Algorithmen und Datenstrukturen
Tutorium III

Michael R. Jung

04. - 09. 05. 2016

1 Abstrakte Datentypen

2 Algorithmusanalyse
- Korrektheitsbeweis
- Laufzeitanalyse

Aufgabe 1

Implementieren Sie einen Stack und eine Queue auf Grundlage einer einfach verketteten Liste!

Folgende Funktionen seien für eine Liste L in natürlicher Weise implementiert:

- `int L.length();`
- `boolean L.isEmpty();`
- `void L.add(real value, int position);`
- `void L.addFirst(real value);`
- `void L.delete(int position);`
- `real L.getValue(int position);`

Stack

```
1  void push (real v){
2    L.addFirst(v);
3  }
4
5  real top(){
6  return getValue(1);
7  }
8
9  real pop(){
10   real h=getValue(1);
11   delete(1);
12   return h;
13 }
14
15 boolean isEmpty() { return L.isEmpty(); }
```

Stack

```
1  void enqueue (real v){
2    L.addFirst(v);
3  }
4
5  real head(){
6  return getValue(L.length());
7  }
8
9  real dequeue(){
10   real h=head();
11   delete(L.length());
12   return h;
13 }
14
15 boolean isEmpty() { return L.isEmpty(); }
```

```
1  function getM(A: array_of_int){
2  n:=|A|
3  if (n=1) then
4    return A[1];
5  endif
6  m:= ⌊ n/2 ⌋;
7  a:=getM(A[1...m]);
8  b:=getM(A[m+1...n]);
9  if (a>b) then return a;
10  else
11    return b;
12 endif;
13 }
```

Aufgabe 2

Was berechnet `getM(A: array_of_int)`? Beweisen sie ihre Behauptung! Welche Laufzeit benötigt dieser Algorithmus?

Behauptung: `getM(A: array_of_int)` berechnet $\max\{x \in A\}$.

Beweis via Induktion über $n := |A|$.

- Induktionsanfang n=1:

  `getM` gibt das einzige Element aus, welches maximal ist.

- Induktionsvoraussetzung:

  `getM` berechnet $\max\{x \in A\}$ für alle A mit $|A| < n$.

- Induktionsbehauptung:

  `getM` berechnet $\max\{x \in A\}$ für alle A mit $|A| = n$.

- Induktionsschluss:

  Betrachte $A[1 \dots n]$ für $n > 1$. Dann wird für $m := \lfloor n/2 \rfloor$ zunächst $\texttt{getM}(A[1 \dots m])$ und $\texttt{getM}(A[m+1 \dots n])$ berechnet. Nach Induktionsvoraussetzung sind also $a = \max\{x | x \in A[1 \dots m]\}$ und $b = \max\{x | x \in A[m+1 \dots n]\}$.

  1.Fall: $a > b$.

  Es gilt $\forall c \in A[1 \dots m] : a \geq c$ sowie $\forall c \in A[m+1 \dots n] : a > b \geq c$ und somit $\forall c \in A[1 \dots n] : a \geq c$. Dies ist auch der Rückgabewert.

  2.Fall: $a \leq b$.

  Analog. □

Seien k, k' geeignete Konstanten. Dann gilt:

$$T(1) = k$$

$$T(n) = k' + 2T(\frac{n}{2}) \qquad (i = 1)$$

Wiederholtes Einsetzen:

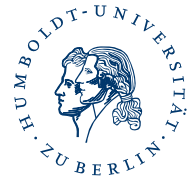
$$T(n) = k' + 2(k' + 2T(\frac{n}{4}))$$

$$= 3k' + 4T(\frac{n}{4}) \qquad (i = 2)$$

$$T(n) = 3k' + 4(k' + 2T(\frac{n}{8}))$$

$$= 7k' + 8T(\frac{n}{8}) \qquad (i = 3)$$

$$= (2^i - 1)k' + 2^i T(\frac{n}{2^i}) \qquad (Vermutung)$$



Abbruchbedingung einsetzen: $\frac{n}{2^i} = 1 \Leftrightarrow i = \log n$.

$\Rightarrow T(n) = (n-1)k' + nT(1) = (n-1)k' + nk$

Probe:

$$T(1) = (1-1)k' + 1k = k \checkmark$$

$$\begin{aligned} T(n) &= k' + 2T(\frac{n}{2}) \\ &= k' + 2((\frac{n}{2} - 1)k' + \frac{n}{2}k) \\ &= k' + (n-2)k' + nk = (n-1)k' + nk \checkmark \end{aligned}$$

Folglich hat der Algorithmus lineare Laufzeit.