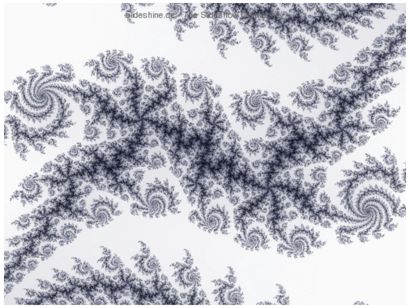
Algorithmen und Datenstrukturen 1

Prof. Dr. Carsten Lecon

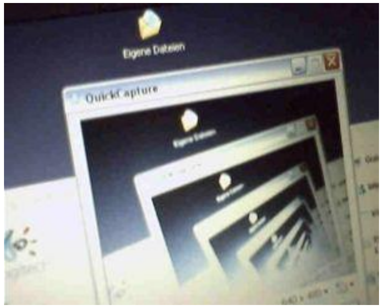
Inhalt

- Rekursion
  - Einführung
  - Beispiel Mergesort (Wdh.)
    - Laufzeitabschätzung: Rekursionsformel
  - Weitere Beispiele
  - Direkte / indirekte Rekursion
  - Backtracking

Rekursion



Bildquelle> http://mathestuff.de



Bildquelle> http://mathestuff.de

- Was ist Rekursion?
  - Lat. *recurrere*
  - Wörtlich: „zurücklaufen“
  - Definition einer Funktion, eines Verfahrens oder eine Datenstruktur durch sich selbst
  - Im Allgemeinen kürzer und leichter verständlich als andere Darstellungen, da sie charakteristische Eigenschaften einer Funktion betonen

Beispiele:

•Mathematische Funktionen

•Suche, Traversierung in Bäumen, Spiele, ...

Bildquelle> http://mathestuff.de

Rekursion

- Beobachtung:
  - Rekursive Definition ist an Bedingung verknüpft
    - Bedingung stellt Terminierung sicher
  - Rekursion erfolgt direkt

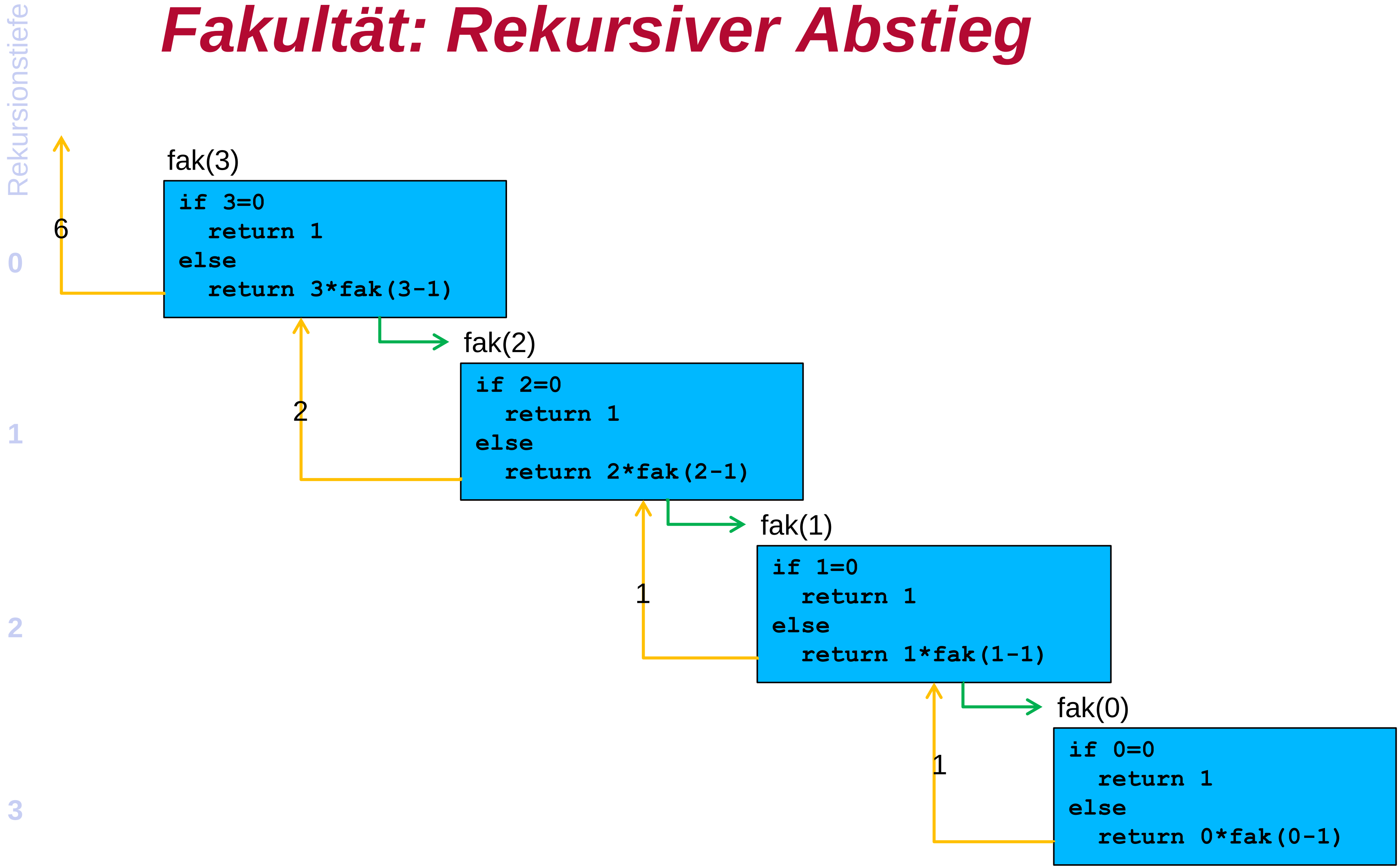
- Beispiel: Berechnung der Fakultät

Rekursion

Pseudocode für die Fakultätsberechnung:

```
function fak(n)
  if n=0 then
    return 1
  else
    return n * fak(n-1)
  end if
end function
```

Fakultät: Rekursiver Abstieg



Ent-Rekursivierung

Iterativer Algorithmus für die Fakultätsberechnung:

$$n! = \begin{cases} 1 & n = 0 \\ \prod_{i=1}^{n} i & n > 0 \end{cases}$$

Ent-Rekursivierung

```
function Fakultät(n)
  f=1
  for i=1 to n do
    f = f*i
  end for
  return f
end function
```

Ent-Rekursivierung

Fibonacci-Zahlen

$\mathbf{f_n = f_{n-1} + f_{n-2}}$, für $\mathbf{n \geq 2}$;
Anfangswerte $\mathbf{f_0=0,\ f_1=1}$

f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}	f_{24}	$f_{...}$
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1.597	2.584	4.181	6.765	10.946	17.711	28.657	46.368	...

Iterativer Algorithmus:

$$F(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

Rekursive Algorithmen

- Gerade gehabt: Mergesort

Divide & Conquer (Pseudocode - 1)

```
function merge(A, p, q, r)
  if Länge(A) ≤ 1 then return
  n1 := q-p+1
  n2 := r-q
  erzeuge Arrays L[1..n1+1] und R[1..n2+1]
  for i=1 to n1 do
    L[i] := A[p+i-1]
  end for
  for j=1 to n2 do
    R[j] := A[q+j]
  end for
  L[n1+1] = ∞
  R[n2+1] = ∞
```

Divide & Conquer (Pseudocode - 2)

```
i := 1
j := 1
for k=p to r do
  if L[i] ≤ R[j]
    A[k] := L[i]
    i := i+1
  else
    A[k] := R[j]
    j := j+1
  end if
end for
```

Divide & Conquer

Algorithmus Mergesort

```
function mergesort(A,p,r)
  if p<r then
    q := |(p+r)/2 |
    mergesort(A,p,q)
    mergesort(A,q+1,r)
    merge(A,p,q,r)
  end if
end function
```

Laufzeitabschätzung: Divide & Conquer

- Allgemeine Laufzeitabschätzung von rekursiven Algorithmen:
- Wenn kleine Problemgröße $n \leq c \rightarrow$ konstante Zeit: $\Theta(1)$
- Sonst:
  - *a* Unterprobleme (Teile von *Divide & Conquer*)
  - Zeitaufwand der Unterprobleme: *1/b* der Zeit des Gesamtaufwands
  - *D(n)*: Aufwand für Teilen (*Divide*)
  - *C(n)*: Aufwand für Zusammenführung (*Combine*)

  - $\rightarrow$ Zeitaufwand: $T(n) = a\, T(n/b) + D(n) + C(n)$

Binäre Suche

Suche in einer geordneten Zahlenfolge

1	2	3	4	5	6	7	8	9	10	11	12
2	4	5	10	12	14	15	17	29	39	40	42

Idee:

1. In der Mitte beginnen
2. Ggf. in der linken oder rechten Hälfte weitersuchen, in der Hälfte erneut die Mitte prüfen
3. etc.

Binäre Suche: Algorithmus

```
function BinäreSuche(A, l, r, x)
  m=(l+r)/2
  if A[m] = x then
    return m
  end if
  if (l=r) then
    return NIL
  end if
  if x>A[m] then
    return BinäreSuche(A, m, r, x)
  else
    return BinäreSuche(A, l, m, x)
  end if
end function
```

Rekursion: Türme von Hanoi

- Erfunden 1883 von Édouard Lucas
- Geschichte:
  - Im großen Tempel zu Benares im Mittelpunkt der Welt, soll ein Turm aus 64 goldenen Scheiben versetzt werden. Wenn dies vollbracht ist, ist das Ende der Welt gekommen.
- Lösung der Mönche:
  - Der älteste Mönch bittet den zweitältesten Mönch, die 63 oberen Scheiben zu versetzen. Er selbst versetzt dann die untere Scheibe.
  - Der zweitälteste Mönch bittet den drittältesten Mönch, die 62 oberen Scheiben zu versetzen. Der Zweitälteste versetzt dann die zweitunterste Scheibe.
- Anzahl Spielzüge für 64 Scheiben:
  - 18.446.744.073.709.551.615
- Zeit (1 Sekunde für jeden Zug):
  - Ca. 584.942.417.400 Jahre
    Vergleich: Unsere Sonne ist ca. 5 Mrd. Jahre alt…

n	T(n)
1	1
2	3
3	7
4	15
…	…

$$T(n) = 2 \cdot T(n-1) + 1 = 2^n - 1 \in O(2^n)$$

Rekursion: Weitere Beispiele

- Ackermann-Funktion:
- `a(0,y) = y+1`
- `a(x,0) = a(x-1,1)`, falls `x > 0`
- `a(x,y) = a(x-1, a(x,y-1))`, falls `x,y > 0`

Werte von $a(n,m)$

n \ m	0	1	2	3	4	m
0	1	2	3	4	5	$m+1$
1	2	3	4	5	6	$m+2$
2	3	5	7	9	11	$2m+3$
3	5	13	29	61	125	$8 \cdot 2^m - 3$
4	13	65533	$2^{65536} - 3 \approx 2 \cdot 10^{19728}$	$a(3, 2^{65536} - 3)$	$a(3, a(4,3))$	$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} - 3$ $(m+3$ Terme$)$
5	65533	$a(4,65533)$	$a(4,a(5,1))$	$a(4,a(5,2))$	$a(4,a(5,3))$	
6	$a(5,1)$	$a(5,a(5,1))$	$a(5,a(6,1))$	$a(5,a(6,2))$	$a(5,a(6,3))$	

Quelle: Wikipedia

Direkte / indirekte Rekursion

- **Direkte Rekursion**: Funktion ruft sich selbst auf
- **Indirekte Rekursion**: Funktion ruft eine andere Funktion auf, die wiederum die erste Funktion aufruft
  - Sollte möglichst nicht verwendet werden, da mitunter sehr unübersichtlich (und dadurch fehlerträchtig)

Direkte Rekursion

```
function f
  if Rekursionsbedingung then
    f
  end if
  …
end function
```

Indirekte Rekursion

```
function f
  if Rekursionsbedingung then
    g
  end if
  …
end function
```

```
function g
  if Rekursionsbedingung then
    f
  end if
  …
end function
```

Inhalt

- Rekursion
  - Einführung
  - Beispiel Mergesort (Wdh.)
    - Laufzeitabschätzung: Rekursionsformel
  - Weitere Beispiele
  - Direkte / indirekte Rekursion
  - Backtracking

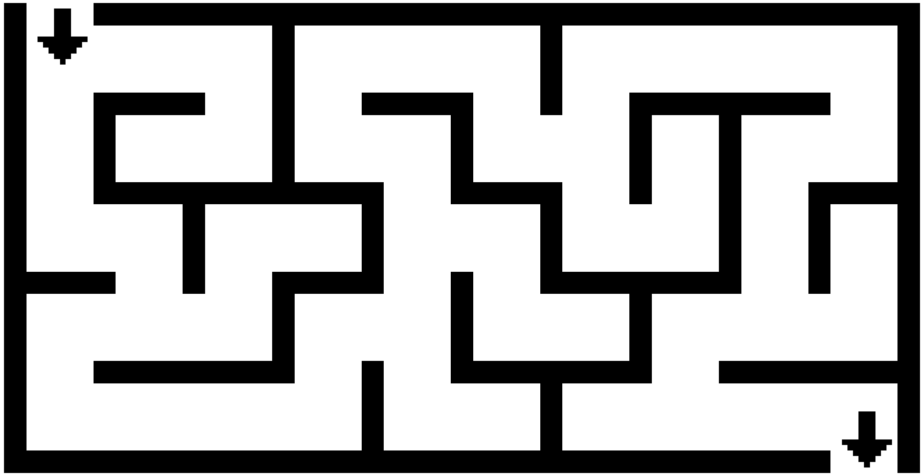
Backtracking

1. Vorgehen
2. Beispiele
   – Irrgarten
   – Springerproblem
   – 8-Damenproblem
3. Abschluss

Backtracking

- Prinzip: „Versuch und Irrtum“ (*trial and error*)

- Beispiel: Irrgarten



Backtracking: Vorgehen

- In jedem Schritt:
  - Alle Möglichkeiten ermitteln
  - Entscheidung für eine Möglichkeit
  - Entscheidung ausprobieren, d.h. nächsten Schritt bearbeiten
  - Solange nicht erfolgreich und es noch Möglichkeiten gibt: (*)
    - Entscheidung für eine noch unversuchte Möglichkeit
    - Neue Entscheidung ausprobieren
  - Sonst:
    - Letzte Entscheidung rückgängig machen, weiter bei (*)
    - Wenn keine Entscheidung mehr möglich → keine Lösung

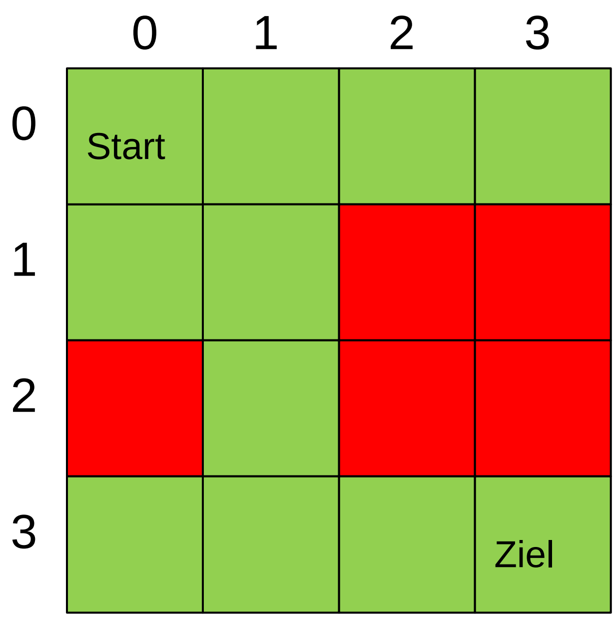
Backtracking: Bemerkungen

- Im *worst case* werden alle Möglichkeiten ermittelt → mitunter hohe Laufzeiten
- Für die Bestimmung der optimalen Lösung müssen alle Möglichkeiten ausprobiert werden und dann bewertet werden
- Backtracking lässt sich gut rekursiv lösen

Backtracking: Algorithmus

```
function findeLösung(Schritt, Lösung)
  Möglichkeiten in Schritt ermitteln
  while es gibt noch offene Möglichkeiten do
    Lösung um gewählte Möglichkeit erweitern
    if Lösung ist vollständig then
      Lösung ausgeben
      HALT
    else
      findeLösung(Schritt+1, Lösung)
      (keine Lösung gefunden)
      Lösungsschritt rückgängig machen
    end if
  end while
end function
```

Backtracking: Beispiel Irrgarten




```
private char feld[][] = {{'·', '·', '·', '·'},
                         {'·', '·', 'X', 'X'},
                         {'X', '·', 'X', 'X'},
                         {'·', '·', '·', '·'}};
```

```
····
··XX
X·XX
····
```

Backtracking: Beispiel Irrgarten

- Zielbedingung?
  - Ziel-Zelle ist erreicht

- Wenn Zielbedingung nicht erfüllt → Lösung erweitern
  - Alle Richtungen ausprobieren

Backtracking: Beispiel Irrgarten

- **`function`** `findeLösung(Schritt, Lösung)`
  - → `wegsuche((x,y), bisheriger Weg)`
- Ist Lösung vollständig?
  - → Zielzelle ist erreicht
- Lösung um gewählte Möglichkeit erweitern
  - → Aktuelle Lösung speichern (Zelle markieren)
  - → in alle Richtungen weitergehen
  - → `findeLösung((x´,y´), bisheriger Weg)`

Backtracking: Beispiel Irrgarten

- Wegsuche (Initialprüfung)

```
public void findeWeg(int x, int y,
        Vector<Integer> wegX, Vector<Integer> wegY) {
    char zelle;

    // Korrekte Grenzen?:
    if ((x<0) || (x>=feld[0].length)
            || (y<0) || (y>=feld[0].length)) {
        return;
    } // if

    // Schritt erlaubt?:
    if (( feld[x][y] == 'X') || (feld[x][y]==MARK)) {
        return;
    } // if
```

Backtracking: Beispiel Irrgarten

- Wegsuche (Weg merken)

```
// Wegschritt merken:
wegX.add(x);
wegY.add(y);
zelle = feld[x][y]; // alten Inhalt sichern
feld[x][y] = MARK; // als besucht markieren
```

Backtracking: Beispiel Irrgarten

- Wegsuche (Lösung um weitere Möglichkeiten erweitern)

```
findeWeg(x-1, y, wegX, wegY);
findeWeg(x+1, y, wegX, wegY);
findeWeg(x, y-1, wegX, wegY);
findeWeg(x, y+1, wegX, wegY);
```

Backtracking: Beispiel Irrgarten

- Wegsuche (Lösung vollständig?)

```
if ((x==ZIEL_X) && (y==ZIEL_Y)) {
    anzahlVersuche++;
    ausgabeWeg(wegX, wegY, true);
    feld[x][y] = zelle; // fuer weitere Versuche
    return;
} // if
```

Backtracking: Beispiel Irrgarten

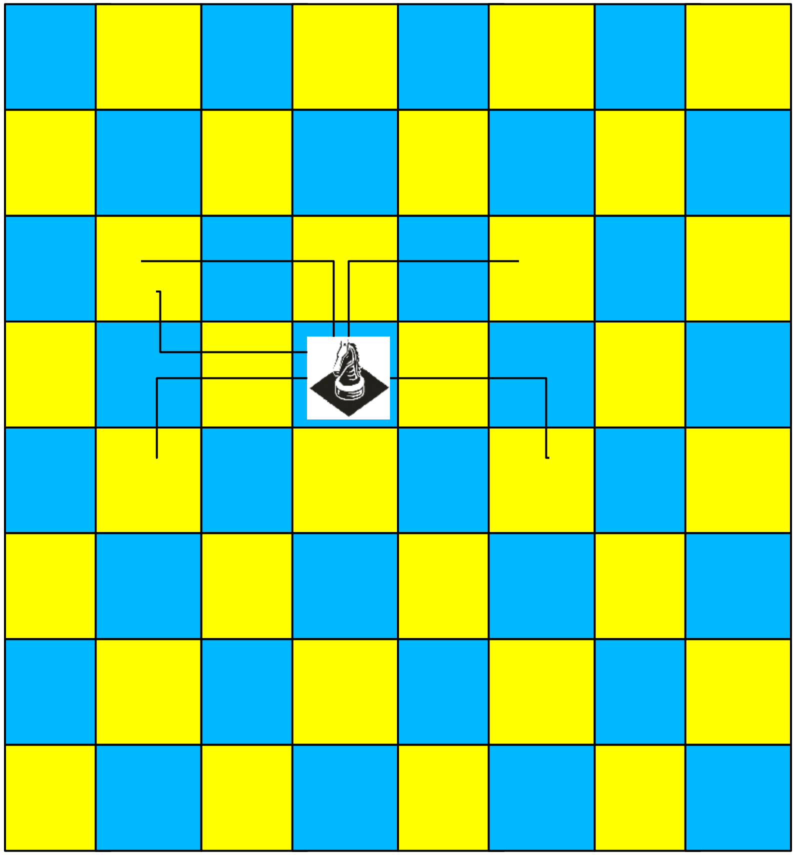
- Wegsuche (Irrweg: Sackgasse)

```
// War wohl nichts:
anzahlVersuche++;
ausgabeWeg(wegX,wegY,false);
feld[x][y] = zelle;
// Letztes Teilstueck entfernen:
wegX.remove(wegX.size()-1);
wegY.remove(wegY.size()-1);
```

Backtracking

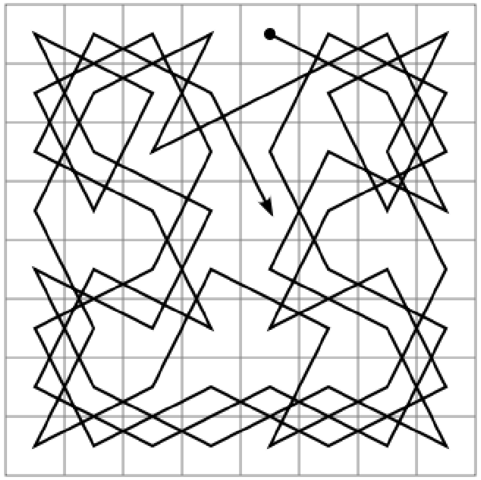
- Laufzeit von Backtracking?
  (**z**: Anzahl Möglichkeiten, **N**: maximale Tiefe)
  - $1+z+z^2+z^3+\ldots+z^N$
  - $O(z^N)$

Backtracking: Beispiel Springerproblem



Ziel: Alle Felder müssen (genau einmal) besucht werden.

Mögliche Bewegungen:
- 2 Felder vor/zurück und 1 Feld links/rechts
- 1 Feld vor/zurück und 2 Felder links/rechts



Backtracking: Beispiel Springerproblem

- Zielbedingung?
  - Alle Felder wurden besucht

- Wenn Zielbedingung nicht erfüllt → Lösung erweitern
  - Alle Springer-Möglichkeiten ausprobieren

Backtracking: Beispiel Springerproblem

- Zielbedingung?

```
if (getAnzahlMarkierungen(true) == (feld.length * feld.length)) {
    anzahlVersuche++;
    ausgabeWeg(wegX, wegY, true);
    System.out.println("FERTIG! Das Programm wird beendet.");
    endZeit = new Date();
    System.out.println( "Endzeit: " + df.format( endZeit ) );
    laufZeit = new Date(endZeit.getTime()-startZeit.getTime());
    System.out.println("Laufzeit: " + df.format(laufZeit));
    System.exit(0);
    feld[x][y] = zelle; // fuer weitere Versuche
    return;
} // if
```

```
private int getAnzahlMarkierungen(boolean tiefe) {
    int anzahl = 0;
    for (int i = 0; i < feld.length; i++) {
        for (int j = 0; j < feld[i].length; j++) {
            if (feld[i][j] != ' ') {
                anzahl++;
            }
        } // for (j)
    } // for(i)
    return anzahl;
} // getAnzahlMarkierungen
```

Backtracking: Beispiel Springerproblem

- Wenn Zielbedingung nicht erfüllt → Lösung erweitern
  - Alle Springer-Möglichkeiten ausprobieren

```
tiefe += 1;
findeWeg(x-1, y-2, wegX, wegY, tiefe);
findeWeg(x-1, y+2, wegX, wegY, tiefe);
findeWeg(x+1, y-2, wegX, wegY, tiefe);
findeWeg(x+1, y+2, wegX, wegY, tiefe);
findeWeg(x+1, y-2, wegX, wegY, tiefe);
findeWeg(x+2, y-1, wegX, wegY, tiefe);
findeWeg(x+2, y+1, wegX, wegY, tiefe);
findeWeg(x-2, y-1, wegX, wegY, tiefe);
findeWeg(x-2, y+1, wegX, wegY, tiefe);
```

Backtracking: Beispiel Springerproblem

- Mögliche Lösung:

3	6	11	18	1
12	17	2	5	10
7	4	13	22	19
16	21	24	9	14
25	8	15	20	23

Backtracking: Beispiel Springerproblem

- Laufzeit (Windows 7, 64Bit, 2,67 Ghz, 3 GB RAM)

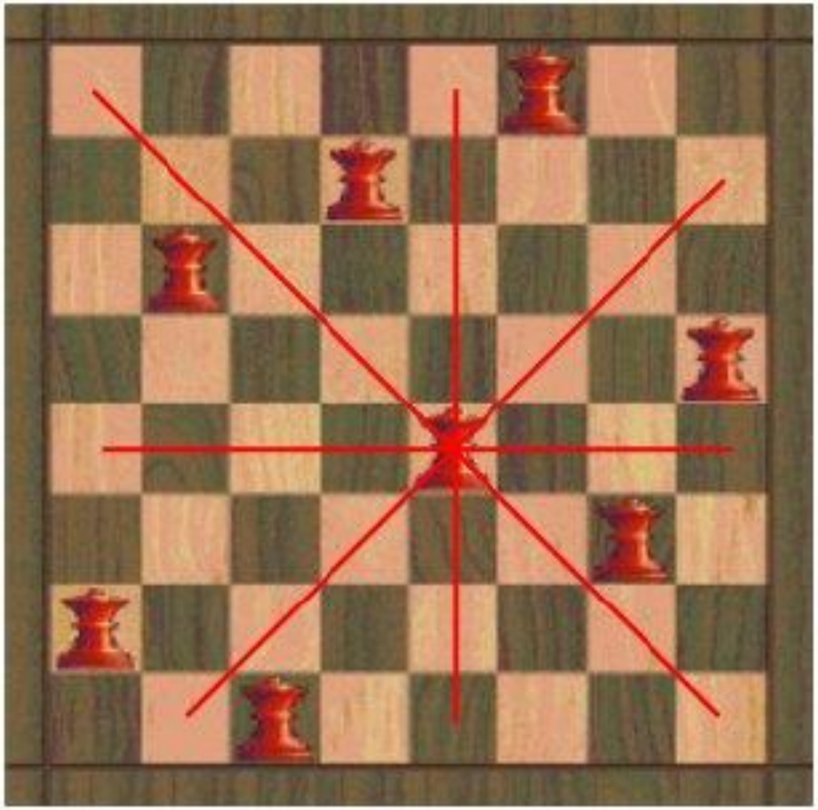
Feldgröße	Anzahl Versuche	Zeit [s]
5x5	8.382	0,021
6x6	21.153.133	0,491
7x7	?	(Stunden?)
8x8	?	(voraus. Tage)

Backtracking: 8-Damenproblem

- Problemstellung:
  - N Damen auf einem NxN-Spielfeld so positionieren, dass sie sich nicht gegenseitig bedrohen

- Klassiker:
  - N=8 → 8-Damenproblem

Backtracking: 8-Damenproblem



modifiziert nach Prof. Sauer (Fachhochschule Regensburg)

Backtracking: 8-Damenproblem

- Wie viele Lösungen gibt es?
  - 8-Damenproblem hat 92 verschiedene Lösungen, bzw. 12 eindeutige
- Bekannt ist Zahl der Lösungen bis 26-Damenproblem:
  - ~ $22 \cdot 10^{15}$ Lösungen (eindeutige: ~$2{,}8 \cdot 10^{15}$)
- Zahl der Lösungen wächst etwas schneller als exponentiell mit der Zahl der Damen
- Webquellen:
  - de.wikipedia.org/wiki/Damenproblem
  - queens.inf.tu-dresden.de
  - www.durangobill.com/N_Queens.html

Backtracking: 8-Damenproblem

N	Lösungen	Eindeutige Lösungen
1	1	1
2	0	0
3	0	0
4	2	1
5	10	2
6	4	1
7	40	6
8	92	12
9	352	46
10	724	92
11	2.680	341
12	14.200	1.787
13	73.712	9.233

Backtracking: 8-Damenproblem

- Lösungsstrategie (naiv):
  - Vollständige Enumeration aller möglichen Platzierungen
  - Es gibt ganz viele Lösungen:
    - $64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57 =$ $178.462.987.637.760 \sim 1{,}78 \cdot 10^{14}$ Möglichkeiten
    - Bei 1000 Tests/ Sekunde: $\sim 10^{11}$ Sekunden $\sim 1.157.407$ Tage

Backtracking: 8-Damenproblem

- Lösungsstrategie (besser):
  - Beobachtung: Jede Zeile und jede Spalte darf nur genau eine Dame enthalten.
  - Vollständige Enumeration aller möglichen Platzierungen der Damen unter Berücksichtigung dieser Beobachtung
  - Es gibt nun n! = 40.320 Möglichkeiten (n=8).
  - Dies ist die Basis für eine schrittweise Lösung.

Backtracking: 8-Damenproblem

- Lösungsstrategie (besser – Forts.):
  - Für jede Spalte genau eine Dame vorsehen
  - Je Spalte Damenposition speichern
  - Dann spaltenweise Damen in richtige Position setzen:
    - Nacheinander in jeder Spalte eine Dame an eine erlaubte Position setzen
    - Im Fall einer Sackgasse (es findet sich keine Lösung, die mit der bisherige Konstellation passt): Vorangegangene Positionierung revidieren (Backtracking)

Backtracking: 8-Damenproblem

- Spielfeld:

	1 a	2 b	3 c	4 d	5 e	6 f	7 g	8 h
1						D		
2				D				
3		D						
4								D
5					D			
6							D	
7	D							
8			D					

Repräsentation: **`spaltenfeld`** (1-dim-Array): {7,3,8,2,5,1,6,4}

Backtracking: 8-Damenproblem - Algorithmus

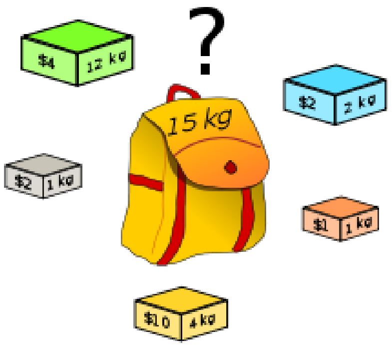
```
procedure NDamen(spalte, spaltenfeld)
  for zeile=1 to spaltenfeld.length do
    spaltenfeld[spalte] = zeile
    if DameErlaubt(spalte, spaltenfeld) then
      if spalte=spaltenfeld.length then
        LoesungAusgeben(spaltenfeld)
        HALT
      else
        NDamen(spalte+1, spaltenfeld)
      end if
    end if
    spaltenfeld[spalte] =0
  end for
end procdure
```

Backtracking: 8-Damenproblem - Algorithmus

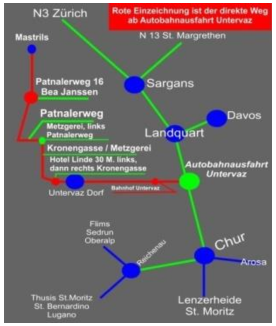
```
function DameErlaubt(spalte, spaltenfeld)
  neueDame = spaltenfeld[spalte]
  for i=1 to spalte-1 do
    if spaltenfeld[i] = neueDame or
      spaltenfeld[i] + spalte - i = neueDame or
      spaltenfeld[i] - spalte + i = neueDame then
      return false
    end if
  end for
  return true
end function
```

Backtracking

- Weitere Beispiele:
  - Rucksackproblem
  - Färbeproblem
  - Brettspiele
  - Wegsuche
  - Sudoku



							1	2
				3	5			
			6				7	
7						3		
			4			8		
1								
			1	2				
	8						4	
	5					6		



Bildquellen: http://upload.wikimedia.org, http://www.spiegel.de, http://www.rueckfuehrungen1.ch

Backtracking: Exkurs PROLOG

- PROLOG: wichtigste logische Programmiersprache (seit 1970)
- Datenbasis mit Fakten und Regeln
- Es lassen sich interaktiv Fragen stellen, Auswertung erfolgt mittels Backtracking.

Backtracking: Exkurs PROLOG

- Einsteins Rätsel
- Problem:
  - Es gibt fünf Häuser mit je einer anderen Farbe
  - In jedem Haus wohnt eine Person mit einer anderen Nationalität
  - Jeder Hausbewohner bevorzugt ein bestimmtes Getränk, raucht eine bestimmte Zigarettenmarke und hält ein bestimmtes Haustier
  - Keiner der fünf Personen trinkt das gleiche Getränk, raucht die gleiche Zigarettenmarke und hält das gleiche Haustier wie seine Nachbarn.
- Frage: Wem gehört der Fisch?

Backtracking: Exkurs PROLOG

- Einsteins Rätsel – Hinweise:

- Der Brite lebt im roten Haus.
- Der Schwede hält einen Hund.
- Der Däne trinkt gern Tee.
- Das grüne Haus steht direkt links neben dem weißen Haus.
- Der Besitzer des grünen Hauses trinkt Kaffee.
- Die Person, die Pall Mall raucht, hält einen Vogel.
- Der Mann, der im mittleren Haus wohnt, trinkt Milch.
- Der Besitzer des gelben Hauses raucht Dunhill.
- Der Norweger wohnt im ersten Haus.
- Der Marlboro-Raucher wohnt neben dem, der eine Katze hält.
- Der Mann, der ein Pferd hält, wohnt neben dem, der Dunhill rau
- Der Winfield-Raucher trinkt gern Bier.
- Der Norweger wohnt neben dem blauen Haus.
- Der Deutsche raucht Rothmans.
- Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt.

Quelle: Wikipedia

Backtracking: Exkurs PROLOG

- Einsteins Rätsel – Lösung:

```
run :-
    X = [_,_,_,_,_],                                  /* Es gibt (nebeneinander) 5 (noch unbekannte) Häuser */
    member([rot,brite,_,_,_],X),                      /* Der Brite lebt im roten Haus */
    member([_,schwede,_,_,hund],X),                   /* Der Schwede hält einen Hund */
    member([_,daene,tee,_,_],X),                      /* Der Däne trinkt gern Tee */
    links([gruen,_,_,_,_],[weiss,_,_,_,_],X),         /* Das grüne Haus steht links vom weißen Haus */
    member([gruen,_,kaffee,_,_],X),                   /* Der Besitzer des grünen Hauses trinkt Kaffee */
    member([_,_,_,pallmall,vogel],X),                 /* Die Person, die Pall Mall raucht, hält einen Vogel */
    mittleres([_,_,milch,_,_],X),                     /* Der Mann, der im mittleren Haus wohnt, trinkt Milch */
    member([gelb,_,_,dunhill,_],X),                   /* Der Besitzer des gelben Hauses raucht Dunhill */
    erstes([_,norweger,_,_,_],X),                     /* Der Norweger wohnt im 1. Haus */
    neben([_,_,_,marlboro,_],[_,_,_,_,katze],X),      /* Der Marlboro-Raucher wohnt neben dem, der eine Katze hält */
    neben([_,_,_,_,pferd],[_,_,_,dunhill,_],X),       /* Der Mann, der ein Pferd hält, wohnt neben dem, der Dunhill raucht */
    member([_,_,bier,winfield,_],X),                  /* Der Winfield-Raucher trinkt gern Bier */
    neben([_,norweger,_,_,_],[blau,_,_,_,_],X),       /* Der Norweger wohnt neben dem blauen Haus */
    member([_,deutsche,_,rothmans,_],X),              /* Der Deutsche raucht Rothmans */
    neben([_,_,_,marlboro,_],[_,_,wasser,_,_],X),     /* Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt */
    member([_,N,_,_,fisch],X),                        /* Der mit der Nationalität N hat einen Fisch */
    write(X),nl,                                      /* Ausgabe aller Häuser */
    write('Der '),write(N),write(' hat einen Fisch als Haustier.'),nl.   /* Antwort auf die Frage */
```

Lösung: Der Fisch gehört dem Deutschen im grünen Haus, der Kaffee trinkt und Rothmanns raucht.

Quelle: Wikipedia

Backtracking - Zusammenfassung

- Backtracking ist ein meist intuitives Lösungsverfahren nach dem „Trial & Error“-Verfahren.
- Häufig wird nicht spontan die optimale Lösung gefunden.
- Backtracking lässt sich gut rekursiv lösen.
- Die Laufzeit ist meist exponentiell.
- Ungünstige Zeitkomplexität kann durch Heuristiken verringert werden.
- Es existieren rekursive und iterative Implementierungen.
- Die Programmiersprache PROLOG enthält Backtracking als festen Bestandteil.