6 Backtracking

Lernziele

- Was ist Backtracking?
- Anwendungen
- Beispiele

Einführung

Prinzip

- Problemlösungsmethode nach dem Prinzip „Versuch und Irrtum“ (trial and error)
- Es wird versucht, eine Teillösung zu einer Gesamtlösung zu erweitern
- Führt eine Teillösung nicht weiter, so wird der letzte Schritt zurückgenommen
- Ergebnis: Es werden alle Lösungsmöglichkeiten errechnet
- Backtracking wird oft rekursiv gelöst
- Laufzeit ist hoch $T(n) \in O(k^n)$

Grundalgorithmus

```
1.  function FINDELOESUNG(Stufe, Lösung)
2.      while es existieren noch Teillösungen do
3.          Wähle einen neuen Lösungsschritt
4.          if Lösungsschritt ist gültig then
5.              Erweitere Lösung um den Lösungsschritt
6.              if Lösung ist vollständig then return true
7.              else if FINDELOESUNG(Stufe+1, Lösung) then return true
8.              else Mache Lösungsschritt rückgängig
9.              end if
10.         end if
11.     end while
12.     return false
13. end function
```

Beispiel: Labyrinth 1

Wegesuche im Labyrinth

Frage: Wie komme ich in einem Labyrinth zum Ausgang?

Antwort: Ich suche systematisch alle möglichen Wege ab.

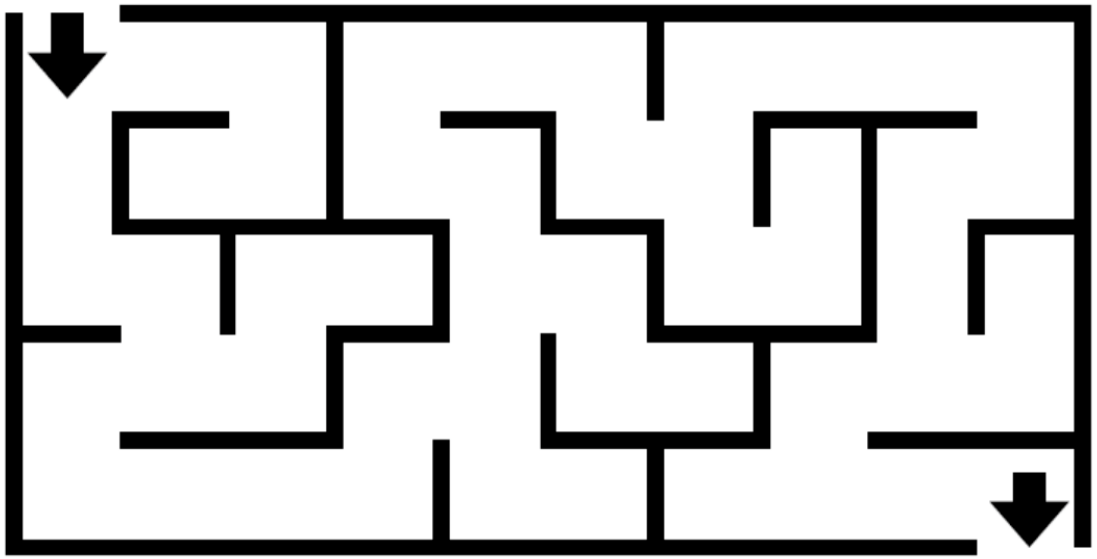
Das beinhaltet, dass

- Wege markiert werden, die schon einmal gegangen wurden
- an einer Kreuzung eine Systematik für die Wegwahl existiert
- eine Rückkehr einkalkuliert wird, falls es nicht mehr weitergeht

Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

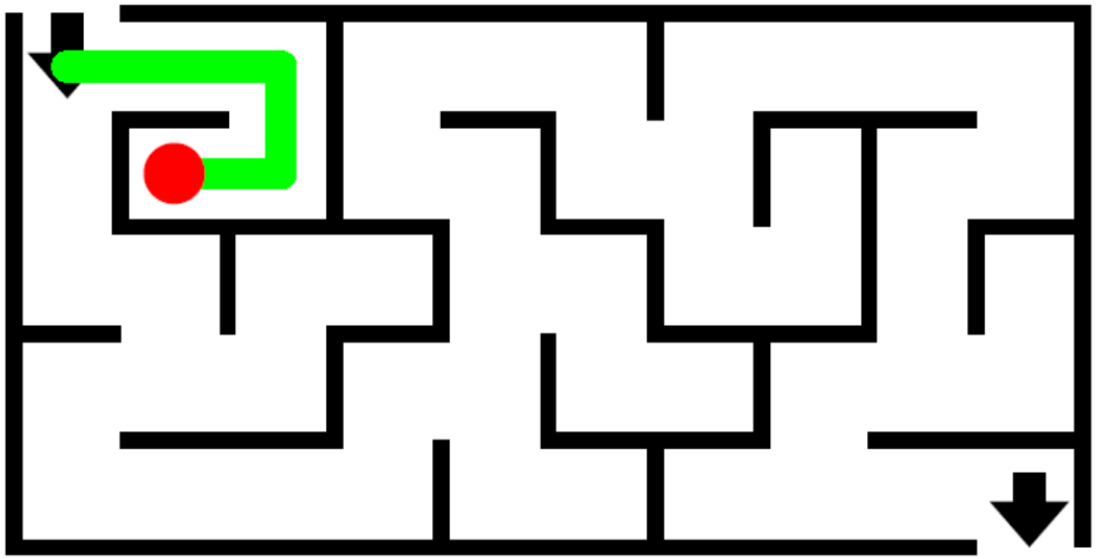
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

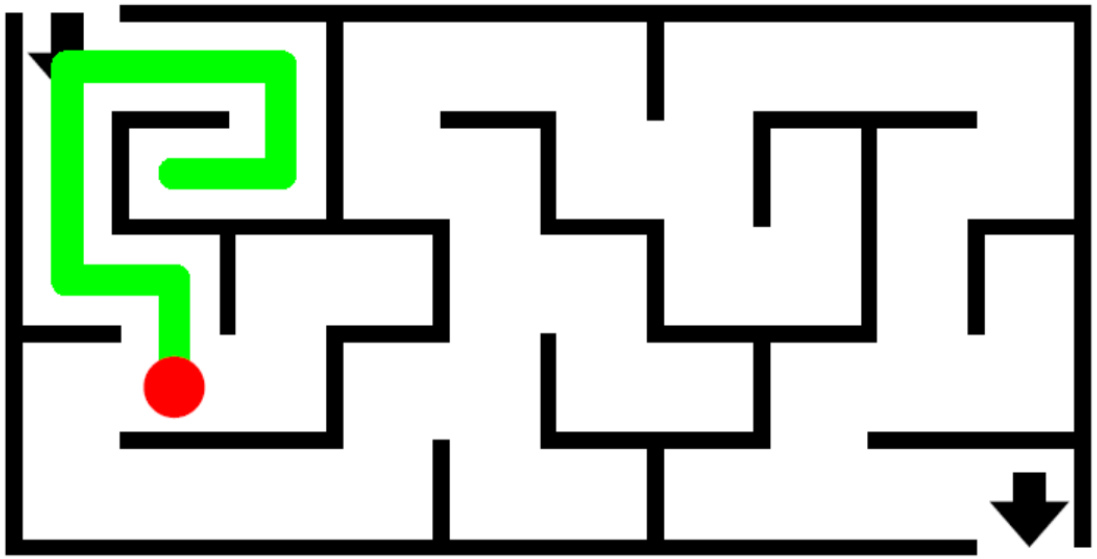
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

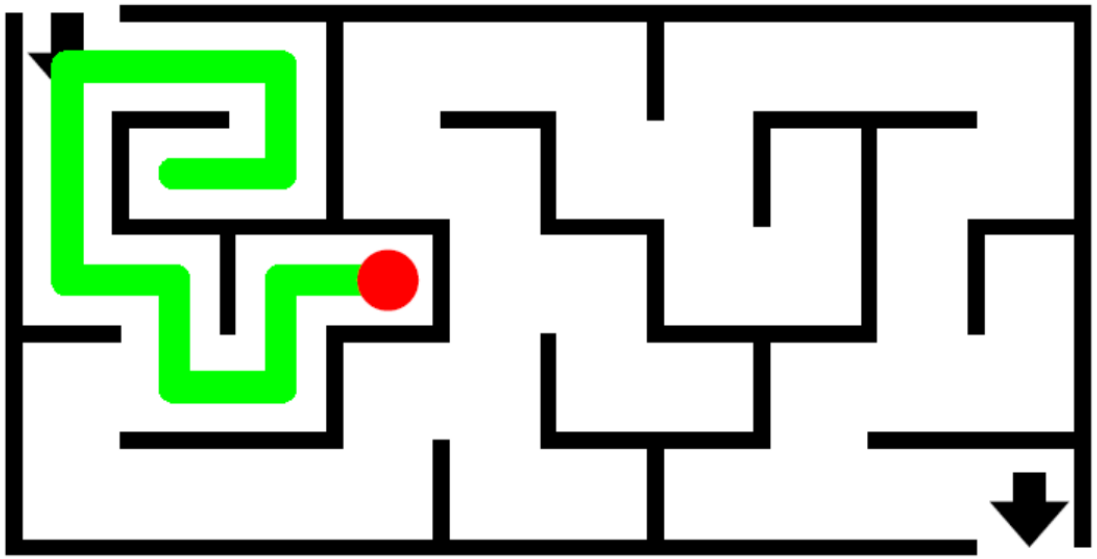
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

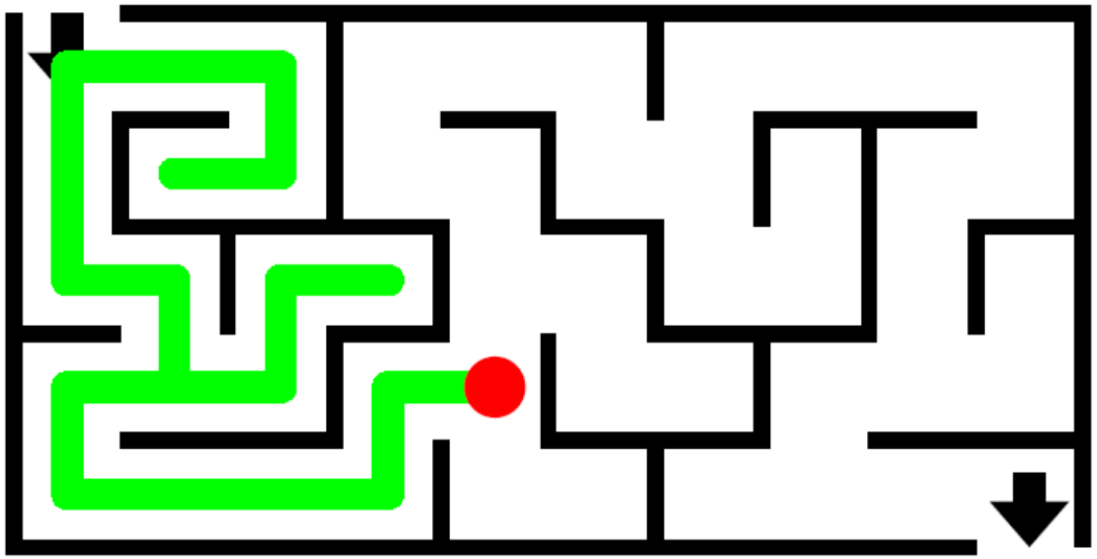
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

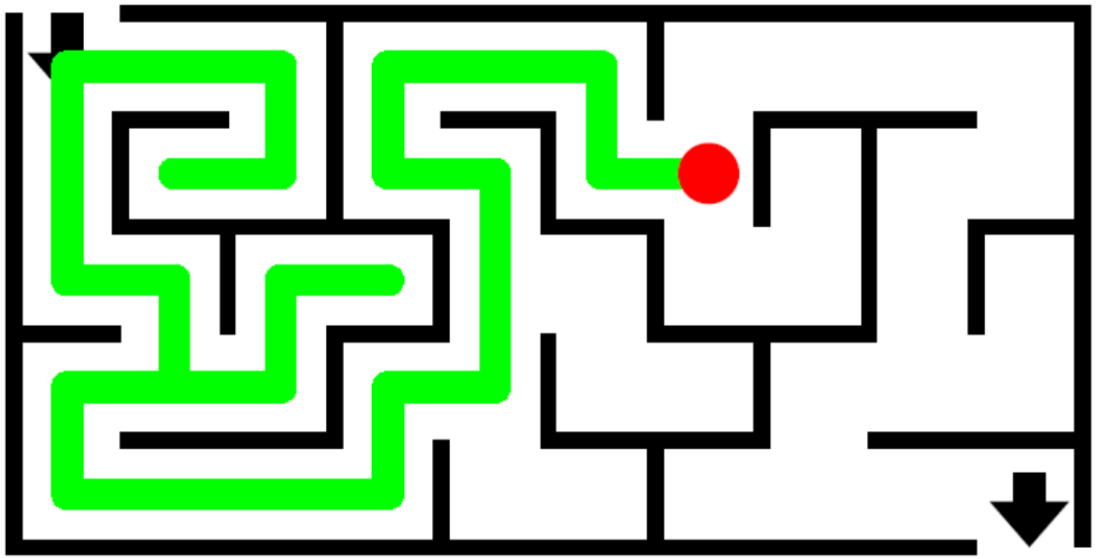
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

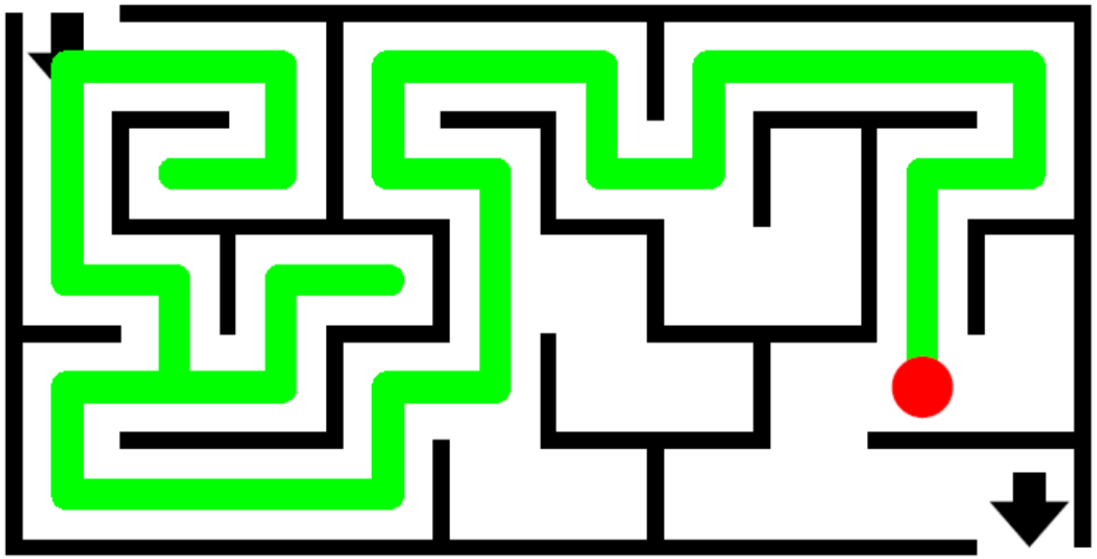
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

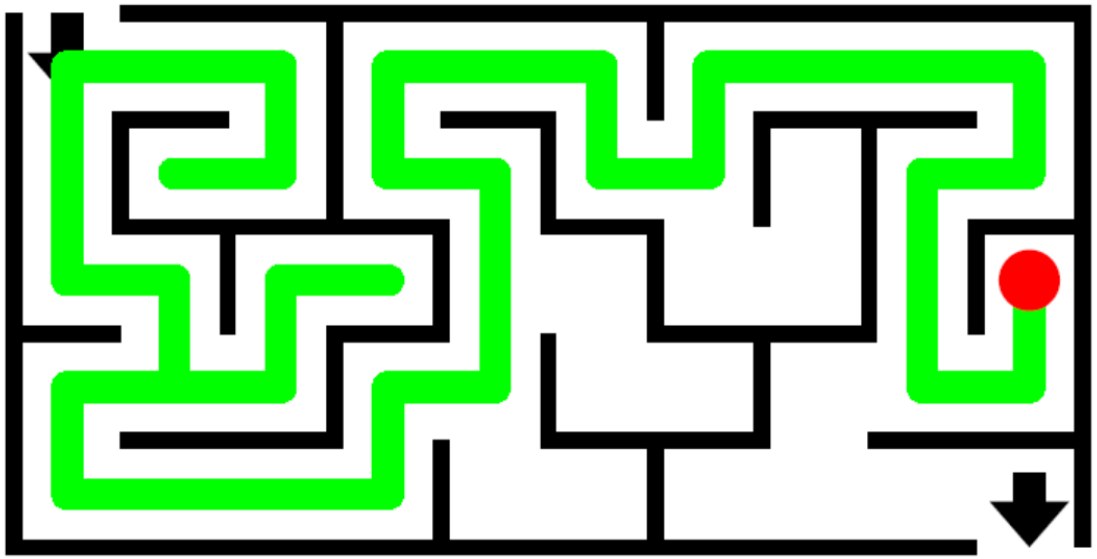
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

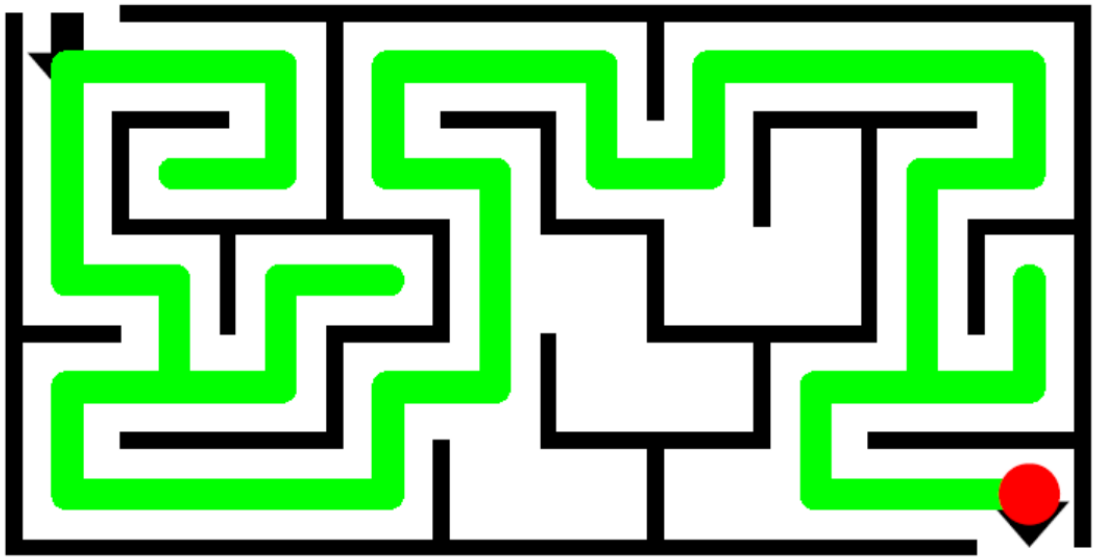
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 1

Wegesuche im Labyrinth: Finde einen Weg zum Ausgang

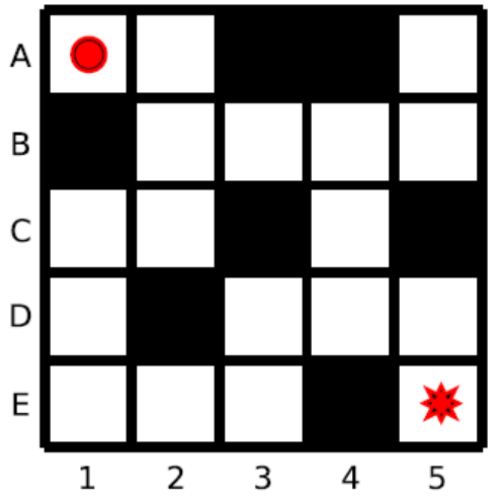
- Strategie an einer Abzweigung
- Vorzugsweise nach links gehen, dann geradeaus, dann rechts.



Beispiel: Labyrinth 2

Reduktion des Problems auf einfache Irrgärten:

- Der Irrgarten ist eine Matrix
- Hindernisse werden z.B. durch einen Eintrag „1“ markiert
- Das Ziel wird z.B. durch einen Eintrag „9“ markiert



$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 9 \end{pmatrix}$$

Beispiel: Labyrinth 2

```
16  void bewege(int zeile, int spalte, int zug);
17
18  int main (int argc, char * const argv[])
19  {
20      bewege(0, 0, 0);
21
22      return 0;
23  }
```

Beispiel: Labyrinth 2

```
25  void bewege(int zeile, int spalte, int zug)
26  {
27      if ((zeile  < 0) || (zeile  >= N_ZEILEN))  return;
28      if ((spalte < 0) || (spalte >= N_SPALTEN)) return;
29
30      if (irrgarten[zeile][spalte] != 1)
31      {
32          if (irrgarten[zeile][spalte] == 9)
33          {   /* Loesung ausdrucken */
41          }
42          else
43          {   /* Weitere Zuege versuchen */
51          }
52      }
53  }
```

Beispiel: Labyrinth 2

```
32          if (irrgarten[zeile][spalte] == 9)
33          {   /* Loesung ausdrucken */
34
35
36                  sprintf(pfad[zug], "%c%d", 'A'+zeile, spalte+1);
37                  for (int i=0; i<=zug; i++)
38                      printf("%s ", pfad[i]);
39                  printf("\r\n");
40
41          }
```

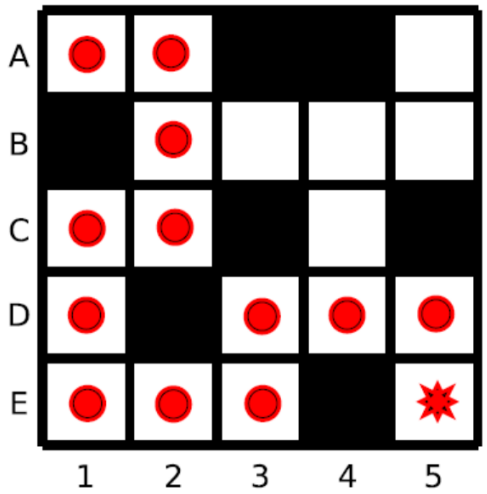
Beispiel: Labyrinth 2

```
42          else
43          {   /* Weitere Zuege versuchen */
44              irrgarten[zeile][spalte] = 1;
45              sprintf(pfad[zug], "%c%d", 'A'+zeile, spalte+1);
46              bewege(zeile+1, spalte, zug+1);
47              bewege(zeile-1, spalte, zug+1);
48              bewege(zeile, spalte+1, zug+1);
49              bewege(zeile, spalte-1, zug+1);
50              irrgarten[zeile][spalte] = 0;
51          }
```

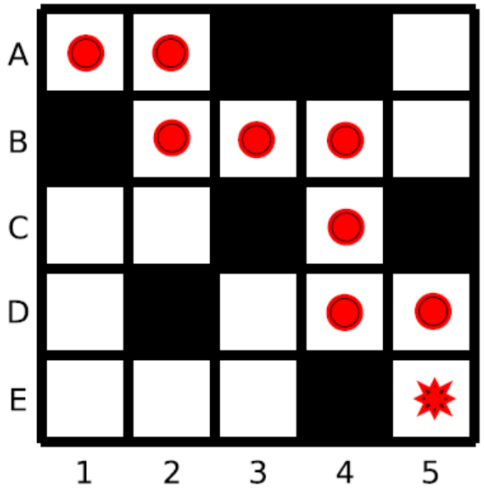
Beispiel: Labyrinth 2

Programmausgabe

A1 A2 B2 C2 C1 D1 E1 E2 E3 D3 D4 D5 E5



A1 A2 B2 B3 B4 C4 D4 D5 E5



Laufzeit

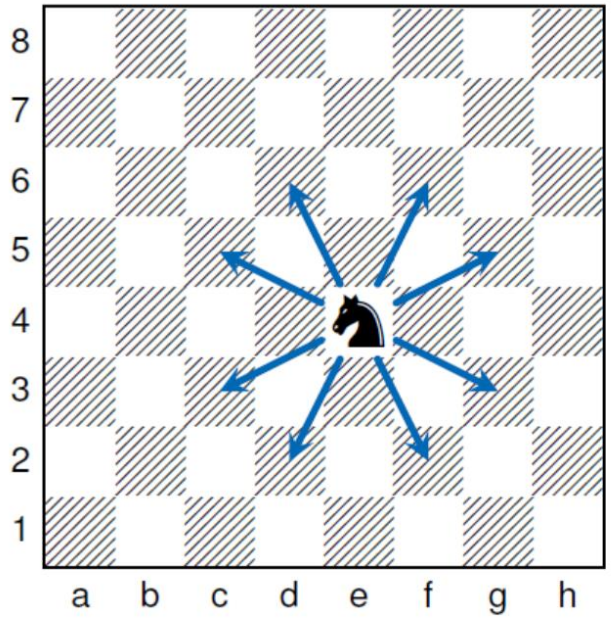
Hat man in jedem Schritt des Backtracking k Verzweigungsmöglichkeiten, so kann die Laufzeit abgeschätzt werden mit

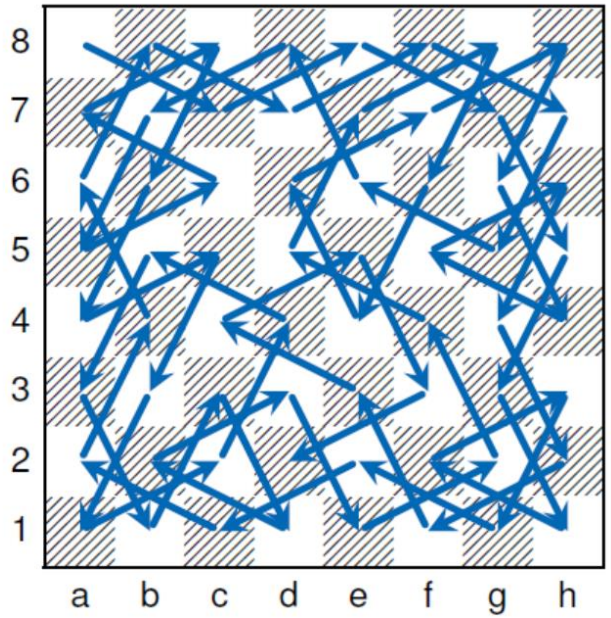
$$T(n) = 1 + k + k^2 + \cdots + k^n \in O(k^n)$$

Beispiel: Springerproblem

Problemstellung

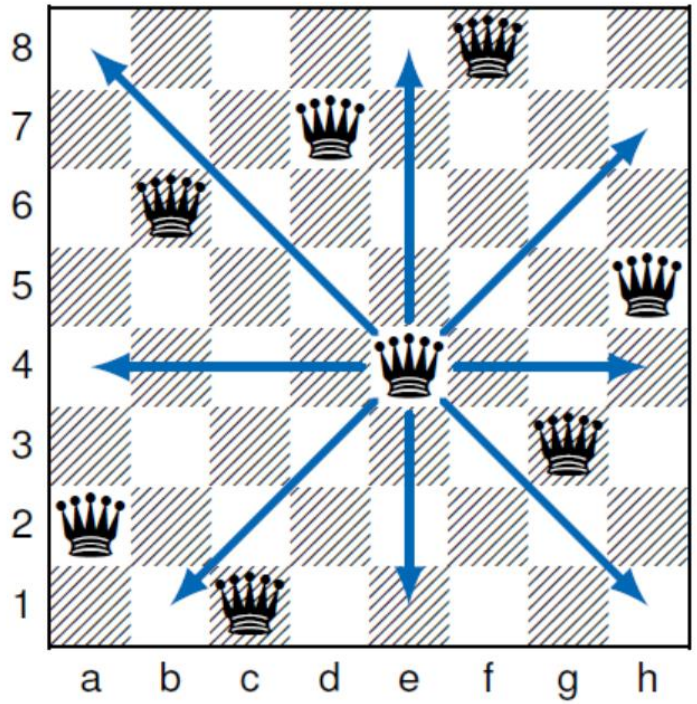
Finden einer Route mit einem Springer, auf der jedes Feld genau einmal besucht wird. Die Tour ist geschlossen, wenn am Ende wieder das Startfeld erreicht wird.





Beispiel: Damenproblem

N Damen Problem: Bedrohung durch eine Dame



Beispiel: Damenproblem

Anzahl der Lösungen

n	Eindeutige Lösungen	Lösungen insgesamt
1	1	1
2,3	0	0
4	1	2
5	2	10
6	1	4
7	6	40
8	12	92
9	46	352
10	92	724
11	341	2680
12	1787	14200
13	9233	73712

Obere Schranke für die Lösungsanzahl D(n) des Damenproblems ist n!

Damenproblem: Lösungsstrategien

Vollständige Enumeration (naiv)

- Alle denkbaren Platzierungen werden ausgetestet
- Bei 8x8 Brettern gibt es $64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57 \approx 1{,}78 \cdot 10^{14}$ denkbare Platzierungen.
- Das benötigt bei 1000 Tests / s ca. 1.1 Millionen Tage Rechenzeit

Damenproblem: Lösungsstrategien

Vollständige Enumeration (verbessert)

- In einer Zeile (und in einer Spalte) kann jeweils nur eine Dame stehen
- Vollständige Enumeration unter Beachtung dieser Nebenbedingung
- Dann sind $8 \cdot 7 \cdot 6 \cdot\cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = n! = 40.320$ Stellungen denkbar

Damenproblem: Lösungsstrategien

Vollständige Enumeration (verbessert)

1. Für jede Zeile eine Dame vorsehen
2. In jeder Zeile die Position (=Spaltenposition) der Dame speichern. Damit mit einem 1D-Array realisierbar
3. Damen zeilenweise an richtige Position setzen

    Nacheinander an jede erlaubte Spaltenposition innerhalb einer Zeile eine Dame setzen

    Im Fall einer Sackgasse vorherige Entscheidung revidieren (= Backtracking)

Damenproblem

Algorithmus

```
1.  function NDAMEN(Zeile)
2.      Spalte = 0
3.      while Spalte < N do
4.          Spalte = Spalte + 1
5.          if Dame in Zeile, Spalte wird nicht bedroht then
6.              Platziere Dame
7.              if Zeile = N then
8.                  Gib Lösung aus
9.              else
10.                 NDAMEN(Zeile+1)
11.             end if
12.             Mache Platzierung rückgängig
13.         end if
14.     end while
15. end function
```

Bemerkungen zum Backtracking

- Programmiersprache PROLOG enthält Backtracking als festen Bestandteil
- Rekursive und iterative Implementierungen realisierbar
- Ungünstige Zeitkomplexität
- Laufzeit kann durch Heuristiken verringert werden

Weitere Beispiele zu Backtracking

- Rucksackproblem (Objekte mit Gewicht und Nutzwert optimal packen)
- Färbeproblem (Einfärbung von benachbarten Knoten in einem Graphen)
- Brettspiele (z.B. Solitär)
- Sudoku (per Trial and Error)
- Wegesuche (von A nach B in einem Graphen)
- ...