Kostenmaße

Bei der TM nur ein Kostenmaß:

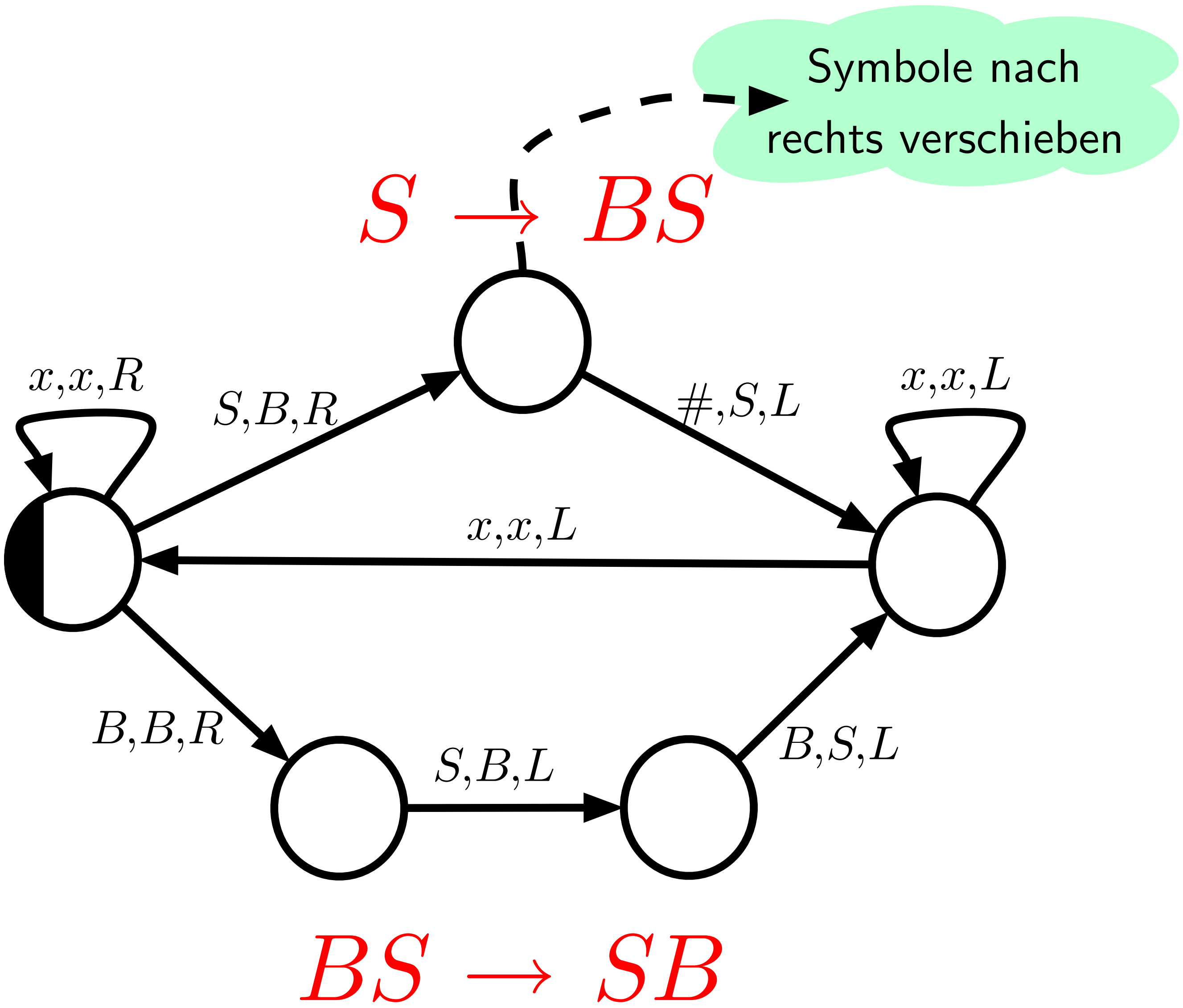
- Ein *Schritt* (Konfigurationsübergang) kostet eine *Zeiteinheit*;
- eine *Bandzelle* kostet eine *Platzeinheit*.

Bei der RAM zwei Kostenmaße:

- **uniformes Kostenmaß:** (wie oben);
- **logarithmisches Kostenmaß:** Die Länge der Binärdarstellung von Registeradressen un Inhalten wird berücksichtigt.

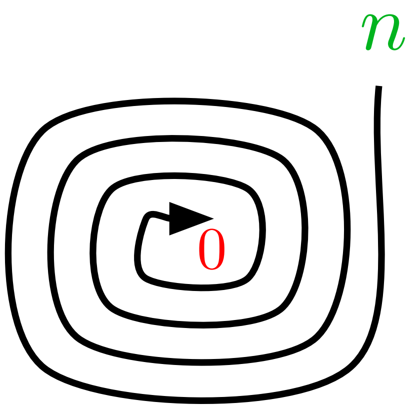
Im folgenden: „uniformes Kostenmaß“!

TM-Simulation von Grammatiken

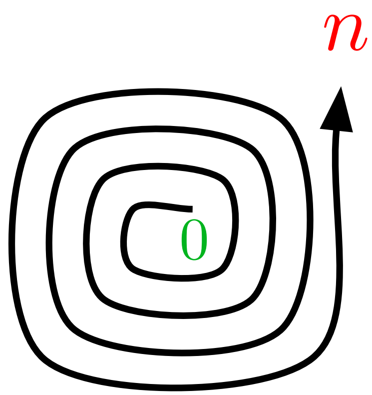


Algorithmentechniken

rekursiv vs. iterativ



rekursiv



iterativ

f(x) := g(...,f(x-1),...)
:= g(...,g(...,f(x-2),...),...)

FOR i FROM 0 TO x DO
f := g(...,f,...)

Beispiel: Fakultät rekursiv

- $n! := \begin{cases} 1 & \text{für } n = 0 \\ n \cdot (n-1)! & \text{sonst} \end{cases}$
- Ein rekursives Programm:

```
FUNCTION fac(n)
  BEGIN
    IF n = 0
      THEN
        RETURN 1
      ELSE
        RETURN n·fac(n-1)
    END.
```

Beispiel: Fakultät iterativ

- Die übliche Definition $n! := \prod_{i=1}^{n} i$ suggeriert eine iterative Lösung.
- Ein iteratives Programm:

```
FUNCTION fac(n)
  BEGIN
    DECL prod INTEGER;
      prod ⟵ 1;
        FOR i FROM 1 TO n DO
        prod ⟵ prod·i
    RETURN prod
    END.
```

Die Qual der Wahl ...

- **Welche der beiden Methoden soll nun verwendet werden?**
  - Diese Frage kann immer nur mit Blick auf das Problem entschieden werden.
  - Manchmal lautet die Antwort: **„Keine von beiden!“**
    - Warum? Z.B. ist bereits ab $13! = 6227020800$ das Ergebnis bei 32-Bit-Integer-Darstellung nicht mehr darstellbar!
- **Was gibt es für Alternativen?**

Die Suche nach Alternativen

- **Im Fall der Fakultätsfunktion ist die Antwort auf die erste Frage einfach:**
  - Diese Funktion sollte gar nicht programmiert, sondern nur durch eine Tabelle implementiert werden!
- Häufig sucht man nach *geschlossenen Formen* (Berechnung ohne Schleifen).
- In jedem Fall sollte die Komplexität abgeschätzt werden, um das effizienteste Verfahren auszuwählen!

Effektivität vs. Effizienz

Effektivität: Ein Problem heißt *effektiv lösbar*, wenn ihm eine berechenbare Funktion zugrunde liegt, d.h. es ein ausführbares, endlich beschreibbares Verfahren gibt, das für alle Probleminstanzen in endlicher Zeit eine korrekte Lösung liefert. *(BF)*

Effizienz: Ein Algorithmus heißt *effizient*, wenn er ein vorgegebenes Problem in möglichst kurzer Zeit und/oder mit möglichst geringem Aufwand an Betriebsmitteln löst. *(Informatik-Duden)*

Das Problem der binären Suche

Gegeben:	eine in aufsteigender Reihenfolge sortierte Liste `list` von Zahlen und ein Wert x
Gesucht:	Kommt x in `list` vor?
Antwort:	JA oder NEIN

- Allgemein kann auch von einer Menge mit vorgegebener linearer Ordnung $<$ ausgegangen werden.
- So formuliert man das Problem dann z.B. auch für Listen von Wörtern.

Schema: binäre Suche

1, 5, 26, 36, 38, 40, 55, 72, 77, 98

Enthält die Liste die 6?

1, 5, 26, 36, 38, 40, 55, 72, 77, 98

$6 \leq 38$ $\qquad$ $6 \geq 40$

1, 5, 26, 36, 38

$6 \leq 26$ $\qquad$ $6 \geq 36$

1, 5, 26

$6 \leq 5$ $\qquad$ $6 \geq 26$

Nein!

Algorithmus für binäre Suche

`Suche(`x`,`A`)`

1. $\left[\begin{array}{lll} |A| = 1 & A \neq \{x\}: & x \text{ nicht in } A \\ & A = \{x\}: & x \text{ in } A \end{array}\right.$

2. Partitioniere den Suchraum in zwei Teile A_1 und A_2, so daß $A = A_1 \uplus A_2$ mit $|A_1| = \lceil \frac{1}{2} |A| \rceil$ und $|A_2| = \lfloor \frac{1}{2} |A| \rfloor$, sowie $max(A_1) < min(A_2)$.

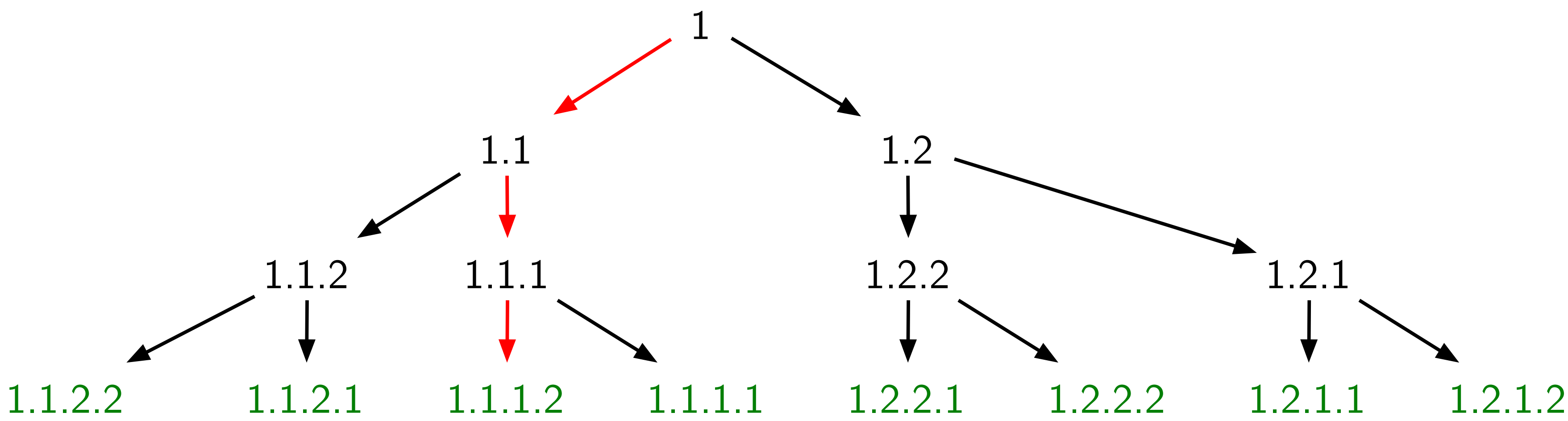
3. $\left[\begin{array}{ll} \max(A_1) < x < \min(A_2): & x \text{ nicht in } A \\ x \leq \max(A_1): & \texttt{Suche}(x, A_1) \\ x \geq \min(A_2): & \texttt{Suche}(x, A_2) \end{array}\right.$

Kosten der binären Suche

- Es wird jeweils mit einem ungefähr halbiertem Suchraum fortgefahren bis das Element gefunden ist.
- Schlimmstenfalls werden $O(\log n)$ Partitionierungen durchgeführt.
  - Ein vollständiger ausgeglichener Binärbaum mit n Blättern hat eine Tiefe von $\log_2 n$.

In einem Feld mit einer Millionen Zahlen sind somit höchstens 20 Partitionierungsschritte nötig um ein bestimmtes Element zu finden!

Binärbäume



Tiefe ist $\log 8 = 3$

||

maximale Pfadlänge

8 Blätter

Beispiel: Sortieren

Gegeben:	eine Folge von Elementen aus einer geordneten Menge: $a = (a_1, a_2, \ldots, a_n)$
Gesucht:	eine sortierte Folge
Antwort:	eine Folge $\bar{a} = (a_{i_1}, a_{i_2}, \ldots, a_{i_n})$ mit den gleichen Elementen und mit $a_{i_1} \leq a_{i_2} \leq a_{i_3} \leq \ldots \leq a_{i_n}$

Beispiel: MERGESORT

1. Zerlegung von a in zwei (fast) gleichgroße Teilfolgen:

$$a' = (a_1, \ldots, a_{\lfloor \frac{n}{2} \rfloor}),$$
$$a'' = (a_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, a_n).$$

2. Sortieren von a' und a'' mit dem gleichen Verfahren zu den Ergebnissen: $\bar{a}', \bar{a}''$.
3. Mischen der sortierten Folgen $\bar{a}', \bar{a}''$ zu einer sortierten Folge $\bar{a}$.

Zeitkomplexität von MERGESORT

…setzt sich zusammen aus der Anzahl der einzelnen Arbeitsschritte in 1.), 2.) und 3.):

1. **Zerlegung:** ein Schritt,
2. **Mischen:** $O(n)$ Schritte,
3. **Sortieren:** Sei $T_A(n)$ die Zeitkomplexität von MERGESORT. $\Rightarrow$ Sortieren der Teilfolgen $\bar{a}'$ und $\bar{a}''$ benötigt jeweils $T_A\left(\frac{n}{2}\right)$.

Kosten als Rekurrenzgleichung

Insgesamt ergibt sich für MERGESORT die Rekursionsgleichung:

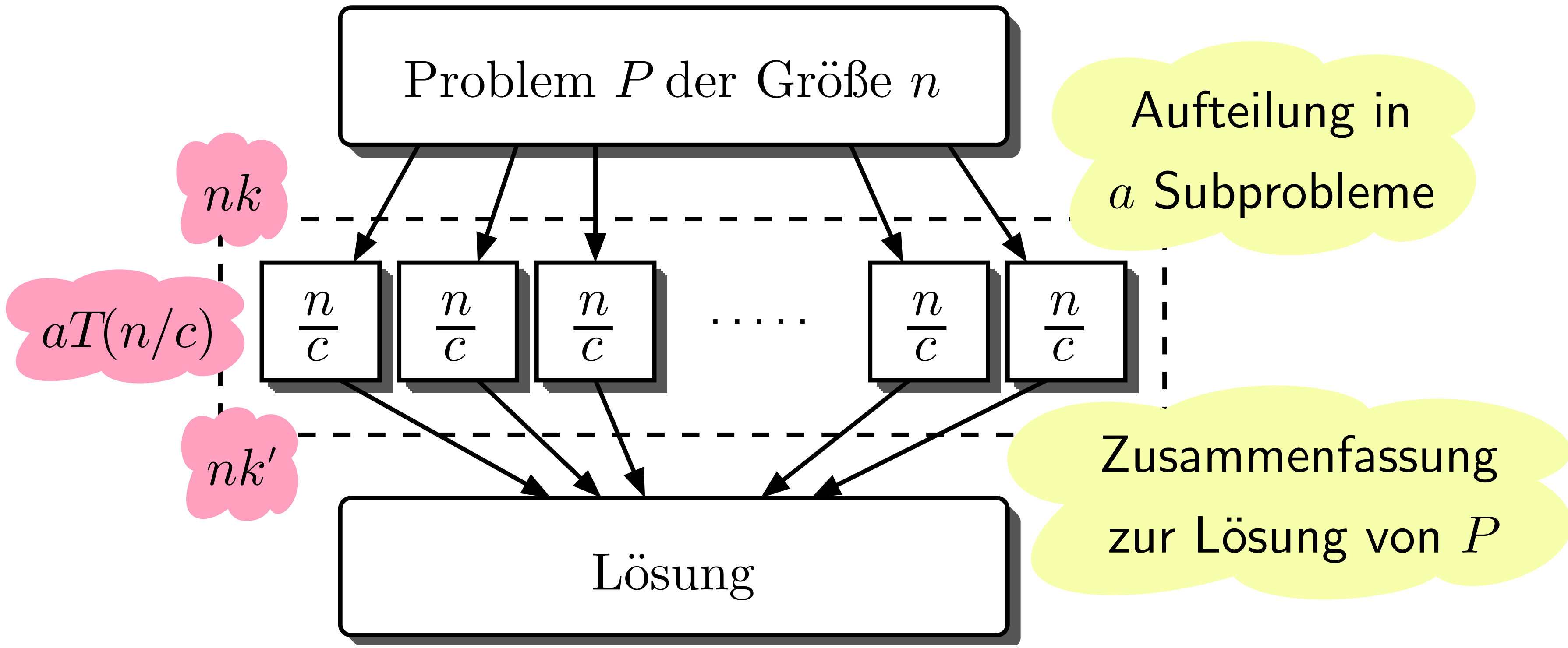
$$T_A(n) = 2 \cdot T_A\left(\frac{n}{2}\right) + O(n),$$

mit der Lösung:

$$T_A(n) = O(n \log n).$$

divide and conquer

MERGESORT arbeitet nach dem *divide and conquer*-Verfahren:

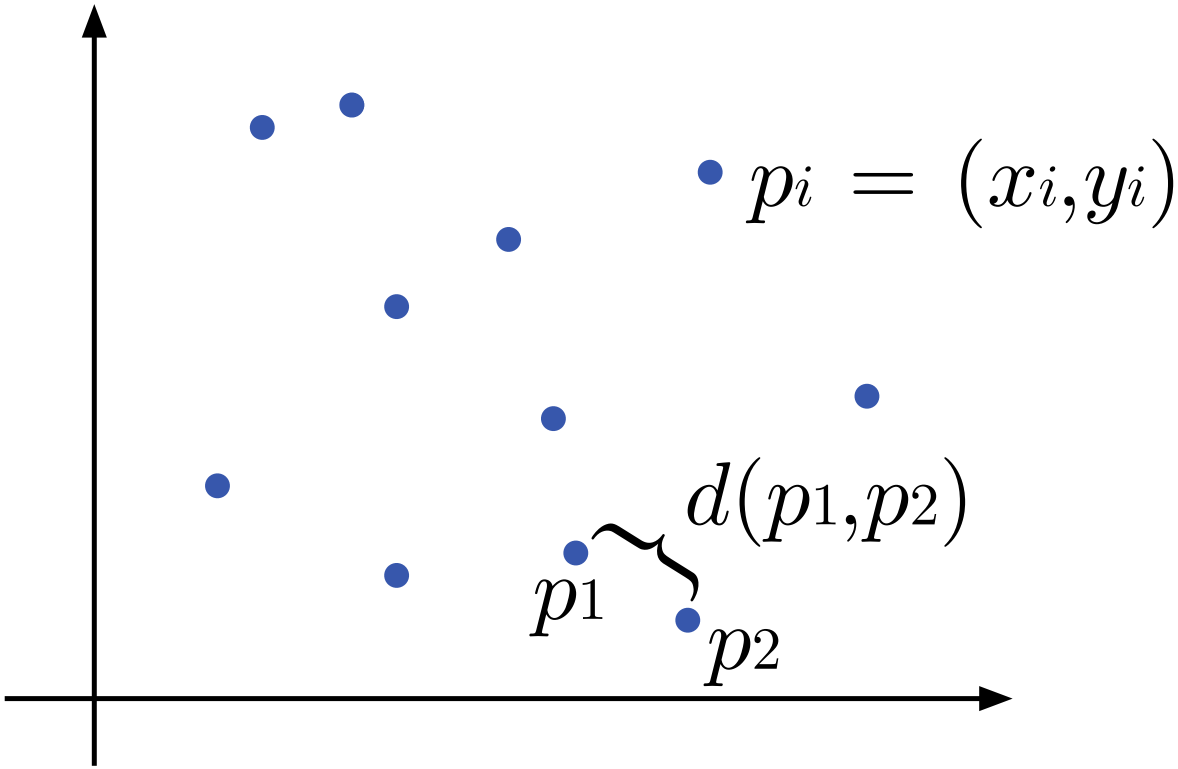


Problem des dichtesten Punktepaares

Gegeben:	eine Menge P von n Punkten in der Ebene: $p_i = (x_i, y_i), \quad i = 1, 2, \ldots, n$
Gesucht:	zwei Punkte minimaler Distanz
Antwort:	Koordinaten der Punkte

- Die **Distanz (Entfernung)** zweier Punkte ist wie folgt definiert:
  $d(p_i, p_k) := \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2}$
- Die Distanz wird in der euklidschen Metrik angegeben.

Distanz von Punkten in der Ebene



Die Punkte liegen nicht sortiert vor!

Eindimensionaler Fall

Gegeben:	Eine Menge P von n Punkten auf einer Geraden: $P = \{p_1, p_2, \ldots, p_n\} = \{x_1, x_2, \ldots, x_n\}$, wobei x_i die Abszisse von p_i ist.
Gesucht:	zwei Punkte minimaler Distanz
Antwort:	Punktkoordinaten

Wir betrachten jetzt einen Algorithmus zur Bestimmung des dichtestes Punktepaars auf einer Linie.

Algorithmus

1. Zerlegung von P in P_1 und P_2 mit Trennungspunkt $x = \alpha$ $(\alpha \notin P)$, so daß $x_i \in P_1 \Leftrightarrow x_i < \alpha$ und $x_j \in P_2 \Leftrightarrow x_j > \alpha$.
2. Bestimmung des jeweils dichtesten Paares in P_1 und P_2:
   Rekursion: δ:= minimale Distanz der beiden Paare, die in P_1 und P_2 gefunden wurden.
3. Sei x^1_{max} größtes Element in P_1 und x^2_{min} kleinstes Element in P_2: $\delta' := \left|x^2_{min} - x^1_{max}\right|$
4. $\delta^* := \min\{\delta, \delta'\}$ ist der dichteste Abstand von Punkten in P.

Ausblick

- Greedy-Algorithmen
- dynamische Programmierung
- Backtracking
- branch and bound
- Heuristiken und Approximation