Lösung der Beispielaufgaben – Informatik I für VIW
Fakultät Verkehrswissenschaften
Fachrichtung Verkehrsingenieurwesen

Formale Sprachen

- $L_3^* = \{\varepsilon, dd, dddd, \ldots\} = \{(dd)^n \mid n \in \mathbb{N}\}$
- $L_1(L_2)^* = \{a, ac, acc, accc, \ldots\} = \{ac^n \mid n \in \mathbb{N}\}$
- $(L_2L_3)^* = \{\varepsilon, cdd, cddcdd, cddcddcdd, \ldots\} = \{(cdd)^n \mid n \in \mathbb{N}\}$

Es gilt $\{cc, cddc, cddddc, cddddddc, \ldots\} = L_2L_3^*L_2$

Ableitungen

- $\underline{S} \to a\underline{S}C \to ab\underline{C} \to abc\underline{C} \to abcc$
- $\underline{S} \to a\underline{S}C \to aa\underline{S}CC \to aab\underline{C}C \to aabc\underline{C} \to aabcc$

Das Wort *aabc* kannn nicht abgeleitet werden, da zum Erzeugen der beiden *a* jeweils die erste Regel angewandt werden muss. Dabei entsteht zweimal das Nichtterminalsymbol *C*. Aus jedem der *C* muss aber mindestens ein *c* abgeleitet werden.

Berechnung der erzeugten Sprache für einfache BNF

$$\begin{aligned} L_S(E) &= \{a\} \cdot L_A(E) \cdot L_A(E) \cup \{b\} \,, \\ L_A(E) &= \{d\} \cup \{c\} \cdot L_B(E) \cdot \{c\} \,, \\ L_B(E) &= \{a\} \,. \end{aligned}$$

Berechnung der Lösung:

$$\begin{aligned} L_B(E) &= \{a\} \,, \\ L_A(E) &= \{d\} \cup \{c\} \cdot \{a\} \cdot \{c\} = \{d, cac\} \,, \\ L_S(E) &= \{a\} \cdot \{d, cac\} \cdot \{d, cac\} \cup \{b\} = \{add, adcac, acacd, acaccac, b\} \,. \end{aligned}$$

Die durch E erzeugte Sprache ist $L(E) = L_S(E) = \{add, adcac, acacd, acaccac, b\}$.

Berechnung der erzeugten Sprache für zyklische BNF

$$L_S(E) = \{a\} \cdot L_S(E) \cup \{a\} \,.$$

Annahme: $L_S(E) = \{a^n \mid n \geq 0\}$. Dann würde die folgende Gleichung gelten:

$$\begin{aligned}\{a^n \mid n \geq 0\} &= \{a\} \cdot \{a^n \mid n \geq 0\} \cup \{a\} \\ &= \{aa^n \mid n \geq 0\} \cup \{a\} \\ &= \{a^n \mid n \geq 1\} \cup \{a\} \\ &= \{a^n \mid n \geq 1\}\,.\end{aligned}$$

Das ist ein Widerspruch, da die linke Seite das leere Wort enthält, die rechte Seite jedoch nicht.

Die erzeugte Sprache ist die Menge $L(E) = L_S(E) = \{a^n \mid n \geq 1\} = \{a, aa, aaa, \ldots\}$.

Schleifen

```
3 7 8 10
```

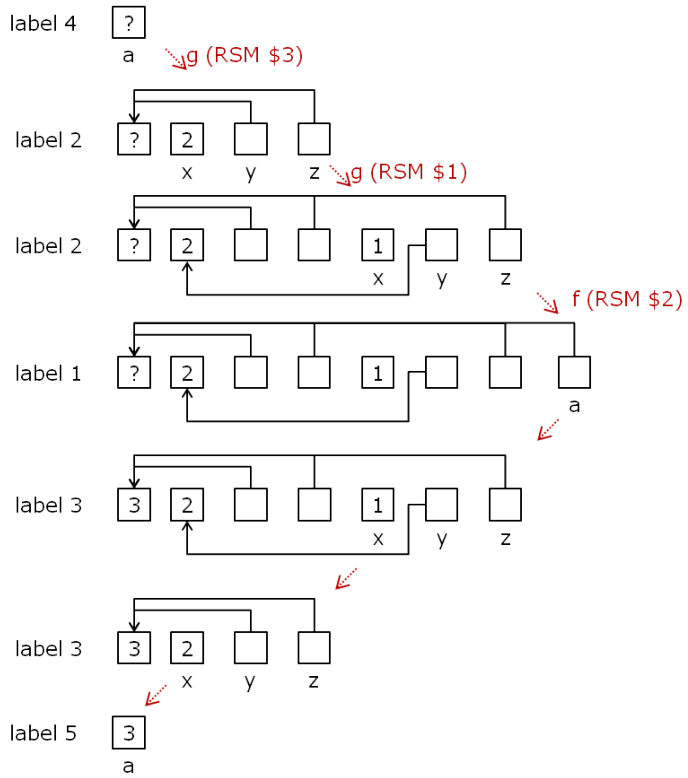
Einfache Funktionen

```
int fak (int n) {
  if (n <= 1)
    return 1;
  return n * fak(n-1);
}

int bin (int n, int k) {
  return fak(n) / (fak(k) * fak(n-k))
}
```

Funktionen mit Arrays

```
int istSortiert(int array[], int n) {
  int i;
  for (i = 0; i < n-1; i++) {
    if (array[i] < array[i+1])
      return 0;
  }
  return 1;
}

int countEven(int array[], int n) {
  int i, z = 0;
  for (i = 0; i < n; i++) {
    if (array[i] % 2 == 0)
      z++;
  }
  return z;
}
```

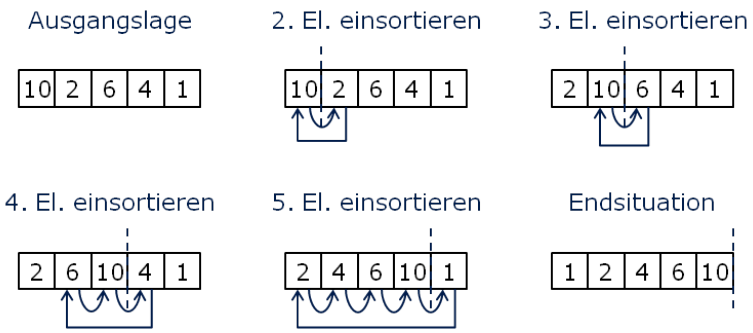
Speicherbelegungsprotokolle



Komplexität

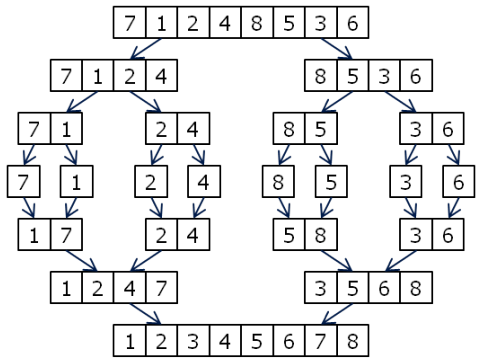
- kurze Ausführungszeit für `[11, 3, 17, 5, 3]`
- lange Ausführungszeit für `[5, 6, 4, 10, 1]`
- Best-Case-Laufzeit: $O(1)$
- Worst-Case-Laufzeit: $O(n)$

Insertion Sort

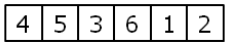


Bei der Eingabefolge 3, 4, 2, 1, 5 sind beispielsweise fünf Verschiebungen nötig.

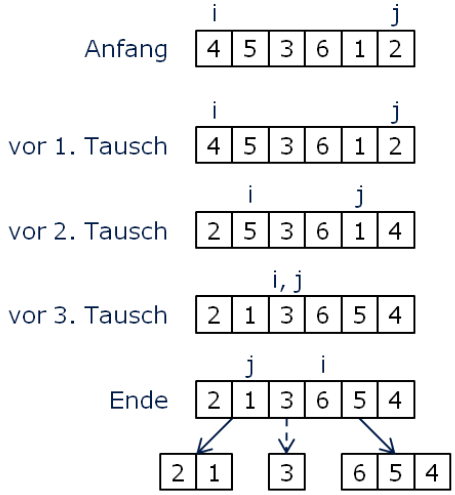
Mergesort



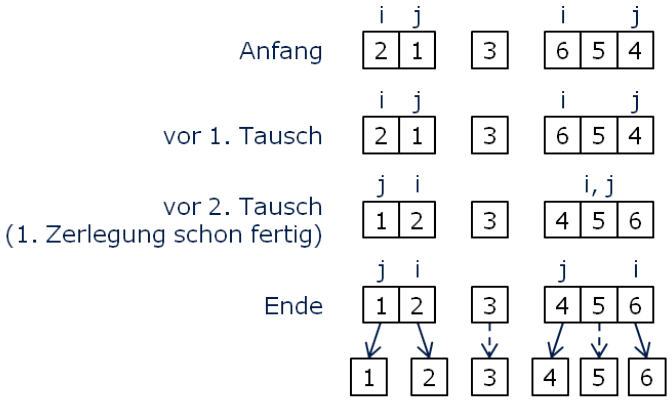
Quicksort



Zerlegung der Gesamtfolge; Pivotelement: 3



Zerlegung von 2, 1 (PE: 2) und von 6, 5, 4 (PE: 5); hier parallel dargestellt



alle Teilfolgen haben Länge 1; keine weiteren Zerlegungen

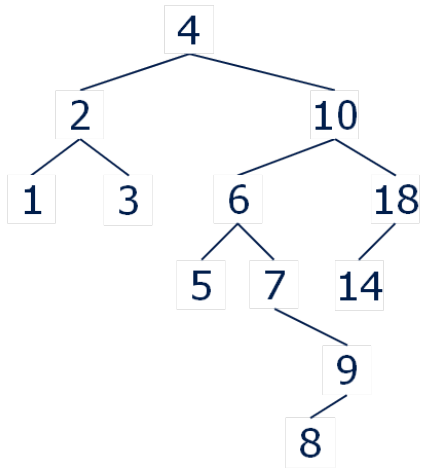
Einfache Listenfunktionen

```
int kleinstes(struct elem *wurzel) {
  int z;
  if (wurzel->next == NULL) /*Liste enthaelt nur noch ein Element */
    return wurzel->wert;

  z = kleinstes(wurzel->next);

  if (wurzel->wert < z)
    return wurzel->wert;

  return z;
}
```

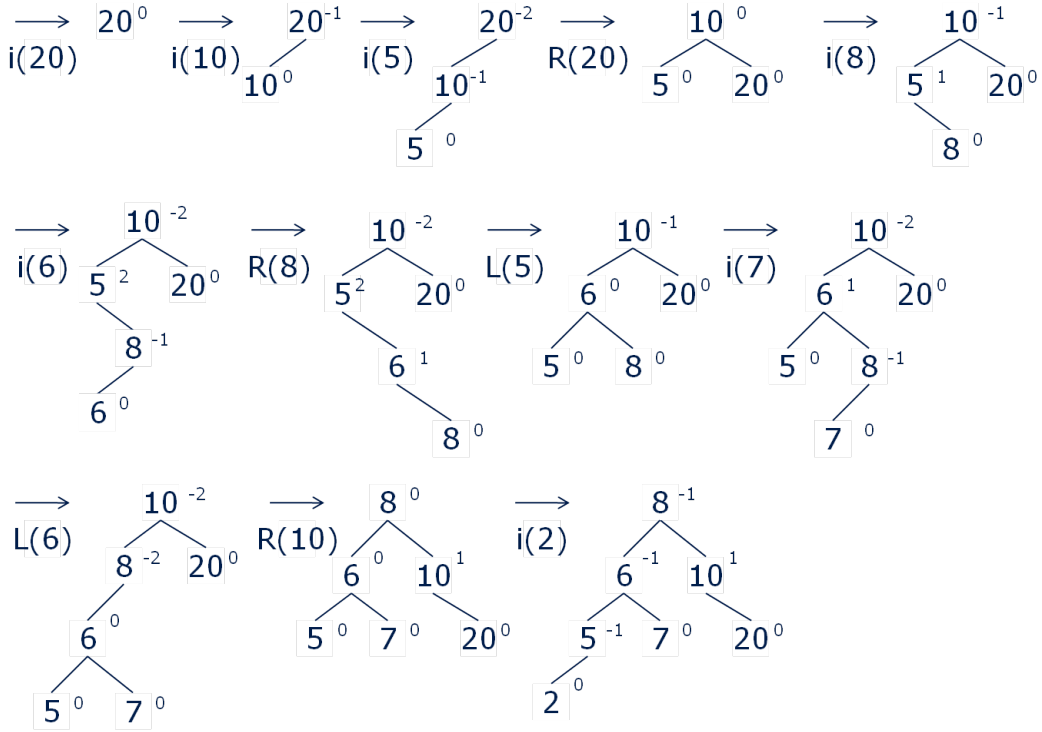
Strukturverändernde Listenfunktionen

```
void remove(struct liste *li) {
  li->wurzel = li->wurzel->next;
}
```

Suchbäume



AVL-Bäume



Dijkstra-Algorithmus

(2, 0, -)	(1, 3, 2), (3, 20, 2), (4, 5, 2), (5, 10, 2)
(1, 3, 2)	(3, 18, 1), (4, 5, 2), (5, 10, 2)
(4, 5, 2)	(3, 18, 1), (5, 7, 4)
(5, 7, 4)	(3, 15, 5), (6, 11, 5)
(6, 11, 5)	(3, 14, 6)
(3, 14, 6)	-

kürzeste Wege

1: 2, 1
2: 2
3: 2, 4, 5, 6, 3
4: 2, 4
5: 2, 4, 5
6: 2, 4, 5, 6