RUB

Vorlesung Datenstrukturen

Listen und Arrays

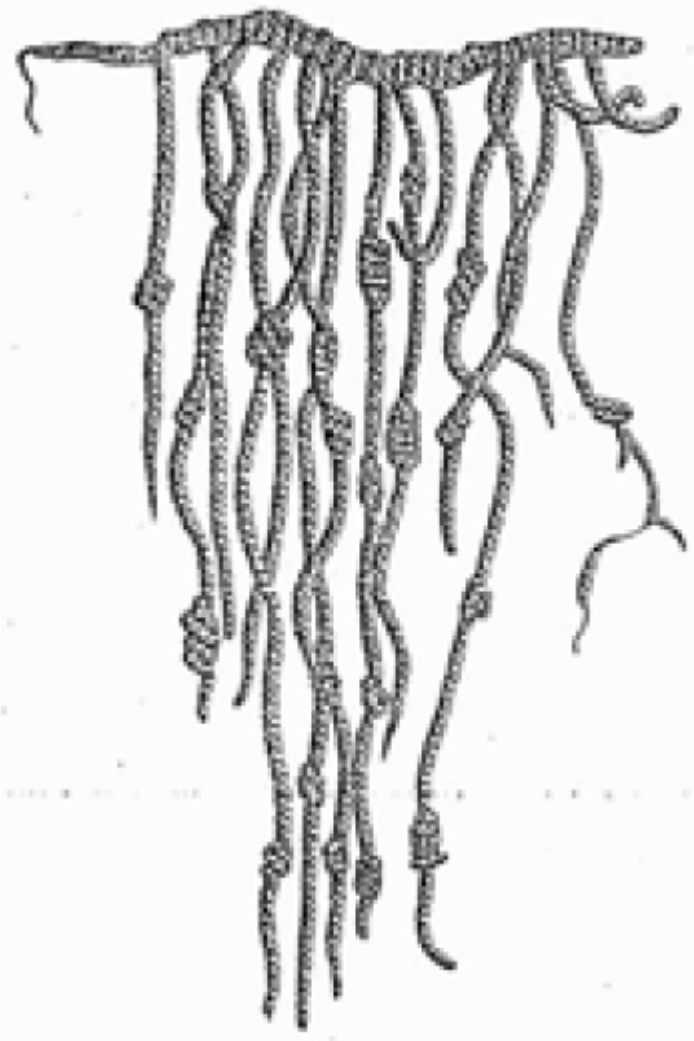
Maike Buchin

27.4.2017

Sehr alte Datenstrukturen

RUB

Keilschrifttafel der Sumerer



Knotenschrift der Inkas

Spezifikation

Objekte: Folge von Zahlen $\langle e_1, \ldots, e_n \rangle$

Operationen: z.B. Einfügen, Löschen, Größe?

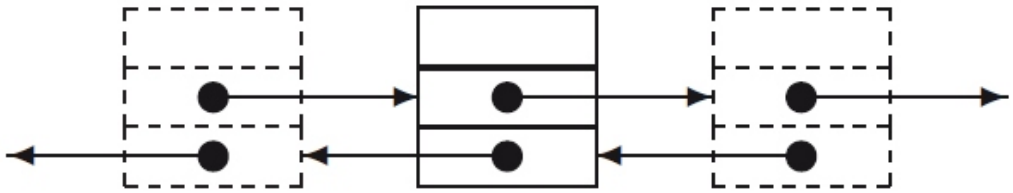
Datenstrukturen: Arrays und Listen

Unterschiede:

- statisch vs. dynamisch
- beschränkt vs. unbeschränkt

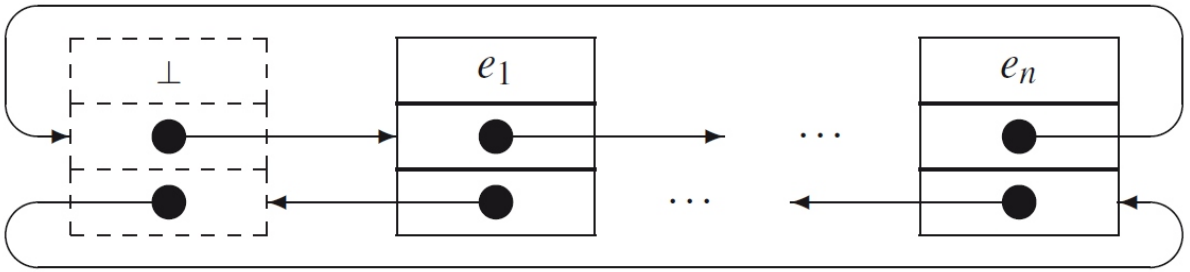
3.1 Verkettete Listen

- sind aus Knoten aufgebaut
- Knoten speichert: Inhalt und



  - einfach verkettet: Zeiger zum Nachfolger
  - doppelt verkettet: Zeiger zum Vorgänger und Nachfolger

Implementierung z.B. als Ring mit Dummyknoten h



Vorteil:

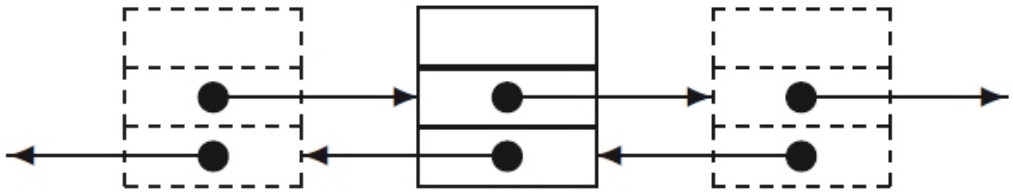
- man kann sie leicht modifizieren (einfügen, löschen, aneinanderhängen)

Nachteil:

- kein wahlfreier Zugriff auf Einträge

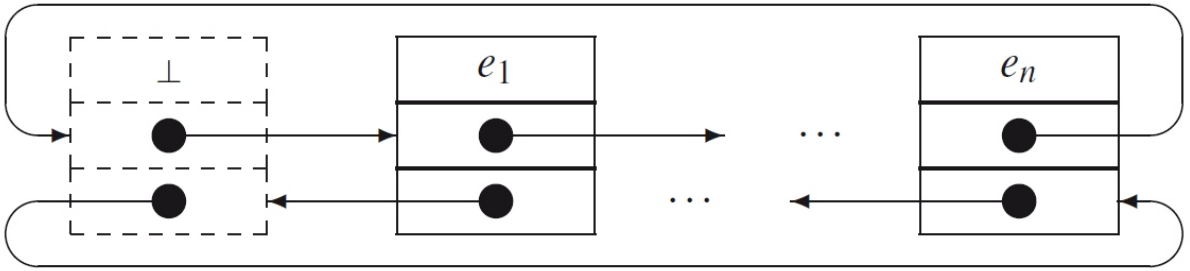
3.1 Verkettete Listen

- sind aus Knoten aufgebaut
- Knoten speichert: Inhalt und



  - einfach verkettet: Zeiger zum Nachfolger
  - doppelt verkettet: Zeiger zum Vorgänger und Nachfolger

Implementierung z.B. als Ring mit Dummyknoten h



Operationen: einfügen, löschen, Größe?, IstLeer?, evtl. aufteilen, evtl. aneinanderhängen

Effizienz: bei geeigneter Implementierung

- Einfügen und Größe? in $O(1)$
- Suche in $O(n)$

3.2 Arrays

RUB

1	2							n
e_1	e_1							e_n

- der index-basierte Zugriff $a[\cdot]$ kostet $O(1)$
- in *beschränkten* Arrays, kosten die Operationen Einfügen, Löschen, und Größe? ebenfalls $O(1)$
- häufig sind aber *unbeschränkte* Arrays wünschenswert
- ein unbeschränktes Array lässt sich durch beschränkte folgendermaßen emulieren:
  - wähle Konstanten $\alpha := 2 < 4 =: \beta$
  - wenn das beschränkte Array der Größe n voll ist, kopiere es in ein neues beschränktes Array der Größe αn
  - wenn das beschränkte Array der Größe n nur zu $\frac{n}{\beta}$ voll ist, kopiere es in ein neues beschränktes Array der Größe $\frac{n}{\alpha}$

3.3 Amortisierte Analyse

von unbeschränkten Arrays

Lemma: Sei a ein unbeschränktes Array, das anfangs leer ist. Jede Folge von m Einfüge- und Lösch-Operationen (am Ende des Array) auf a wird in Zeit $O(m)$ ausgeführt.

Beweis: Bankkonto-Methode
Wir führen ein Konto von 'Jetons' und

- zahlen für jedes Einfügen 2 Jetons ein
- zahlen für jedes Löschen 1 Jeton ein
- heben für Kopieren pro Eintrag 1 Jeton ab

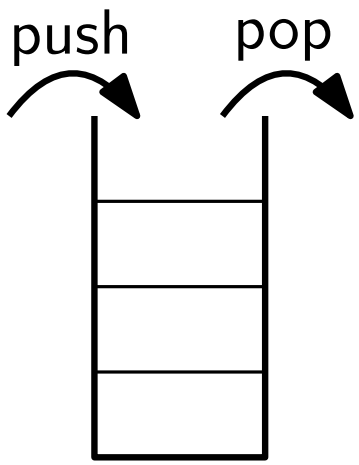
Beh.: Der Kontostand ist immer positiv.

Bew.: Ind(i) Konto nach i-tem Kopieren ist positiv.

3.4 Stapel und Warteschlangen

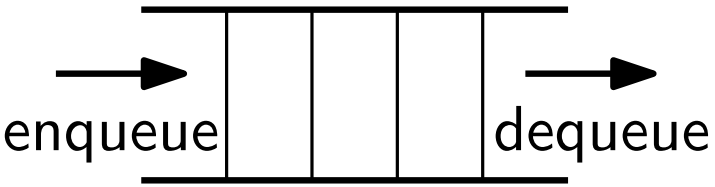
oft werden nur eingeschränkte Operationen auf Folgen benötigt

Stack
dt. Stapel



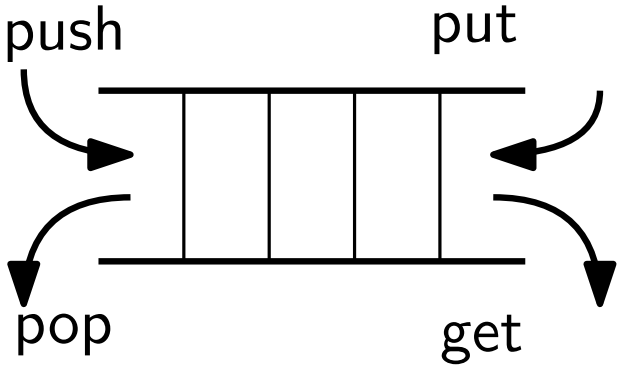
isEmpty?

Queue
dt. Warteschlange



isEmpty?

Dequeue
dt. Doppelschlange



isEmpty?

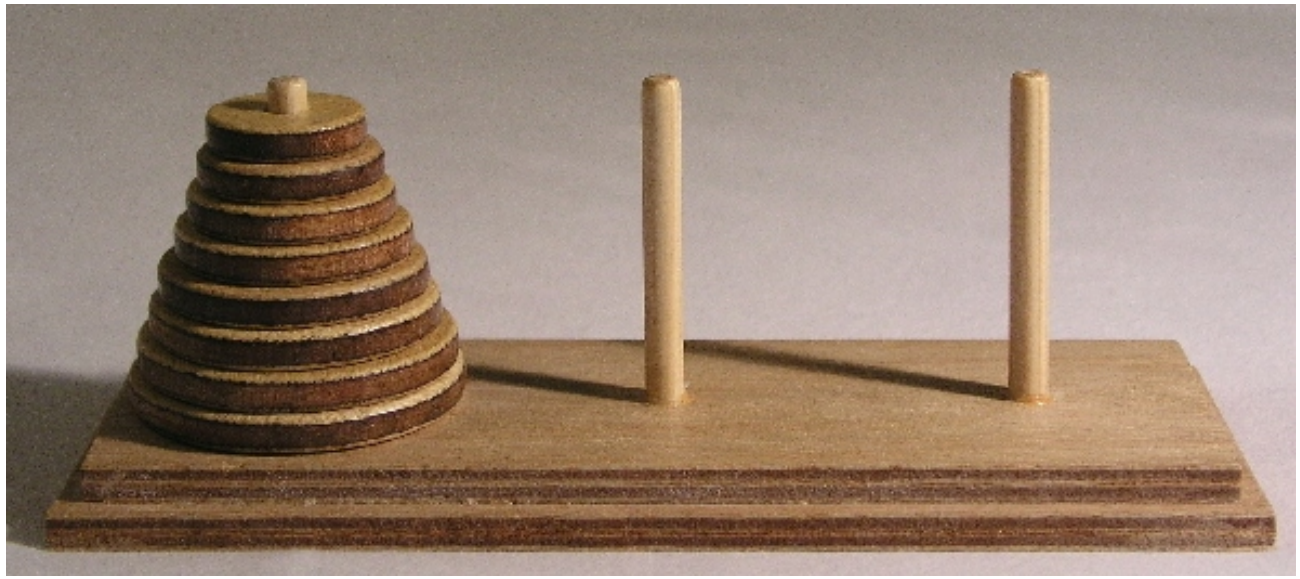
Implementierung: mit verketteten Listen oder Arrays

Effizienz: alle Operationen in $O(1)$

Türme von Hanoi

RUB

Rätsel: Auf einem von drei Stäben liegt ein Stapel von n Scheiben der Größe nach geordnet. Pro Zug darf die oberste Scheibe eines Stapels oben auf einen anderen Stapel gelegt werden, vorausgesetzt sie ist größer als die dort oben liegende Scheibe. Wieviel Züge werden benötigt, um den gesamten Stapel auf einen anderen Stab zu bewegen?



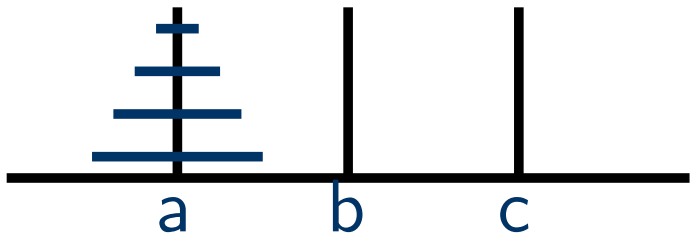
Türme von Hanoi

RUB

Lässt sich mit einem **rekursiven Algorithmus** lösen:

```
move (int i, a, b, c)
   if (i>0)
      move (i-1, a,c,b)
      put disc i on peg c
      move (i-1, b,a,c)
```

move i discs from peg a using peg b to peg c



Korrektheit: lässt sich zeigen mit Zusicherungen:

Vorbedingung: Die i Ringe liegen der Größe nach geordnet auf a, während b und c leer sind.

Nachbedingung: Die i Ringe liegen der Größe nach geordnet auf c, während a und b leer sind.

Laufzeit: $T(n) = 2T(n-1) + 1 = 2^n - 1 = O(2^n)$