5.12.07

4 Einfache Datenstrukturen

4.1 Grundoperationen

Aufgabenstellung:

- Verwalten einer Menge S von Objekten
- Ausführen von verschiedenen Operationen (s.u.!)

Im Folgenden:

S	Menge	
k	Wert eines Elements	(Name, Größe, d.h. Speicherinhalt)
x	Zeiger auf Element	(d.h. Speicheradresse)
NIL	spezieller, „leerer" Zeiger	

Dann:

SEARCH (S, k): „Suche in S nach k"

Durchsuche die Menge S nach einem Element von Wert k.

Ausgabe: Zeiger x, falls x existent
NIL, falls kein Element wert k hat

INSERT (S, x): „Füge x in S ein"

Erweitere S um das Element, das unter der Adresse x steht.

DELETE (S, x): „Entferne x aus S"

Lösche das unter der Adresse x stehende Element aus der Menge S.

MINIMUM (S): „Suche das Minimum von S"

Finde in S ein Element von kleinstem Wert.
(Annahme: Die Werte lassen sich vollständig ordnen)

Ausgabe: Zeiger x auf so ein Element

MAXIMUM (S): „Suche das Maximum von S"

Finde in S ein Element von größtem Wert.
(Annahme: Werte lassen sich ordnen)

Ausgabe: Zeiger x auf Element.

SUCCESSOR (S, x): „Finde das nächstgrößere Element"

Für ein in x stehendes Element in S,
bestimme ein ~~nächstgrößeres~~ Element von nächstgrößerem Wert in S.
(Annahme: Totalordnung)

Ausgabe: Zeiger y auf Element
NIL, wenn x Maximum von S ergibt

PREDECESSOR (S, x): „Finde das nächstkleinere Element"

Für ein in x stehendes Element in S, bestimme ein Element von nächstgrößerem Wert in S.
(Annahme: Totalordnung)

Ausgabe: Zeiger y auf Element
NIL, wenn x Minimum von S angibt.

Wie nimmt man das vor?
Wie lange dauert das? → kann unterschiedlich lange sein!

Beispiel: Sortierte Unterlagen →
Suche Blatt (1) ≈ vorne
Suche Blatt (62) ≈ hinten
Suche Blatt (32) ≈ Mitte
Suche Blatt (8) → Mitte von vorderer Hälfte von vorderer Hälfte

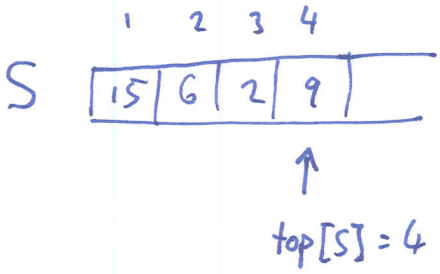
Unsortierte Unterlagen → immer alles durchgehen!

Laufzeit: In Abhängigkeit der Kardinalität n der Menge S

Also:
$O(1)$: Konstante Zeit → sehr schnell (z.B. Blatt ① in sortiertem Manuskript)
$O(\log n)$: Logarithmische Zeit → schnell (z.B. Blatt (8) in sortiertem Manuskript)
$O(n)$: lineare Zeit → langsam (Suche in unsortierten M)

4.2 Stapel und Warteschlange

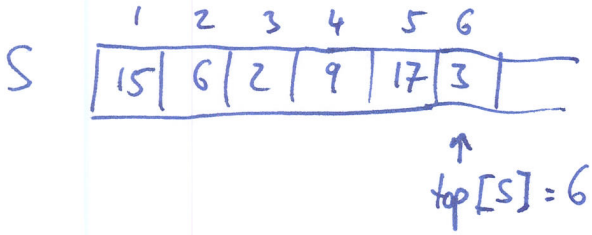
Stapel



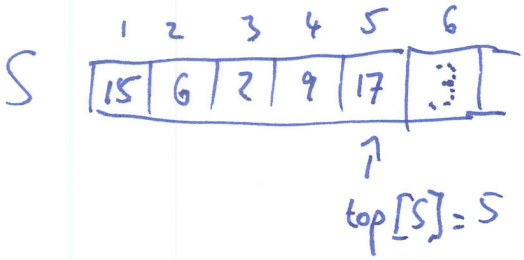
Stapel mit 4 Elementen

~~INSERT(S)~~

Einfügen von 17 und 3 liefert



Löschen?! Nur für letztes („oberstes") Element möglich:



Spezielle Namen: PUSH
und POP!

In Pseudocode:

```
STACK-EMPTY(S)
1  if top[S] = 0
     then return WAHR
     else return FALSCH
```

} Absicherung gegen Zugriff auf leeren Stapel

Einfügen:

```
PUSH (S, x)
1  top[S] := top[S] + 1
2  S[top[S]] := x
```

(wenn Speicherplatz ausgeschöpft, dann „Stack overflow"!)

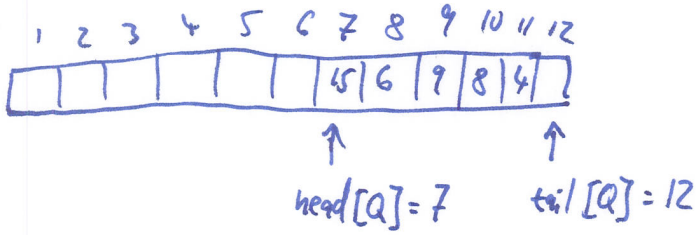
Löschen:

```
POP (S)
1  if STACK-EMPTY (S)
       then error "underflow"
       else top[S] := top[S] - 1
            return S[top[S] + 1]
```

Aufwand?! Jeweils $O(1)$!

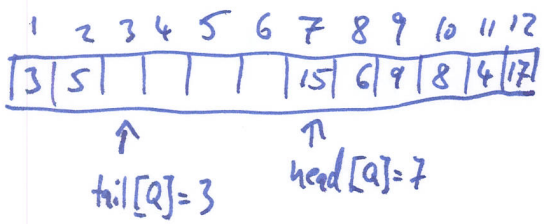
(Aber: Suchen etc. dauern länger!)

Warteschlange

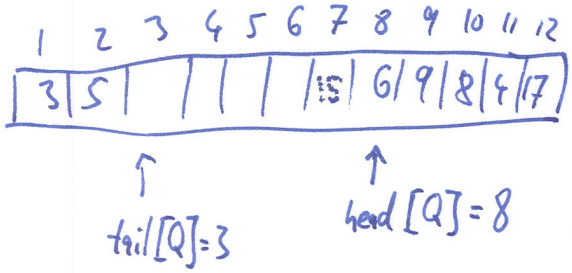


Einfügen von 17, 3, 5 liefert

(hinten!)



Entfernen → nur von Vorne! von 15 liefert



In Pseudocode; hier ohne Fehlerkontrolle

Einfügen:

```
ENQUEVE (Q, x)
    Q [tail [Q]] := x
    if  tail [Q] = länge [Q]
      then  tail [Q] := 1
      else  tail [Q] := tail [Q] + 1
```

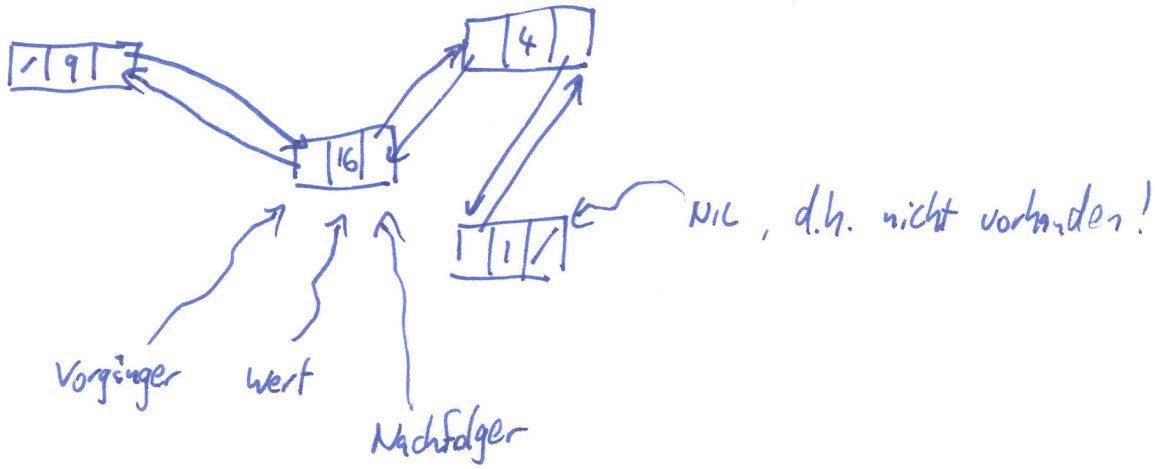
Löschen

```
DEQUEVE (Q)
    x := Q [head [Q]]
    if  head [Q] = länge [Q]
      then  head [Q] := 1
      else  head [Q] := länge [Q] + 1
```

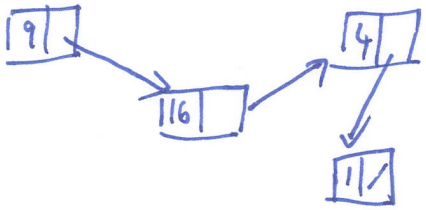
Aufwand wieder $O(1)$, aber Suchen etc. aufwändiger!

4.3 Verkettete Listen

Idee: Ordne Objekte nicht explizit in aufeinanderfolgenden Speicherzellen an, sondern gib jeweils Vorgänger und Nachfolger an:



Einfach verkettete Liste: Nur Nachfolger wird gemerkt



Doppelt verkettete Liste: Nachfolger und Vorgänger!

Alle Operationen werden ermöglicht:

LIST-SEARCH (L, k) (Suche nach Objekt)

```
1  x := head[L]
2  while x ≠ NIL und wert[x] ≠ k
3        do  x := nachf[x]
4  return x
```

Laufzeit: $O(n)$

LIST-INSERT (L, x) (Einfügen vorne)

```
1  nachf[x] := head[L]
2  if head[L] ≠ NIL
3     then vorg[head[L]] := x
4  head[L] := x
5  vorg[x] := NIL
```

Laufzeit: $O(1)$

LIST-DELETE (L, x) (Entfernen irgendwo)

```
1  if vorg[x] ≠ NIL
2     then nachf[vorg[x]] := nachf[x]
3     else head[L] := nachf[x]
4  if nachf[x] ≠ NIL
      then vorg[nachf[x]] := vorg[x]
```

Laufzeit $O(1)$