# Backtracking

by

Dr. Günter Kolousek

# Einführung

- allgemeine Problemlösung für das Finden von Lösungen
  - nicht durch Berechnung
  - sondern durch Versuchen und Nachprüfen (*trial and error*)
- Problem in Teilschritte zerlegen
  - oft rekursiv zu formulieren
  - Problem mit Teilproblemen als Baum darstellbar
  - in diesem Suchbaum ist die Lösung zu suchen
  - Suchbaum in der Regel **sehr** groß!
    - Idee ist: Teilbäume abzuschneiden
- Prinzip
  - Suche in Richtung Ziel
  - zeichne Weg auf
  - stellt sich heraus, dass Weg nicht zielführend (Sackgasse)
    - eingeschlagenen Weg verwerfen und zurückgehen

# Springerproblem – Aufgabenstellung

- geg.: ist ein Schachbrett der Größe $n \times n$
- ges.: Finden eines Weges sodass ein Springer genau einmal über jedes der $n^2$ Felder springt (soferne dies möglich ist)
- Also: Positionieren auf Feld (1, 1) und von dort alle Möglichkeiten durchprobieren.
- Aber: Das sind **sehr** viele Möglichkeiten!

# Analyse der Aufgabenstellung

- Feld als zweidimensionales Array oder Liste

```
n = 5
h = [[0] * n for i in range(n)]
n2 = n ** 2
```

- Wege eines Springers

| | | | | |
|---|---|---|---|---|
| | 3 | | 2 | |
| 4 | | | | 1 |
| | | S | | |
| 5 | | | | 8 |
| | 6 | | 7 | |

- Springen durch Addition von Differenzwerten

```
a = (2, 1, -1, -2, -2, -1, 1, 2)
b = (1, 2, 2, 1, -1, -2, -2, -1)
u = x + a[k]  # 0 <= k < 8
v = y + b[k]
```

# Analyse der Aufgabenstellung – 2

- Bruteforce Algorithmus
  - Setze Springer am Anfang auf Position (1, 1)…
  - Alle Möglichkeiten probieren: Wenn außerhalb oder besetzt, dann verwerfen und nächsten probieren.
  - Wenn alle $n^2$ Felder besucht, dann Lösung gefunden.
- !!! *d.h. alle Möglichkeiten probieren* !!!

# Analyse der Aufgabenstellung – 2

- Bruteforce Algorithmus
  - Setze Springer am Anfang auf Position (1, 1)…
  - Alle Möglichkeiten probieren: Wenn außerhalb oder besetzt, dann verwerfen und nächsten probieren.
  - Wenn alle $n^2$ Felder besucht, dann Lösung gefunden.
- !!! *d.h. alle Möglichkeiten probieren* !!!
  - bei n=5 gibt es 304 Lösungen!
  - bei n=6 gibt es 524.486 Lösungen!!
  - bei n=8 gibt es schon 13.267.364.410.532 Lösungen!!!
  - d.h. praktisch unmöglich!
- Verbesserung mittels Backtracking!

# Analyse der Aufgabenstellung – 2

- Bruteforce Algorithmus
  - Setze Springer am Anfang auf Position (1, 1)…
  - Alle Möglichkeiten probieren: Wenn außerhalb oder besetzt, dann verwerfen und nächsten probieren.
  - Wenn alle $n^2$ Felder besucht, dann Lösung gefunden.
- !!! *d.h. alle Möglichkeiten probieren* !!!
  - bei n=5 gibt es 304 Lösungen!
  - bei n=6 gibt es 524.486 Lösungen!!
  - bei n=8 gibt es schon 13.267.364.410.532 Lösungen!!!
  - d.h. praktisch unmöglich!
- Verbesserung mittels Backtracking!
  - …aber selbst für Schachbretter moderater Größe ist dies sinnlos!

# Analyse der Aufgabenstellung – 3

- Weg nicht zielführend, dann eingeschlagenen Weg verwerfen und zurückgehen $\rightarrow$ Backtracking
- Prinzip:

```
Funktion nächsten Zug versuchen:
  initialisieren der Datenstrukturen
  wiederholen:
    nächsten Zug wählen
    wenn annehmbar:
      Zug aufzeichnen
      wenn brett nicht voll:
        nächsten Zug versuchen
        wenn nicht erfolgreich:
          lösche vorhergehenden Zug
  bis erfolgreich oder keine Züge mehr
```

# Springerproblem – eine Lösung

```
def try_next(i, x, y): # Zug, Pos x, Pos y
    k = 0
    res = False
    while True:
        u = x + a[k]
        v = y + b[k]
        if 1 <= u <= n and 1 <= v <= n:  # zug annehmbar? - 1
            u1 = u - 1
            v1 = v - 1
            if h[u1][v1] == 0:  # zug annehmbar? - 2
                h[u1][v1] = i  # aufzeichnen
                if i < n2:  # brett nicht voll
                    res = try_next(i + 1, u, v)
                    if not res:
                        h[u1][v1] = 0 # nicht erfolgreich: loeschen
                else:  # fertig => erfolgreich
                    res = True
        k += 1
        if res or k == 8: # erfolgreich oder alle Zuege
            break             # besser in Kopf von while!
    return res
```

# Allgemeine Struktur

```
initialisiere Wahl der Kandidaten
wiederholen:
  nächsten Kandidaten wählen
  wenn annehmbar:
    Kandidaten aufzeichnen
    wenn Lösung unvollständig:
      nächsten Schritt versuchen
      wenn nicht erfolgreich:
        lösche Aufzeichnung
bis erfolgreich oder keine weiteren Kandidaten
```

# Implementierung für *eine* Lösung

- Voraussetzungen
  - expliziter Stufenparameter
    - der die Tiefe der Rekursion angibt
    - der eine einfache Bedingung der Terminierung erlaubt ($n$)
  - # der möglichen Kandidaten in jedem Schritt $= m$

# Implementierung für *eine* Lösung – 2

```
def try_next(i):
    k = 0
    res = False
    while True:
        k += 1
        waehle_k_ten_kandidaten()
        if annehmbar():
            if i < n:
                res = try_next(i + 1)
                if not res:
                    loesche_aufzeichnung()
        if res or k == m:
            break
    return res

res = try_next(1)
if res:
    print_loesung()
else:
    print(”Keine Loesung”)
```

# Implementierung für *alle* Lösungen

- ges. alle Lösungen eines Problems
- dann:

```
def try_next(i):
    k = 0
    while True:
        k += 1
        waehle_k_ten_kandidaten()
        if annehmbar():
            if i < n:
                try_next(i + 1)
            else:
                print_solution()
            loesche_aufzeichnung()
        if k == m:
            break

try_next(1)
```

# Springerproblem – Alle Lösungen

```
def try_next(i, x, y):
    k = 0
    while k != 8:  # nicht alle Kandidaten
        u = x + a[k]
        v = y + b[k]
        if 1 <= u <= n and 1 <= v <= n:  # zug annehmbar 1
            u1 = u - 1
            v1 = v - 1
            if h[u1][v1] == 0:  # zug annehmbar 2
                h[u1][v1] = i
                if i < n2:  # brett nicht voll
                    try_next(i + 1, u, v)
                else:  # eine Lösung gefunden
                    print_solution(h)
                h[u1][v1] = 0  # loeschen
        k += 1

h[0][0] = 1  # 1. Zug
try_next(2, 1, 1)
```