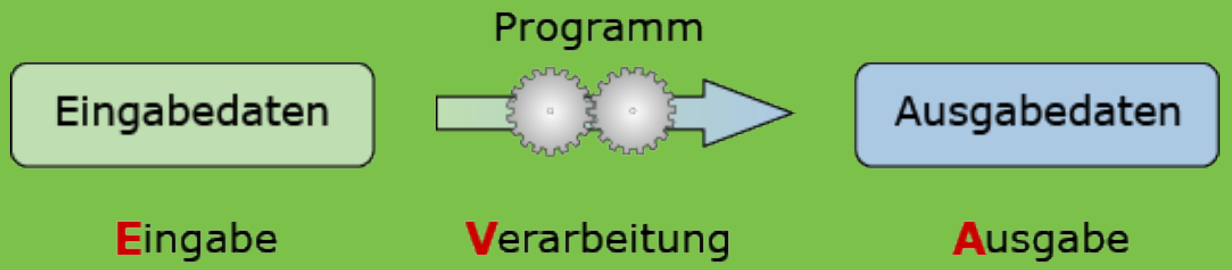
Hochschule Fulda
University of Applied Sciences

Programmierung 1 – Backtracking



Yvonne Jung

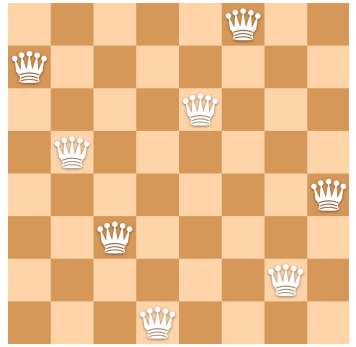
Backtracking

Hochschule Fulda
University of Applied Sciences

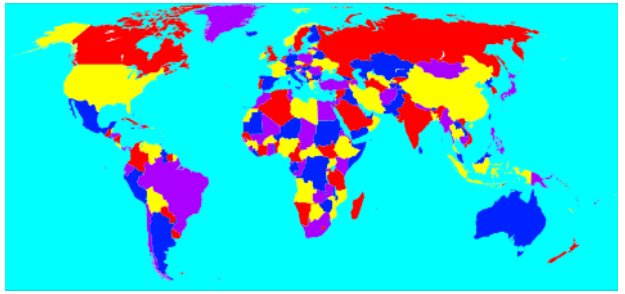
- Prototypischer Anwendungsfall von Rekursion
  - Algorithmenmuster für Such- und Optimierungsprobleme
- Trial-and-Error-Verfahren
  - Versucht Teil- zu Gesamtlösung auszubauen durch systematisches Ausprobieren aller in Frage kommenden Lösungsmöglichkeiten
  - Einzelne Schritte auf Ziel hin werden (durch neuen Rekursionsaufruf) ausprobiert und notiert
  - Falls diese in Sackgasse führen, wird letzter Schritt rückgängig gemacht („backtrack“) und anderer versucht, bis Ziel evtl. erreicht ist
- Exponentielle Laufzeit im schlechtesten Fall, daher nur für kleine Problemgrößen geeignet

Beispiele

Hochschule Fulda
University of Applied Sciences



- Wegsuche (z.B. durch Labyrinth)
- N-Damen Problem
  - *n* x *n* Schachbrett und *n* Damen
  - Damen so platzieren, dass keine Dame eine andere bedroht
    - Durchlaufe Brett zeilenweise und versuche, Dame auf nicht bedrohtem Feld zu platzieren
    - Falls Dame nicht platziert werden kann, nimm zuletzt platzierte Dame weg u. bewege sie weiter (korrigiere Bedrohungsinformation)



- Teilsummenproblem
- Färbeproblem
  - Landkarte mit *k* Ländern
  - Soll mit *n* Farben eingefärbt werden, so dass Nachbarn je verschieden gefärbt

Übungsaufgabe

Hochschule Fulda
University of Applied Sciences

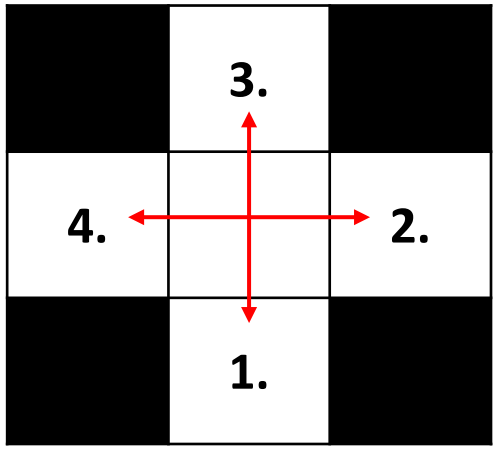
- Implementieren Sie in C ein Verfahren, das mit Hilfe von Backtracking zu einem gegebenen Labyrinth den Weg vom Startpunkt zum Ziel findet
- Der Algorithmus arbeitet auf einem zweidimensionalen Feld, wobei die Anzahl der verwendeten Zeilen *m* und Spalten *n* benutzerdefiniert ist
  - Sie dürfen davon ausgehen, dass max. 20 Zeilen und 20 Spalten benötigt werden ($m, n \leq 20$), d.h. Sie können initial ein Feld fester Größe anlegen, wovon ggfs. nur ein Teil verwendet wird
  - Start ist immer bei (0, 0), das Ziel ist bei (*m*-1, *n*-1)
- Das Labyrinth wird zunächst in einer Datei (z.B. *labyrinth.txt*) angegeben, wobei in der ersten Zeile zwei Zahlen für die Anzahl der Zeilen u. Spalten stehen sollten
  - Danach folgen in ausreichender Zahl (also *m*n*-mal) die Ziffern 0 und 1, durch beliebig viele Leerzeichen und Zeilenumbrüche voneinander getrennt (also formatfreie Eingabe)
    - 0 an Stelle (i, j) bedeutet, dass es sich um einen Teil des Weges handelt und passiert werden kann,
    - 1 an Stelle (i, j) heißt, dass es sich um ein Hindernis handelt und nicht überwunden werden kann
- Das Ergebnis soll schließlich auf der Kommandozeile ausgegeben werden

Wegsuche in Labyrinth

Hochschule Fulda
University of Applied Sciences

- Labyrinth als zweidimensionales Feld gegeben
- Problem bei Wegsuche
  - Wie kann man Sackgassen oder Schleifen in Labyrinth erkennen und trotzdem ans Ziel kommen?
    - Bei Sackgasse Weg wieder zurückgehen (Backtracking)
    - Markierung („Brotkrumen“) wie Sackgasse behandeln, um nicht im Kreis zu laufen
- Lösungsidee (Detailansicht)
  - Betrachte aktuelles Feldelement und gehe von da aus in alle 4 Himmelsrichtungen
  - Markiere bereits gelaufenen Weg

Start	0	1	1	0	1	
Weg	0	0	0	0	1	
	0	1	1	0	1	
	0	1	0	0	1	Hindernis
	0	1	0	0	0	Ziel



Datenstrukturen & Variablen

Hochschule Fulda
University of Applied Sciences

```
#define MAXROWS  20
#define MAXCOLS  20
#define WEG      '0'
#define MAUER    '1'
#define MARKIERT 'x'

typedef char Labyrinth[MAXROWS][MAXCOLS];

// Funktionsprototypen (insbes. für search())

int main() {
  Labyrinth laby;           // Zweidimensionales Feld
  int currRows, currCols;   // Tatsächliche Feldgröße einlesen
  bool geschafft = false;   // Ziel gefunden wenn geschafft
  …
```

ADT

Start	x	1	1	0	1
	x	x	x	x	1
	0	1	1	x	1
	0	1	0	x	1
	0	1	0	x	x

Ziel

- Aufruf: `search(laby, &geschafft, 0, 0, currRows-1, currCols-1);`

Wegsuche-Funktion

Hochschule Fulda
University of Applied Sciences

```
void search(Labyrinth l, bool *done, int y, int x, int zy, int zx) {
  l[y][x] = MARKIERT;
  if (x == zx && y == zy) {  // Am Ziel?
    *done = true;
  }
  else {  // Weg suchen
    if (!*done && y + 1 <= zy && l[y + 1][x] == WEG)  // runter
      search(l, done, y + 1, x, zy, zx);
    if (!*done && x + 1 <= zx && l[y][x + 1] == WEG)  // rechts
      search(l, done, y, x + 1, zy, zx);
    if (!*done && y - 1 >=  0 && l[y - 1][x] == WEG)  // hoch
      search(l, done, y - 1, x, zy, zx);
    if (!*done && x - 1 >=  0 && l[y][x - 1] == WEG)  // links
      search(l, done, y, x - 1, zy, zx);
    if (!*done)
      l[y][x] = WEG;
  }
}
```

Vielen Dank!

Noch Fragen?