



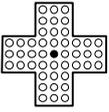
Informatik II Backtracking

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

- Szenario: man hat eine Reihe von **Entscheidungen** aus einer Menge von **Wahlmöglichkeiten** zu treffen, wobei
 - man nicht genug Informationen hat, um zu wählen;
 - jede Entscheidung zu einer neuen Menge von Wahlmöglichkeiten führt;
 - eine Folge von Wahlmöglichkeiten (oder auch mehrere) die Lösung des Problems sein können;
 - manche Entscheidungen evtl in eine "Sackgasse" führen.
- **Backtracking** ist der methodische Weg, verschiedene Folgen von Entscheidungen zu **probieren**, bis eine gefunden wird, die das Problem löst

G. Zachmann Informatik 2 - SS 06 Backtracking 2

Beispiele

- **Ausweg aus einem Labyrinth:**
 - finde den Weg vom Eingang zum Ausgang
 - an jeder Kreuzung gibt es 2-3 Entscheidungsmöglichkeiten: gerade aus, rechts, links
 - Algo: s. "Faden der Ariadne" = früheste Beschreibung des Backtracking-Verfahrens
- **Färben einer Landkarte:**
 - eine Karte soll mit nur 4 Farben gefärbt werden
 - Einschränkung: benachbarte Länder müssen verschieden gefärbt werden
- **Solitaire (Brettspiel):**
 - Alle Löcher, bis auf eins, sind mit Stiften besetzt
 - Man kann mit einem Stift über einen anderen springen
 - Übersprungene Stifte werden entfernt
 - Ziel: alle Stifte, bis auf den letzten zu entfernen

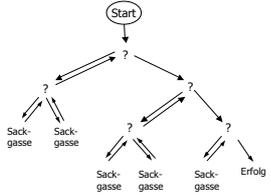
G. Zachmann Informatik 2 - SS 06 Backtracking 3

Entscheidungsbaum

Abstrakter Algorithmus:

```

explore( n = node ):
if n ist Blatt:
    if n Zielknoten:
        return "Erfolg"
    else:
        return "Sackgasse"
foreach Kind c von n:
    explore( c )
if c erfolgreich:
    return "Erfolg"
return "Sackgasse"
    
```

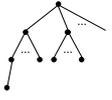


- Backtracking kann als Suche nach einem speziellen "Zielblatt" eines Baumes gesehen werden
- Achtung: dieser Baum wird beim Backtracking nie real aufgebaut!
- Wesentliche Komponente: Kriterium, so daß Sackgassen möglichst weit oben erkannt werden
- Realisierung der Pfadrückverfolgung durch Rekursion

G. Zachmann Informatik 2 - SS 06 Backtracking 4

Zusammenhang zu Depth-First-Search

- Ein Knoten wird als **nützlich** bezeichnet, falls er zu einer Lösung führen **könnte**, sonst **nicht nützlich**
- Hauptidee: Backtracking besteht aus der Anwendung von DFS auf den Entscheidungsbaum, wobei für jeden Knoten entschieden wird
 - ob er nützlich ist → weiter absteigen;
 - ob er nicht nützlich ist → Rückschritt zu seinem Elternknoten
- Weitere Betrachtungsweise:
 - Der Suchraum wird durch den Baum ausgefüllt
 - Jedes Blatt ist eine Lösung oder keine Lösung
 - der Suchraum wird mit Hilfe von Rekursion und Backtracking durchsucht, um eine Lösung zu finden



G. Zachmann Informatik 2 - SS 06 Backtracking 5

Färbungsproblem

- Vier-Farben-Satz:** jede Landkarte in der Ebene lässt sich mit höchstens 4 Farben so färben, daß keine 2 Länder mit gemeinsamer Grenze die gleich Farbe haben
- Datenstruktur muß folgendes speichern:
 - eine Farbe für jedes Land,
 - zu jedem Land alle seine Nachbarn

G. Zachmann Informatik 2 - SS 06 Backtracking 6

Historie

- Ursprung:
 - 1852 aufgestellte Vermutung durch Francis Guthrie beim Färben der Karte der Ländereien von England: 4 Farben genügen um die Länder einer Karte so zu färben, daß alle benachbarten Länder unterschiedliche Farben tragen
 - Veröffentlichung des Problems durch Brief des Mathematik-Professors De Morgan an Hamilton
- Beweisfindung:
 - Beweise durch Kempe (1878) und Tait (1880) wurden 1890 bzw. 1891 widerlegt
 - 1890: Formulierung des 5-Farben-Satzes durch Heawood → Existenz einer oberen Schranke bewiesen
 - 1977: Beweis durch Ken Appel und Wolfgang Haken
 - Beweis reduziert die problematischen Fälle von unendlich auf 1.936, später weniger, welche vom Computer geprüft wurden
 - 2004: Entwicklung eines formalen Beweises mittels Beweis-Assistenten COQ durch Werner und Gonthier
 - (S.a.: bewiesenermaßen korrekter Compiler in Coq)
 - Erstes Problem das mittels Computer bewiesen wurde

G. Zachmann Informatik 2 - SS 06 Backtracking 7

Erschöpfende Suche

- Länder werden durchnummeriert, dito die Farben
- Erzeugung aller möglichen Färbungen durch „Tacho-Mehode“:
 - jedes Rädchen = ein Land
 - starte mit Zählerstand 0...0
 - drehe rechtes Zählrädchen einmal durch alle Ziffern (= Farben)
 - bei Übergang von 3 → 0 im rechten Rädchen: drehe zweitrechtes Rädchen 1 Ziffer weiter
 - bei jedem Zählerstand: überprüfe Konsistenz (4-Farben-Bedingung)



G. Zachmann Informatik 2 - SS 06 Backtracking 8

G. Zachmann Informatik 2 - SS 06 Backtracking 9

▪ Nachteile:

- im ungünstigsten Fall müssen fast alle Kombinationen durchprobiert werden
- Algorithmus zählt weiter, obwohl klar ist, daß einige Kombinationen übersprungen werden können
- sehr hoher Zeitaufwand

G. Zachmann Informatik 2 - SS 06 Backtracking 10

Backtracking-Lösung

- Ein Land färben und dann versuchen, das nächste zu färben, so daß folgende Bedingungen erfüllt sind:
 - die benachbarten, bereits gefärbten Länder haben unterschiedliche Farben
 - keines der benachbarten, noch ungefärbten Länder wird un färbar
- Bedingungen können erfüllt sein, aber dennoch zu keiner Lösung führen (Sackgasse)

G. Zachmann Informatik 2 - SS 06 Backtracking 11

7 ist un färbar geworden!
→ Backtracking

7 wieder un färbar

G. Zachmann Informatik 2 - SS 06 Backtracking 12

- Vorteil: wesentlich effizienter als erschöpfende Suche
- Nachteil: abhängig von der Nummerierung, d.h. bei ungünstiger Nummerierung ist der Aufwand ebenfalls enorm hoch
- Implementierung:
 - Länder sind Instanzen einer Klasse, deren jede neben den Referenzen der Nachbarländer ein Array der Länge 4 enthält:
 - Wert 1 an der Stelle i = Land i ist in der Farbe i gefärbt
 - Wert -1 = Land i kann mit Farbe i gefärbt werden
 - Wert 0 = Land kann **nicht** mit i gefärbt werden
 - Initialisierung mit -1
 - Realisierung der Pfädrückverfolgung durch Rekursion

G. Zachmann Informatik 2 - SS 06

Backtracking 13

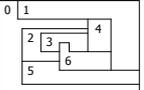
Vereinfachte Implementierung in Python

- Achtung:
 - im folgenden wird die "Unfärbbar"-Heuristik weggelassen
 - es gibt nur 1 Klasse für alle Länder zusammen
 - Bessere Implementierung à la vorige Folie → Übungsaufgabe
- Datenstruktur:
 - Klasse `Map` mit 2 internen Arrays
 - Array `colors`, wobei `colors[i]` die Farbe des i -ten Landes ist
 - Array `neighbors` mit den benachbarten Ländern
 - Beispiel: `neighbors[5][3] == 8` bedeutet: Land 8 ist zu Land 5 benachbart (hier ist es zufällig das dritte Land in der Liste der zu Land 5 benachbarten Länder)

G. Zachmann Informatik 2 - SS 06

Backtracking 14

```
class Map(object):
    def __init__(self):
        self.neighbors = 7 * [None]
        self.neighbors[0] = [ 1,4,2,5 ]
        self.neighbors[1] = [ 0,4,6,5 ]
        self.neighbors[2] = [ 0,4,3,6,5 ]
        self.neighbors[3] = [ 2,4,6 ]
        self.neighbors[4] = [ 0,1,6,3,2 ]
        self.neighbors[5] = [ 2,6,1,0 ]
        self.neighbors[6] = [ 2,3,4,1,5 ]
        self.colors = 7 * [None]
```



The diagram shows a map with 7 countries labeled 0 through 6. Country 0 is at the top, 1 is to its right, 2 is below 0, 3 is below 1, 4 is to the right of 3, 5 is below 2, and 6 is below 3. Lines connect adjacent countries to show their neighbors.

G. Zachmann Informatik 2 - SS 06

Backtracking 15

```
def main():
    map = Map()
    print map.explore(0, "red")
    print map.colors

class Map
...
def explore( self, country, color ):
    if country >= len( self.neighbors ):
        return True
    if okToColor( country, color ):
        self.colors[country] = color
        for c in [ "red", "yellow", "green", "blue" ]:
            if explore( country+1, c ):
                return True
        return False
```

G. Zachmann Informatik 2 - SS 06

Backtracking 16

```

# teste, ob irgend ein Nachbar von 'country'
# die Farbe 'color' hat
def okToColor( self, country, color ):
    for i in range( 0, len(map[country]) ):
        if self.colors[ map[country][i] ] == color :
            return False
    return True

```

G. Zachmann Informatik 2 - SS 06 Backtracking 17

Kurze Wiederholung

- Es wurden alle Länder rekursiv durchlaufen, begonnen wurde mit Land 0
- Für jedes Land war eine Farbe zu wählen:
 - die Farbe mußte von denen aller Nachbarn verschieden sein
 - wenn keine Farbe gefunden werden konnte, wurde ein Fehler gemeldet
 - wenn eine Farbe gefunden werden konnte, wurde sie benutzt und mit dem nächsten Land fortgefahren
 - als alle Länder durchlaufen waren (alle wurden gefärbt), wurde Erfolg gemeldet
- Nach der Rückkehr vom obersten Aufruf war der Algorithmus fertig

G. Zachmann Informatik 2 - SS 06 Backtracking 18

Komplexität

- Exponentieller Aufwand:
 - Anzahl der Kombinationen steigt exponentiell
 - Anzahl der möglichen Sackgassen steigt ebenfalls exponentiell, daher ist Backtracking oftmals deutlich schneller als erschöpfende Suche
- Fällt in Klasse NP-schwer:
 - D.h., es gibt wahrscheinlich keinen effizienten (= polynomiellen) Algorithmus

G. Zachmann Informatik 2 - SS 06 Backtracking 19

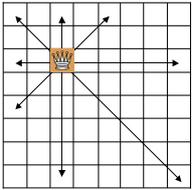
Anwendungen

- Stundenplanproblem
 - Veranstaltungen = Länder
 - Verant., die nicht gleichzeitig ablaufen können, sind "benachbart"
 - Anzahl der Farben = Anzahl der verschiedenen Zeitfenster
- Registerzuweisungs-Probleme
- Bandbreitenzuweisungs-Probleme
- Viele mathematische Probleme lassen sich als Knotenfärbeproblem formulieren

G. Zachmann Informatik 2 - SS 06 Backtracking 20

Das Acht-Damen-Problem

- Problem: stelle 8 Damen so auf ein Schachbrett, daß keine Dame eine andere angreift
- Schachbrett = 8x8 Felder
- eine Dame kann jede andere Figur angreifen, die in derselben Zeile, derselben Spalte, oder der Diagonalen steht
- (die Dame ist die mächtigste Figur beim Schach)



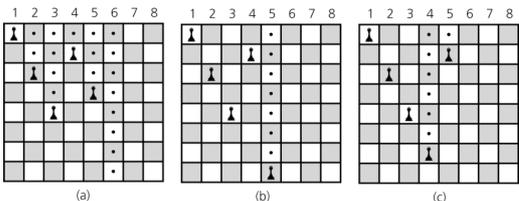
G. Zachmann Informatik 2 - SS 06 Backtracking 21

Naive Strategie

- Naive Strategie: alle möglichen Belegungen ausprobieren
 - Anzahl Möglichkeiten = $64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57$
 - zu viele → nicht praktikabel
- Einfache Beobachtung:
 - keine Dame kann zusammen mit einer anderen in einer Zeile oder Spalte stehen
 - es müssen nur noch $8! = 40\,320$ Damen-Anordnungen auf Diagonalangriffe überprüft werden

G. Zachmann Informatik 2 - SS 06 Backtracking 22

Backtracking-Beispiel



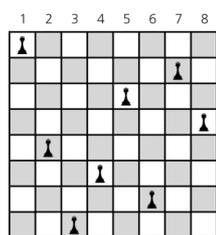
(a) 5 Damen können sich nicht gegenseitig angreifen, aber jede Dame in Spalte 6 könnte es

(b) Backtracking zu Spalte 5 um eine anderes Feld für die Dame zu probieren

(c) Backtracking zu Spalte 4 um eine anderes Feld für die Dame zu probieren und dann Spalte 5 erneut zu betrachten

G. Zachmann Informatik 2 - SS 06 Backtracking 23

Eine Lösung des Problems



G. Zachmann Informatik 2 - SS 06 Backtracking 24

Implementierung

- Verallgemeinerung:
 - Konstruiere alle Lösungen
- Methode:
 - Reduziere $n \times n$ -Problem auf $(n-1) \times n$ -Problem (eine Zeile weniger)
 - Per Konstruktion können keine 2 Damen in derselben Zeile stehen
 - Ausgabe: Liste von Lösungen; Lösung = Array von Spaltennummern, Index = Zeilennummer

```
# Erzeuge eine Liste von Lösungen auf einem Brett mit
# reihen und spalten.
# Eine Lösung wird durch eine Liste der Spaltenpositionen,
# indiziert durch die Reihennummer, angegeben.
# Die Indizes beginnen mit Null.

def damenproblem( reihen, spalten ):
    if reihen == 0:
        return [[]]
        # Eine Lösung, nämlich: keine Dame
        # auf dem nicht-existierenden Brett
    else:
        return dame_dazu(reihen-1, spalten,
                        damenproblem(reihen-1, spalten) )
```

```
for loesung in damenproblem( 8, 8 ):
    print loesung
```

Erinnerung: dies ist ein Bsp. für eine verschachtelte Rekursion ("compound recursion") [s. 1. Semester]

```
# Probiere für alle gegebene Teilloesung,
# eine Dame in "neue_reihe" und alle Spalten zu stellen.
# Wenn kein Konflikt mit der Teilloesung auftritt,
# ist eine neue Loesung des um eine Reihe erweiterten
# Bretts gefunden.

def dame_dazu(neue_reihe, spalten, vorherige_loesungen):
    neue_loesungen = []
    for loesung in vorherige_loesungen:
        # Versuche jede Spalte von neue_reihe
        for neue_spalte in range(spalten):
            if kein_konflikt( neue_reihe, neue_spalte,
                             loesung ):
                # Kein Konflikt → dieser Versuch ist Lsg
                neue_loesungen.append( loesung +
                                       [neue_spalte] )
    return neue_loesungen
```

```
# Kann Dame in neue_spalte / neue_reihe gestellt werden,
# ohne dass sie eine der schon stehenden D. schlagen kann?

def kein_konflikt( neue_reihe, neue_spalte, loesung ):
    # Stelle sicher, dass neue Dame mit keiner exist.en
    # Dame auf gleicher Spalte oder Diagonale steht
    for reihe in range(neue_reihe):
        if ( loesung[reihe] == neue_spalte or
            loesung[reihe] + reihe == neue_spalte + neue_reihe or
            loesung[reihe] - reihe == neue_spalte - neue_reihe ):
                # gleiche Spalte
                # gleiche Diagonale
                # gleiche Diagonale
    return 0
    return 1
```

Logo

G. Zachmann Informatik 2 - SS 06 Backtracking 29

Demo

Backtracking: Die Acht-Damen Aufgabe (erfordert Java-Script)

Ziel: Acht Damen sollen so auf ein Schachbrett gestellt werden, dass keine Dame von einer anderen Dame "bedroht" wird.

Ein Mausklick auf ein leeres Feld des Schachbretts bringt eine Dame auf dieses Feld. Die "bedrohten" Felder werden angezeigt, das Feld mit der zuletzt gesetzten Dame ist grün markiert.

Jeweils die zuletzt gesetzte Dame kann durch Anklicken wieder vom Schachbrett entfernt werden.

Aktivieren des Schachbretts "automatisch" bewirkt den Ablauf eines einfach gehaltenen Zurückverfolgungsalgorithmus ("backtracking") zum Auffinden einer Lösung. Sein Ablauf kann durch Aktivieren der Option "Mausklick" unterbrochen werden. Der Algorithmus endet, wenn eine Lösung gefunden wurde, oder - bei vorangegangener ungünstiger Positionierung von Damen zur Mausclick - mit dem leeren Schachbrett. In diesen Fällen kann mit dem Knopf "neue Lösung" nach weiteren Lösungen gesucht werden. Mit Einstellen einer Verzögerung (in Millisekunden) kann die Geschwindigkeit der Bildausgabe angepasst werden.

© H.B. Meyer

<http://www.fh-wuppertal.de/mhb/backtrack/achtdamen/eight.htm>

G. Zachmann Informatik 2 - SS 06 Backtracking 30

Unterhaltsame "Lehrinheit"

Math⁴(Prism)⁴

Backtracking
"nach vorn wenn möglich, zurück wenn nötig"

Inhaltsverzeichnis Arbeitsblatt

Los geht's!

Autor: Andrea Prosser - 10. Dezember 2004

<http://www.matheprisma.uni-wuppertal.de/Module/BackTr/index.htm>

G. Zachmann Informatik 2 - SS 06 Backtracking 31