente sind.

2. Anstatt wie beim Stack die Elemente *zwischen* `h` und dem neustem Element hinzuzufügen, können wir die Elemente ans "Ende" des Stacks mit `pushFront` schreiben:

```
class Queue of Element
    /* Like Stack */
    proc pushBack(e : Element)
        pushFront(h)
        h.e = e
        h = h.next
```

Auf diesem Weg werden zuerst die Elemente abgearbeitet, die sich bereits am längsten im Stack befinden. Addieren wir beispielsweise zwei Elemente auf die Queue, und löschen dann eins, läuft es wie folgt ab (das blaue Element ist jeweils das header-element):



3. Eine Implementierung von `popBack` ist in unserer einfach verketteten Liste nicht mehr möglich, da nun kein Handle mehr existiert, der direkt auf das letzte Element zeigt. Ein Zugriff ist also nur möglich, indem alle Einträge der Liste abgegangen werden, bis man auf das Element mit dem Handle stößt, der auf den Header zeigt (in der Skizze oben wäre das `e2`). In einer doppelt-verketteten Liste gibt es Handles in beide Richtungen, und ein Zugriff auf das letzte Element wäre somit kein Problem.