Algorithmen und Datenstrukturen 1

Prof. Dr. Carsten Lecon

Elementare Datenstrukturen

- Einführung
- Stapel
- Schlangen
- Verkette Listen
- Repräsentation von Bäumen
- Zusammenfassung

Bildquellen: presse.openface.de; www.bcm-news.de

Lernziele

- Kennenlernen (weiterer) wichtiger, grundlegender Datenstrukturen
- Kenntnis der programmtechnischen Umsetzung der Datenstrukturen (konkret in Java)
- Kenntnis der wichtigsten Begriffe
- Verständnis des Prinzips der Abstraktion

Einführung

- Menge (engl. *set*)
  - Fundamentale Datenstruktur (s. Mathematik…)
  - Ansammlung von Elementen
  - Dynamische Datenstruktur
    - Wachsen
    - Schrumpfen
- …
- **Algorithmen** führen Operationen auf **Datenstrukturen** aus.

Einführung

- Definition Wörterbuch (engl. *dictionary*)

- Eine Datenstruktur heißt **Wörterbuch**, wenn sie die Operationen
  - Suchen
  - Einfügen
  - Löschen

  unterstützt.

Einführung

- Jedes Element einer solchen Datenstruktur enthält verschiedene Felder:
  - **Schlüssel** (*key*)
    Identifizierendes Element
  - **Daten**
  - **Verwaltungsdaten**

Einführung

- Notation :

- **S** Die Datenstruktur selbst
- **x** Ein Element der Datenstruktur
- **key(x)** Schlüssel des Elements x

Einführung

- Typen von Operationen auf dynamischen Elementen

- Unterscheidung in zwei Gruppen:
  - **Anfragen** (geben eine Information zurück)
  - **Modifizierende Operationen** (MO)

Einführung

- Operationen auf dynamischen Mengen

- Typische Operationen:
  - **INSERT(S,x)**
    Füge ein Element x in S ein (MO)
  - **DELETE(S,x)**
    Lösche Element x auf S
  - **SEARCH(S,x)**
    Suche nach Element x in S

Einführung

- Operationen auf dynamischen Mengen

- **MINIMUM(S)**
  Liefert kleinstes Element in S
- **MAXIMIMUM(S)**
  Liefert größtes Element in S
- **SUCCESSOR(S,x)**
  Liefert Element, das auf x folgt
- **PREDECESSOR(S,x)**
  Liefert Element, das vor x steht

Einführung

- Anmerkungen:
- Bei SUCCESSOR und PREDECESSOR ist Ordnung erforderlich (ggf. Reihenfolge des Einfügens)
- Leeres Element (bspw., wenn Anfrage erfolglos): NIL bzw. null

Elementare Datenstrukturen

- Einführung
- Stapel
- Schlangen
- Verkette Listen
- Repräsentation von Bäumen
- Zusammenfassung

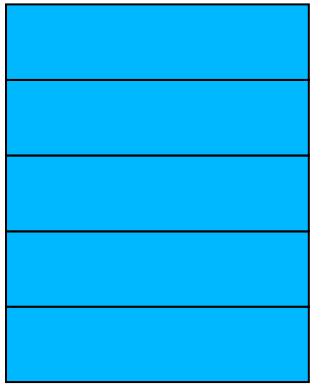
Bildquellen: presse.openface.de; www.bcm-news.de

Stapel

- Stapel: Datenstruktur, bei die modifizierenden Operationen auf fest definierten Positionen wirken:
  - **INSERT(S,x)**
    Dem Stapel S wird x als neues, oberes Element hinzugefügt.
  - **DELETE(S)**
    Das oberste Element des Stapels S wird gelöscht und zurückgegeben.
- **LIFO**-Prinzip.
  Last In – First Out

Stapel

- Andere Namen für INSERT / DELETE:
  - INSERT → **push**
  - DELETE → **pop**



pop

push

Stapel

- Notationen:
- **S**: Der Stapel selbst
- Bei Implementierung durch Feld (Array):

  - **top(S)** Position des obersten Elements von S
  - **S(i)** Element an Position i
  - **S(1,…,top(S))** Alle Elemente des Stapels

Stapel – Operationen (Pseudocode)

```
1    function isEmpty(S)
2      if top(S) = 0 then
3        return true
4      else
5        return false
6      end if
7    end function
```

Stapel – Operationen (Pseudocode)

```
1 procedure push(S,x)
2   top(S) = top(S)+1
3   S[top(S)] = x
4 end procedure
```

Fehlerquellen?

Stapel – Operationen (Pseudocode)

```
1    function pop(S)
2      if isEmpty(S) then
3        Fehlermeldung „Underflow“
4      else
5        top(S) = top(S)-1
6        return S[top(S)+1]
7      end if
8    end function
```

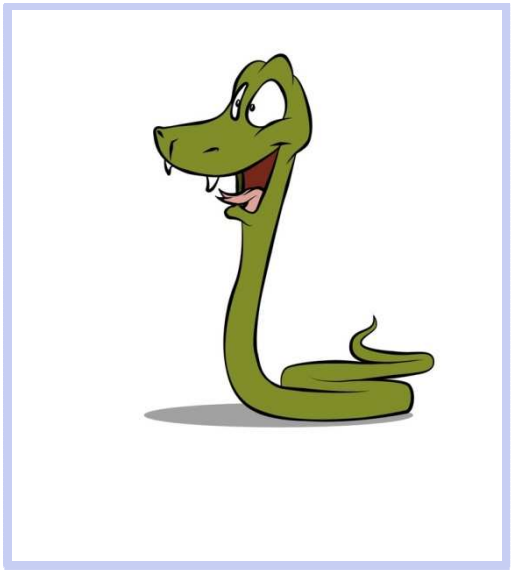
Stapel

- Bemerkungen zu dieser Realisierung:
  - Ein gelöschtes Element wird nicht zerstört.
  - Es muss auf Fehler reagiert werden.

- Laufzeiten:
  - Unabhängig von der Stapelgröße n: O(1)

Elementare Datenstrukturen

- Einführung
- Stapel
- Schlangen
- Verkette Listen
- Repräsentation von Bäumen
- Zusammenfassung



Bildquellen: presse.openface.de; www.bcm-news.de

Schlangen

- Schlangen sind eine Unterordnung der Schuppenkriechtiere.
- Schlangen spielen in der Kulturgeschichte und der Mythologie eine große Rolle.

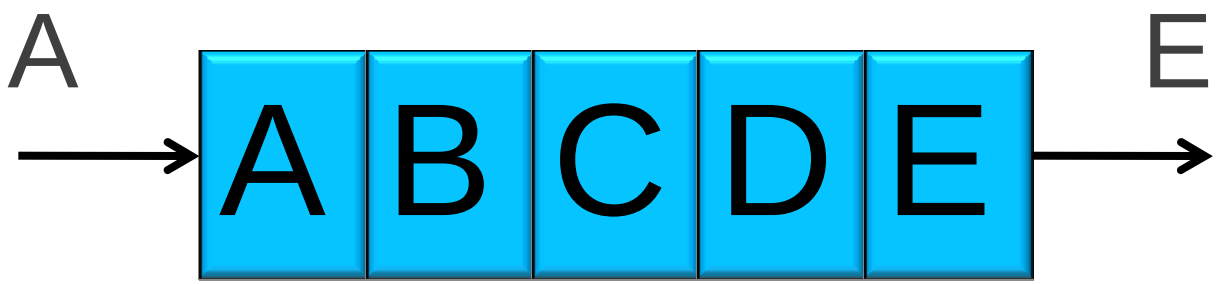
- (Warte-) Schlangen sind eine Unterordnung von Datenstrukturen.
- Schlangen spielen in der Informatik eine große Rolle.

Bildquellen: Wikipedia

Schlangen

- Einsatzgebiet:
  - Pufferung von Anforderungen, die nicht sofort bearbeitet werden können
- Anwendungsbereiche:
  - Druckerwarteschlange
  - Netzwerkübertragung

Schlangen



FIFO-Prinzip: First In – First Out

Schlangen

- Schlangen: Datenstruktur, bei die modifizierenden Operationen auf fest definierten Positionen wirken:
  - **INSERT(S,x)**
    Dem Stapel S wird x als neues Element hinzugefügt (hinten anstellen)
  - **DELETE(S)**
    Das älteste Element des Stapels S wird gelöscht und zurückgegeben.
- **FIFO**-Prinzip.
  Fist In – First Out

Schlange

- Andere Namen für INSERT / DELETE:
  - INSERT → **enqueue**
  - DELETE → **dequeue**

Schlange

- Notationen:
- **Q**: Die Schlange selbst
- Bei Implementierung durch Feld (Array):

  - **length(Q)** Max. Anzahl an Elementen in Q
  - **head(Q)** Position des ersten Elements in Q
  - **tail(S)** Freie Position für das nächste Element
  - **Q(i)** Element an Position i

Schlange - Pseudocode

```
1    function isEmpty(Q)
2      if head(Q) = tail(Q) then
3        return true
4      else
5        return false
6      end if
7    end function
```

Schlange – Pseudocode

```
1  procedure enqueue(Q,x)
2    Q[tail(Q)] = x
3    if (tail(Q) = length(Q)) then
4      tail(Q)= 1
5    else
6      tail(Q) = tail(Q)+1
7    end if
8  end procdure
```

Fehlermöglichkeiten?

Schlange – Pseudocode

```
1     function dequeue(Q)
2       x = Q[head(Q)]
3       if head(Q) = length(Q) then
4         head(Q) = 1
5       else
6         head(Q) = head(Q)+1
7       end if
8       return x
9     end function
```

- Fehlermöglichkeiten?

Schlange

- Bemerkungen zu dieser Realisierung:
  - Ein gelöschtes Element wird nicht zerstört.
  - Es muss auf Fehler reagiert werden.
  - Q ist leer ↔ head(Q) = tail(Q) (→ isEmpty())
  - Q ist voll ↔ ?

- Laufzeiten:
  - Unabhängig von der (maximimalen) Stapelgröße n: O(1)

Elementare Datenstrukturen

- Einführung
- Stapel
- Schlangen
- Verkette Listen
- Repräsentation von Bäumen
- Zusammenfassung

Bildquellen: presse.openface.de; www.bcm-news.de

Verkettete Listen

- Kennzeichen:
  - Die Elemente sind linear angeordnet
  - Die Anordnung der Elemente ergibt sich durch **Verkettung**
    → Die Position der Elemente kann nicht errechnet werden
- Verkettete Listen unterstützen alle zuvor definierten Operationen (Wörterbuchfunktionen)
  - Auch Löschen eines beliebigen Elements
  - Komplexität?
- Im Folgenden: Doppelt verkettete Listen (nicht als Array implementiert)

Verkettete Listen

- Dynamische Datenstruktur
- Ein Eintrag besteht aus:
  - Daten
  - Referenz auf den nächsten Eintrag
  - Referenz auf den vorigen Eintrag

Zeiger auf Vorgänger

Datenwert

Zeiger auf Nachfolger

Zeiger auf Nachfolger

Zeiger auf Vorgänger

Verkettete Liste

Notation:

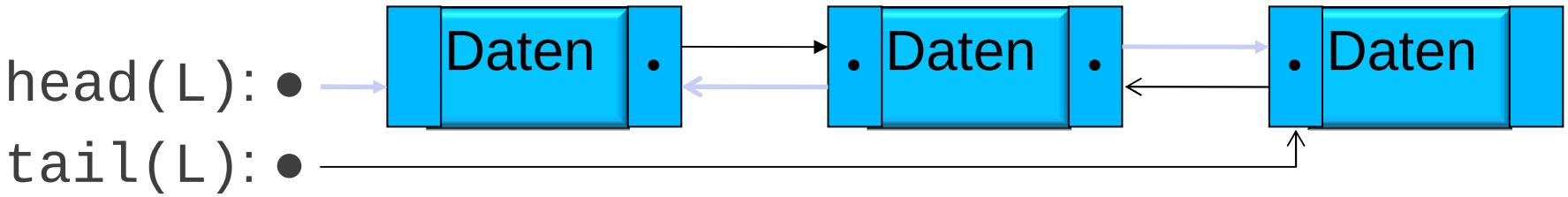
- **L** Die Liste selbst
- **head(L)** Zeiger auf das erste Element der Liste (Listenkopf)
- **tail(L)** Zeiger auf das letzte Element der Liste (Listenende)
- **x** Ein Listenelement
- **key(x)** Der Schlüssel des Listenelements x
- **next(x)** Zeiger auf den Nachfolger in der Liste
- **prev(x)** Zeiger auf den Vorgänger in der Liste

Verkettete Liste

- Beispiele:

- Leere Liste:
  `head(L):` ●

- Liste mit drei Elementen:



Verkettete Liste

Bemerkungen:

- **`prev(x) = NIL`**
  - → Element hat keinen Vorgänger.
  - → Listenkopf
- **`next(x) = NIL`**
  - → Element hat keinen Nachfolger.
  - → Listenende
- **`head(L) = NIL`**
  - → Liste ist leer.

Verkettete Liste

Bemerkungen (Forts.)

- Einfach verkettete Liste: Spezialfall der doppelt verketteten Liste (ohne **`prev`**)
- Listen können sortiert sein.
- Zyklische Listen:
  - **`prev(head(L)) = tail(L)`** und
    `next(tail(L)) = head(L)`

Verkettete Listen - Operationen

```
1    function listSearch(L,k)
2      x=head(L)
3      while x≠NIL and key(x)≠k do
4        x = next(x)
5      end while
6      return x
7    end function
```

Verkettete Listen - Operationen

Hinzufügen am Anfang:

```
1     procedure listInsert(L, x)
2       next(x) = head(L)
3       prev(x) = NIL
4       if head(L)≠NIL then
5         prev(head(L))=x
6       else
6         tail = x
        end if
7       head(L)=x
8     end procedure
```

Hinzufügen am Ende?

Hinzufügen nach/vor Element y?

Verkettete Listen - Operationen

```
1    procedure listDelete(L,x)
2      if prev(x) ≠ NIL then
3        next(prev(x)) = next(x)
4      else
5        head(L) = next(x)
6      end if
7      if next(x) ≠ NIL then
8        prev(next(x))=prev(x)
9      end if
10   end procedure
```

Verkettete Listen - Operationen

- Löschen:
  - Löschen eines Elements aus der Mitte
  - Löschen eines Elements am Anfang
  - Löschen eines Elements am Ende

Ergänzungen zu Listen

- Probleme bei Listen und Binären Suchbäumen:
  - Komplizierte Anfragen auf Anfang/Ende der Liste bzw. des Baumes

Liste: Wächter

- Komplizierte Anfragen auf Anfang/Ende der Liste (Randelemente)
- Verbesserung: Einführung von „Wächtern“ (sentinel); z.B. `sentinel(L)`
- Alle Verweise auf NIL werden durch Verweise auf den Wächter ersetzt.
- `head` und `tail` wird nicht mehr benötigt.

Liste (Wächter): Suchen

```
1 function listSearch(L,k)
2   x = next(sentinel(L))
3   while x≠sentinel(L) and key(x)≠k  do
4     x = next(x)
5   end while
6   return x
7 end function
```

Liste (Wächter): Hinzufügen am Anfang

```
1 procedure listInsert(L,x)
2   next(x) = next(sentinel(L))
3   prev(x) = sentinel(L)
4   prev(next(sentinel(L))) = x
5   next(sentinal(L)) = x
6 end procedure
```

Hinzufügen am Ende?

Liste (Wächter)

Anmerkungen:

- Aus der Liste wird eine zirkulare Liste.
- Die leere Liste besteht nur aus dem Wächter.
- Die Liste ist leer, wenn `next(sentinel(L))=prev(sentinel(L))`.

Liste (Wächter): Löschen

```
1 procedure listDelete(L,x)
2   next(prev(x)) = next(x)
3   prev(next(x)) = prev(x)
4 end procedure
```