Theoretische Informatik

Algorithmik

Autor: Prof. Dipl.-Math. E. Engelhardt

Stand: 02. Januar 2023

Klassische Algorithmen – Quadratwurzel (1/6)

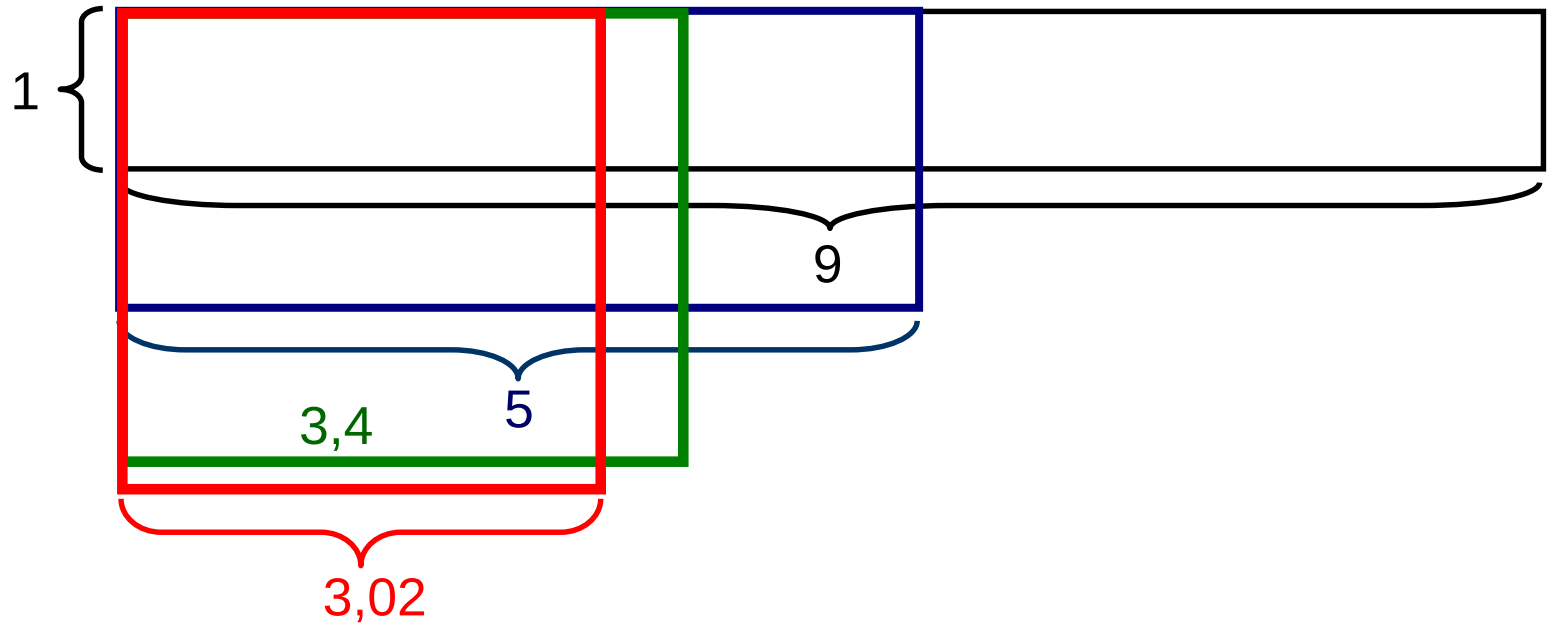
- Entstehung aus Problemen der Flächenmessung durch die Babylonier (etwa 1700 vor Christi)
- Aufgegriffen durch Heron von Alexandria (100 nach Christi)
  - Die Quadratwurzel soll berechnet werden
  - Dazu werden schrittweise immer quadratähnlichere Rechtecke konstruiert, ohne den Flächeninhalt zu verändern
  - Eine Seite wird als arithmetisches Mittel der beiden Rechteck-seiten gewählt und die andere Seite wird berechnet, so dass der Flächeninhalt sich nicht verändert

$x_1 = (x_0 + y_0) / 2$ und $y_1 = a/x_1$

$x_{n+1} = (x_n + y_n) / 2$ und $y_{n+1} = a/x_{n+1}$

$x_{n+1} = (x_n + a/x_n) / 2$ (Pseudocode auf einer folgenden Folie)

Klassische Algorithmen – Quadratwurzel (2/6)



1. Schritt: a= 9 x_0= 9 y_0= 1
2. Schritt: x_1= (9 + 1) / 2= 5 y_1= 9/5
3. Schritt: x_2= (5 + 9/5) / 2= 34/10= 3,4 y_2= 9 / (34/10)= 2,647
4. Schritt: x_3= (3,4 + 2,647) / 2= 3,02 y_3= 9 / (3,02)= 2,98

Klassische Algorithmen – ggT (3/6)

- Problem des größten gemeinsamen Teilers (ggT)
  - wichtige Rolle in der Mathematik z. Bsp. Bruchrechnung

- Euklid von Alexandria (ca. 365 bis 300 v. Chr.)
  - „Nimmt man abwechselnd immer das Kleinere vom Größeren weg,
  - dann muss der Rest schließlich die vorhergehende Größe messen …“
  - („Die Elemente, Zehntes Buch §3“)
  - a= q * b + r und 0 <= r < b
  - Bsp.: 18 und 4 | 18 - 4= 14 ->
    14 und 4 | 14 - 4= 10 ->
    10 und 4 | 10 - 4= 6 ->
    6 und 4 | 6 - 4= 2 ->
    4 und 2 | 4 - 2= 2
  - **-> 2 ist ggT**

Klassische Algorithmen – Primzahlen (4/6)

- Siebverfahren zur Ermittlung von Primzahlen
  - wichtige Rolle in der Zahlentheorie

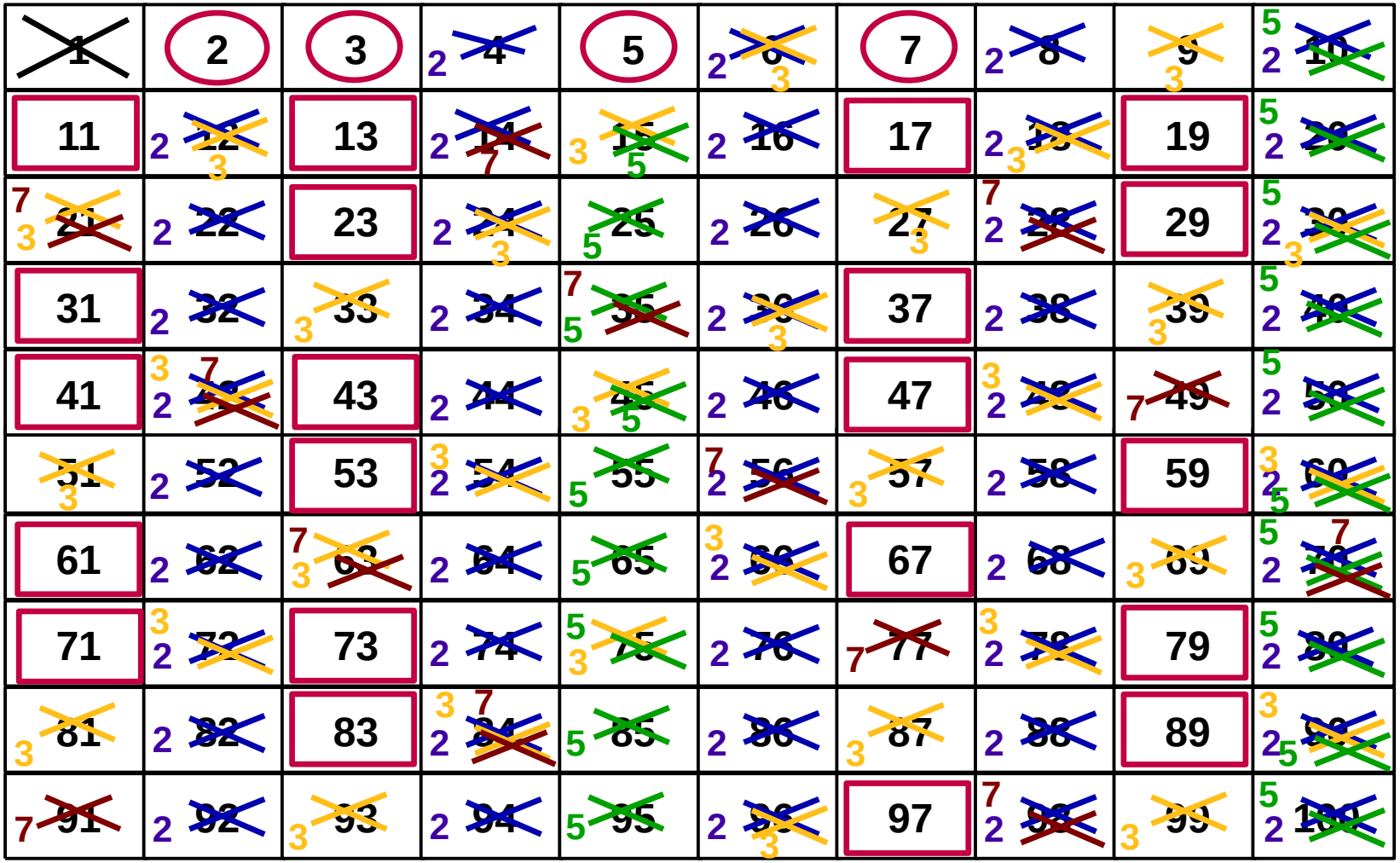
- Eratosthenes von Kyrene (ca. 276 – 194 v.Chr.)
  1. Es werden alle Zahlen des zu untersuchenden Bereichs aufgeschrieben (nur Bitfelder notwendig)
  2. Es wird zur nächsten noch nicht gestrichenen Zahl vorgerückt.
  3. Von dieser Zahl ausgehend werden wieder alle Vielfachen gestrichen
  4. Gehe nach Schritt 2., wenn Zahl kleiner als Wurzel … siehe Bem.

  Alle nicht gestrichenen Zahlen des Feldes sind die Primzahlen

  Bemerkung: Es müssen nur Vielfache einer Zahl gestrichen werden, höchstens bis zur Wurzel der größten zu ermittelnden Primzahl

  Die Komplexität des Siebverfahrens ist O(n log log n)

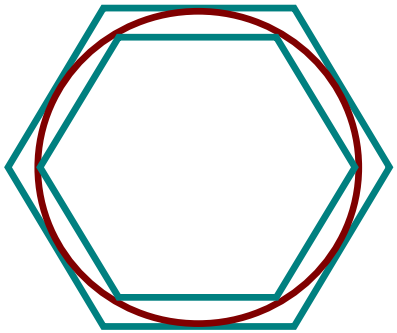
Klassische Algorithmen – Primzahlen (5/6)



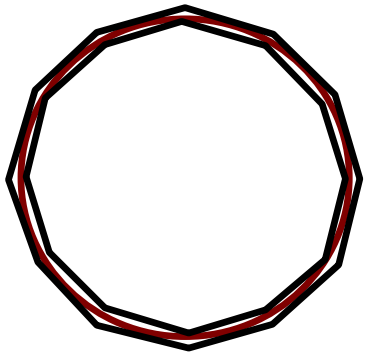
Klassische Algorithmen – Kreiszahl π (6/6)

- Approximation von π
  - wichtige Rolle bei Flächen und Körperberechnungen
- Archimedes von Syrakus (ca. 287 – 212 v.Chr.)

1. Die Annäherung des Kreises erfolgt durch einbeschriebene und umbeschriebene regelmäßige Vielecke.
2. Die Umfänge der Vielecke werden berechnet unter Verwendung des Satzes von Pythagoras.
3. Die Umfänge der einbeschriebenen und umbeschriebenen Vielecke nähern sich sehr stark an, so dass das 96-Eck schon einen auf drei Dezimalstellen genauen Wert für π liefert.



Sechseck



Zwölfeck

Problemlösungsstrategien – Einteilung (1/8)

- Für die Lösung von Problemen muss zunächst die Art des Problems heraus gefunden werden
  - Nichtalgorithmische Probleme (Blei in Gold verwandeln)
  - Theoretisch nicht lösbare algorithmische Probleme (Halteproblem, Entscheidungsproblem)
  - Praktisch nicht lösbare bzw. sehr schwer lösbare Probleme (Problem des Handelsreisenden, Stundenplan-Problem)
  - Praktisch lösbare Probleme (Suchen, Sortieren)

- Die Einteilung kann unter verschiedenen Gesichtspunkten vorgenommen werden
  - Hier sollen zunächst einige heuristische Strategien genannt werden

Problemlösungsstrategien – Heuristik (2/8)

- Der Begriff Heuristik kommt aus dem Griechischen
  - Verb ‚heuriskein' – finden, entdecken
  - Der berühmteste Gelehrte des Altertums, Archimedes von Syrakus, soll nach der Entdeckung des Auftriebs während eines Bades „heureka" rufend auf die Straße gelaufen sein
  - Heuristik ist die Lehre des Problemlösens

Abb.: Antje Elsner

Problemlösungsstrategien – Elementare (3/8)

- Elementare (direkte, naive, straightforward) Methoden
  - Oft sehr allgemein. Sie sind für viele Fälle anwendbar.
  - Sehr effizient, wenn exakte Formeln bekannt sind.
  - Raffiniertere Herangehensweisen sind oft nur bei bestimmten Fällen anwendbar

- Methode der rohen Gewalt „brute force method"
  - Durchprobieren aller Möglichkeiten
  - Diese Methode ist meist sehr aufwändig (hohe Komplexität)
  - Beispiel: Durchprobieren aller Züge bis zu einer bestimmten Tiefe beim Schachspiel

- Gierige Strategie „greedy strategy"
  - Es wird immer das nächste, für das Problem beste, Teilproblem erledigt
  - Beispiel TSP: Es wird immer zur nächstgelegenen Stadt gereist

Strategien – Computerorientierte (4/8)

- Modularität
  - Das Problem wird in Teilprobleme zerlegt, die einfacher zu lösen sind (Baukastenprinzip)
  - Lösung der einzelnen Teilprobleme
  - Einzellösungen werden zur Gesamtlösung zusammen gesetzt
- Rekursion
  - Fundamentales heuristische Prinzip
  - Viele Probleme sind „von Hause aus“ rekursiv
  - Selbstähnlichkeit (Ornamente, Fraktale)
  - Eng verbunden mit dem Prinzip der vollständigen Induktion
  - Kleinere Kopien desselben Problems bis zur kleinsten Stufe
- Versuch und Irrtum (trial and error)
  - Nacheinander Ausprobieren verschiedener Möglichkeiten

Problemlösungsstrategien – Rekursion (5/8)

- Wieviele Kaninchenpaare gibt es nach einem Jahr, wenn es anfangs ein junges Paar gibt, jedes Paar jeden Monat ein neues Paar zur Welt bringt und alle Jungen sich jeweils nach einem Monat fortpflanzen?

- Zahlenreihe:
  1 1 2 3 5 8 13 21 34 55 89 144 …

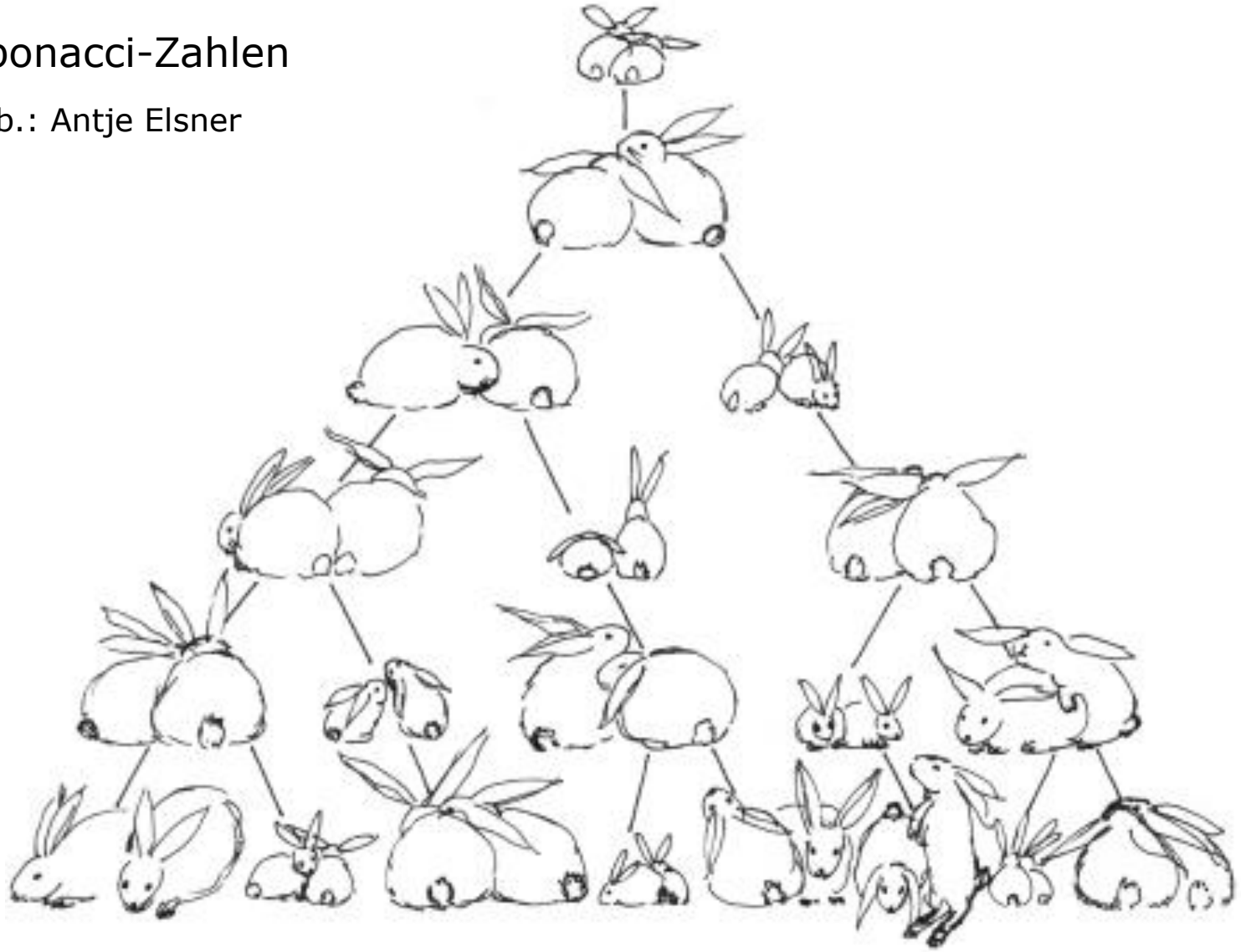
- Mathematische Beschreibung:
  $F_0 = 0,\ F_1 = 1,\ F_{n+2} = F_{n+1} + F_n;\ n >= 0$

- Am Beispiel der Fibonacci-Zahlen lässt sich leicht zeigen, dass bei einfacher Übernahme der rekursiven Formel für die Programmierung bereits bei wenigen hundert Zahlen der Computer für Stunden oder Tage ausgelastet ist

Problemlösungsstrategien – Rekursion (6/8)

Fibonacci-Zahlen

Abb.: Antje Elsner



Problemlösungsstrategien (7/8)

- Prinzip „Teile und Herrsche“ („divide and conquer“)
  - Das Problem wird in etwa gleichgroße Teilprobleme zerlegt, die gelöst werden
  - Die Gesamtlösung wird aus den Teillösungen zusammen gesetzt
  - Algorithmen werden meist rekursiv formuliert
  - Beispiel: Quicksort

- Systematisches Durchlaufen von Baumstrukturen
  - Breitensuche („breadth first“)
  - Tiefensuche („depth first“)
  - Backtracking als Modifikation der Tiefensuche, das heißt, es wird nur so weit in die Tiefe gegangen, wie zum Erkennen einer Lösung nötig ist

Problemlösungsstrategien (8/8)

- Gezielte mathematische Analyse
  - liefert oft sehr effiziente Lösungen
  - Beispiel: Lösung von quadratischen Gleichungen
  - NIM-Spiel
- Geometrische Algorithmen
- Graphen-Algorithmen
- Probabilistische Verfahren
  - Verzicht auf hundertprozentige Sicherheit (Pseudo-Primzahlen)
  - Beispiel: Public Key Cryptography (RSA-Verfahren)
- Simulationsverfahren
- Evolutionäre Algorithmen (EA), Genetische Algorithmen (GA)
  - DNA-Algorithmen
  - Zellulare Automaten
  - Neuronale Netze
- Parallele Algorithmen
- Dynamische Programmierung

Gierige Strategie „greedy strategy“

- **Prinzip:** in jedem Teilschritt so viel wie möglich erreichen

- Bei manchen Problemen kann dies eine optimale Strategie sein
- Beispiel: Herausgabe von Wechselgeld
- Ziel: möglichst wenige Scheine und Münzen heraus geben
  - Preis: 37,34 € => Herausgabe 12,66
  - 10,- €-Schein + 2,- €-Münze + 0,50 €-Münze + 0,10 €-Münze + 0,05 €-Münze + 0,01 €-Münze

- Bei anderen Problemen kann die gierige Strategie auch zu falschen Lösungen führen.
- Beispiel: Schachspiel

Elementare Sortierverfahren
Select-Sort Beschreibung (1/12)

- Aktionen je Schritt
  - Das i-te Element wird zunächst als kleinstes Element des Restfeldes angenommen
  - Vom i-ten Element beginnend wird das Feld nach kleineren Elementen durchsucht und der Index wird festgehalten
  - Pro Schritt wird das gefundene kleinste Element mit dem i-ten Element vertauscht (wenn es ungleich dem i-ten Element ist)
- Ergebnisse je Schritt
  - Je Schritt ist höchstens eine Vertauschung nötig
  - Die kleinste Zahl wurde an die erste Stelle des untersuchten Restfeldes getauscht.
  - Das zu untersuchende Feld wird je Schritt um ein Element kleiner
- Ende des Algorithmus
  - bei n Elementen nach n-1 Schritten

Select-Sort C-Funktion (2/12)

```
int selectSort(int feld[], int anz)
{
   int i, j, hilf, minIndex;

   for (i= 0; i < anz-1; i++)
   {
      minIndex= i;

      for (j= i+1; j < anz; j++)
         if (feld[j] < feld[minIndex])
            minIndex= j;

      if (minIndex != i)
      {
         hilf= feld[i];
         feld[i]= feld[minIndex];
         feld[minIndex]= hilf;
      }
   }
   return 0;
}
```

Insert-Sort Beschreibung (3/12)

- Aktionen je Schritt
  - Wie beim Einsortieren der Karten (beim Kartenspiel) werden die Elemente nacheinander betrachtet und an den richtigen Platz der bereits betrachteten Elemente eingefügt.
  - Die größeren Elemente des bereits vorsortierten (linken) Feldes werden um eine Stelle nach rechts verschoben.
  - Das aktuell betrachtete Element wird an die frei gewordenen Stelle eingefügt.
- Ergebnisse je Schritt
  - Die links stehenden Elemente sind sortiert, aber es kommen weitere Elemente hinzu.
  - Der Test j > 0 in der inneren Schleife ist ineffizient!
  - Das zu untersuchende Restfeld wird je Schritt um ein Element kleiner.
- Ende des Algorithmus
  - bei n Elementen nach n-1 Schritten

Insert-Sort Animation (4/12)

- Sortierung:
  - Buben ganz links in der Reihenfolge: Kreuz, Pik, Herz, Karo
  - Rechts von den Buben die Reihenfolge der Farben:
    Kreuz, Herz, Pik, Karo
  - Die Karten der Farben in der Reihenfolge:
    As, Zehn, König, Dame, Neun, Acht, Sieben

Insert-Sort C-Funktion (5/12)

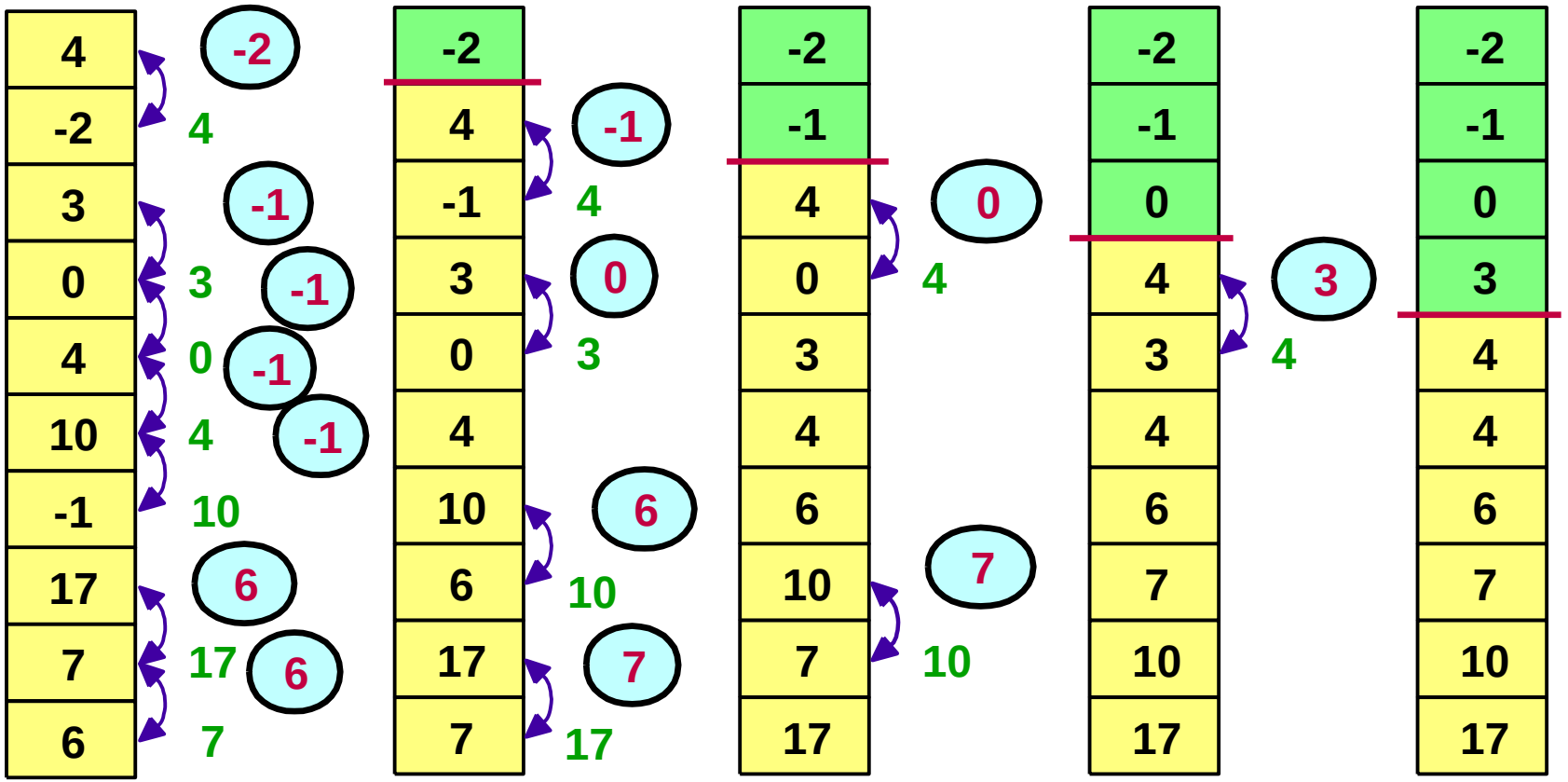
```
int insertSort(int feld[], int anz)
{
   int i, j, hilf;

   for (i= 1; i < anz; i++) // anz-1 Schritte
   {
      hilf= feld[i];

      for (j= i; feld[j-1] > hilf && j > 0; j--)
         feld[j]= feld[j-1];

      feld[j]= hilf;
   }
   return 0;
}
```

Bubble-Sort Beschreibung (6/12)

- Aktionen je Schritt
  - Von unten beginnend werden die zwei benachbarten Zahlen verglichen und gegebenenfalls vertauscht.
  - Dadurch steigen die kleineren Zahlen wie Blasen nach oben und größere Zahlen je Schritt um eine Stelle nach unten.
- Ergebnisse je Schritt
  - Die kleinste Zahl ist garantiert an die erste Stelle des untersuchten Restfeldes gestiegen.
  - Das zu untersuchende Feld wird je Schritt um ein Element kleiner.
- Ende des Algorithmus
  - bei n Elementen nach höchstens n-1 Schritten
  - wenn in einem Schritt kein Tausch mehr durchzuführen ist.

Bubble-Sort Animation (7/12)

1. Schritt 2. Schritt 3. Schritt 4. Schritt fertig



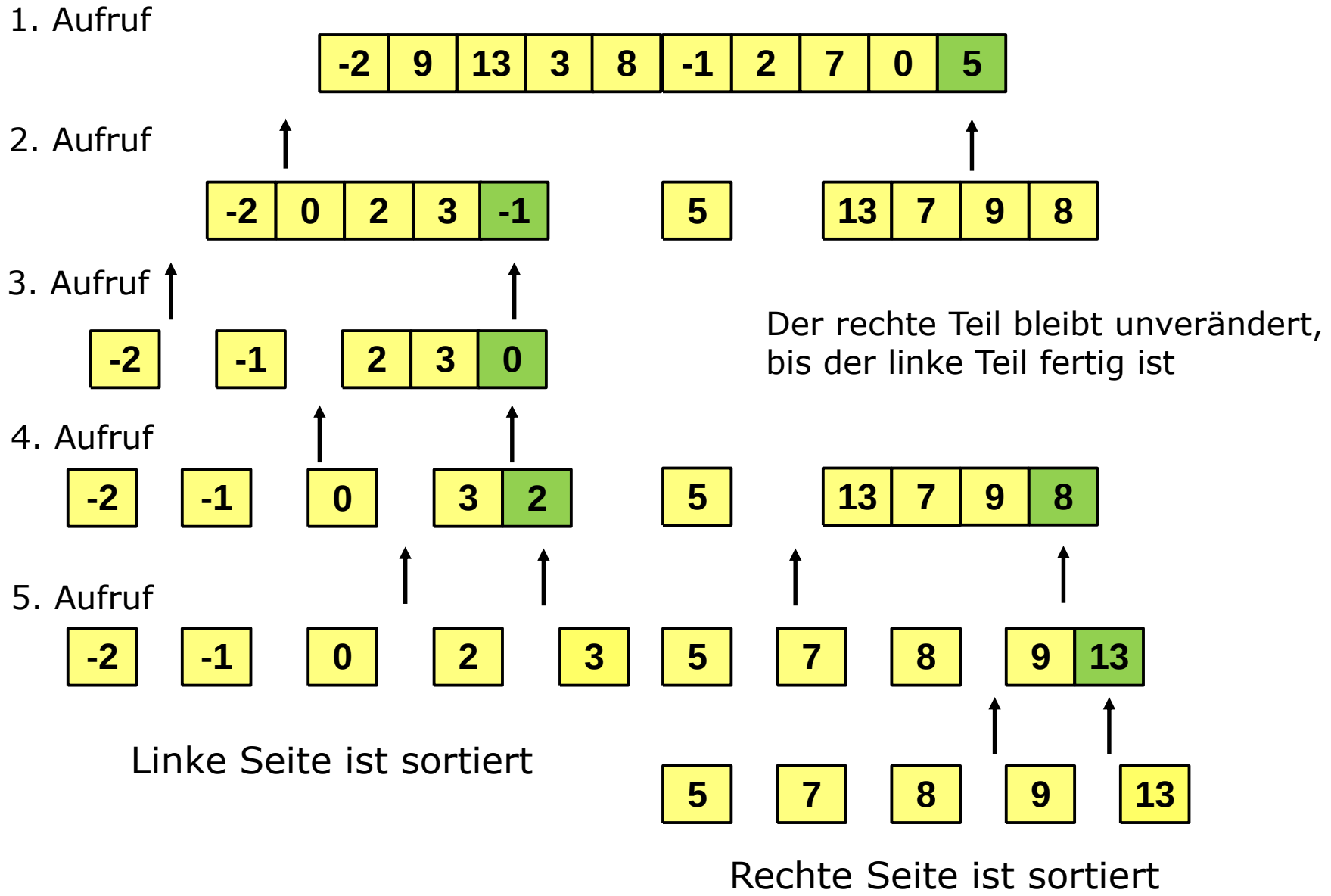
Bubble-Sort C-Funktion (8/12)

```
int bubbleSort(int feld[], int anz)
{
   int i, j, hilf;
   int sortiert= 1;

   for (i= 0; i < anz-1; i++)
   {
      sortiert= 1;

      for (j= anz-1; j > i; j--)
         if (feld[j] < feld[j-1])
         {
            hilf= feld[j-1];
            feld[j-1]= feld[j];
            feld[j]= hilf;
            sortiert= 0;
         }

      if (sortiert) break;
   }
   return 0;
}
```

Quicksort Beschreibung (9/12)

- Entwickelt 1960 von C. A. R. Hoare
- Prinzip „Teile und Herrsche“
- Aktionen je Schritt
  - Die zu sortierende Folge wird in zwei zu sortierende Folgen zerlegt.
  - Zuerst wird ein Element gewählt, das in die endgültige Position gebracht werden soll (möglichst in die Mitte der Folge).
  - Dazu laufen zwei Zeiger von links und rechts aufeinander zu und die Elemente werden mit dem Vergleichs-Element verglichen.
  - Wenn die Zeiger stoppen, werden die Elemente, die offensichtlich an falscher Stelle stehen, vertauscht.
  - Die entstandenen vorsortierten Teilfolgen werden rekursiv erneut betrachtet.
- Ende des Algorithmus
  - Die zu untersuchende Teilfolge besteht nur noch aus einem Element.

Quicksort Animation (10/12)



Quicksort C-Funktion (11/12)

```
int quickSort(int feld[], int li, int re)
{
   int i, j, vergl, hilf;

   if (re > li)
   {
      vergl= feld[re]; i= li-1; j= re;
      while(1)
      {
         while(feld[--j] > vergl) ;      // Suche von rechts
         while(feld[++i] < vergl) ;      // Suche von links
         if (i >= j) break;              // Zeiger getroffen
         hilf= feld[j]; feld[j]= feld[i]; // Tausch,
         feld[i]= hilf;                  // wenn noetig
      }
      hilf= feld[re]; feld[re]= feld[i]; // Vergleichs-
      feld[i]= hilf;       // Element in Mitte tauschen

      quickSort(feld, li, i-1);          // linker Teil
      quickSort(feld, i+1, re);          // rechter Teil

   }
   return 0;
}
```

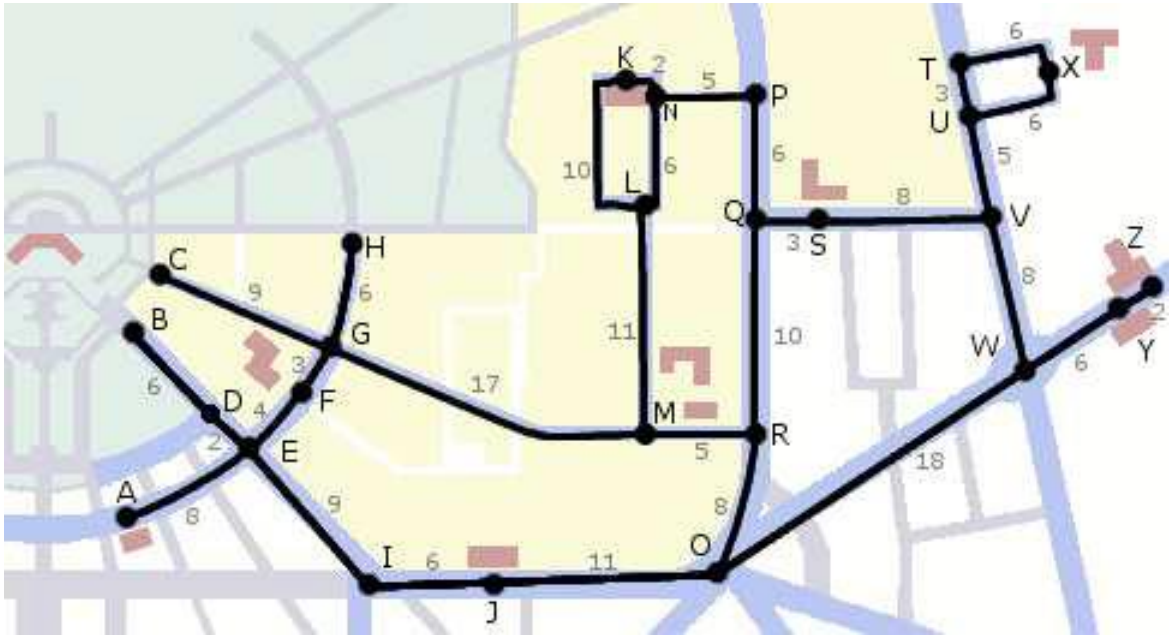
Komplexität der Sortierverfahren (12/12)

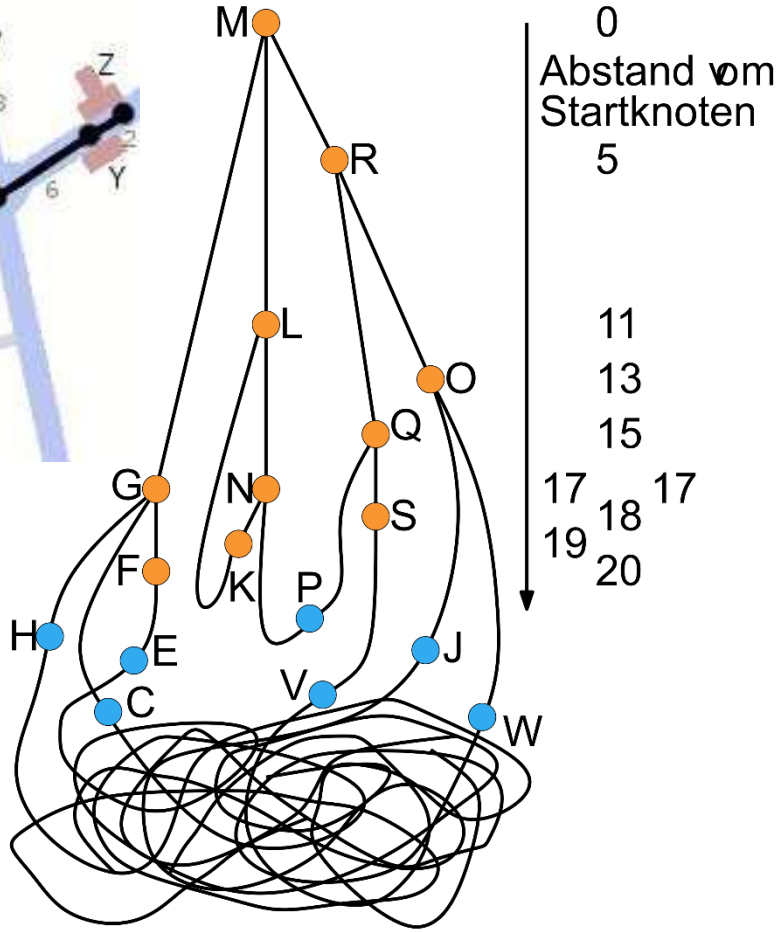
- Select-Sort (Auswahl-Sortieren)
  - Die äußere Schleife wird n-1 mal durchlaufen
  - Die innere Schleife wird im Durchschnitt (n-1)/2 mal durchlaufen.
  - → $O((n-1)*((n-1)/2)) = O((n-1)^2/2) = O(n^2)$
- Insert-Sort (Einfüge-Sortieren)
  - Die äußere Schleife wird n-1 mal durchlaufen
  - Die innere Schleife wird im Durchschnitt ½ * ½ * (n-1) mal durchlaufen. Im Mittel wird erwartet, dass das neue Element in die Mitte des sortierten Bereichs gehört
  - → $O((n-1)*(1/2)*((n-1)/2)) = O((n-1)^2/4) = O(n^2)$
- Bubble-Sort
  - Die äußere Schleife wird n-1 mal durchlaufen
  - Die innere Schleife wird (n-1)/2 mal durchlaufen.
  - → $O((n-1)*((n-1)/2)) = O((n-1)^2/2) = O(n^2)$
- Quicksort
  - Die Rekursionstiefe hängt vom Vergleichselement ab uns ist im besten Fall ld(n) und im mittleren Fall etwa 1,4 * ld(n)
  - → O(n ld (n))

Dijkstra-Algorithmus

- Problem: kürzeste Wege von Startpunkt zu allen Zielpunkten in einem gegebenen Netz
- Lösung: Der Algorithmus von Dijkstra berechnet die kürzesten Wege von einem Startpunkt zu allen erreichbaren Zielpunkten.
- Der Dijkstra-Algorithmus funktioniert nur bei positiv gewichteten Graphen
- – arbeitet analog zur trickreichen Lösung mittels Bindfäden

Dijkstra-Algorithmus





Veranschaulichung durch Bindfäden

Aus: Peter Sanders, Johannes Singler
Institut für Theoretische Informatik,
Universität Karlsruhe

Dijkstra-Algorithmus

Dijkstra (*Startknoten)*

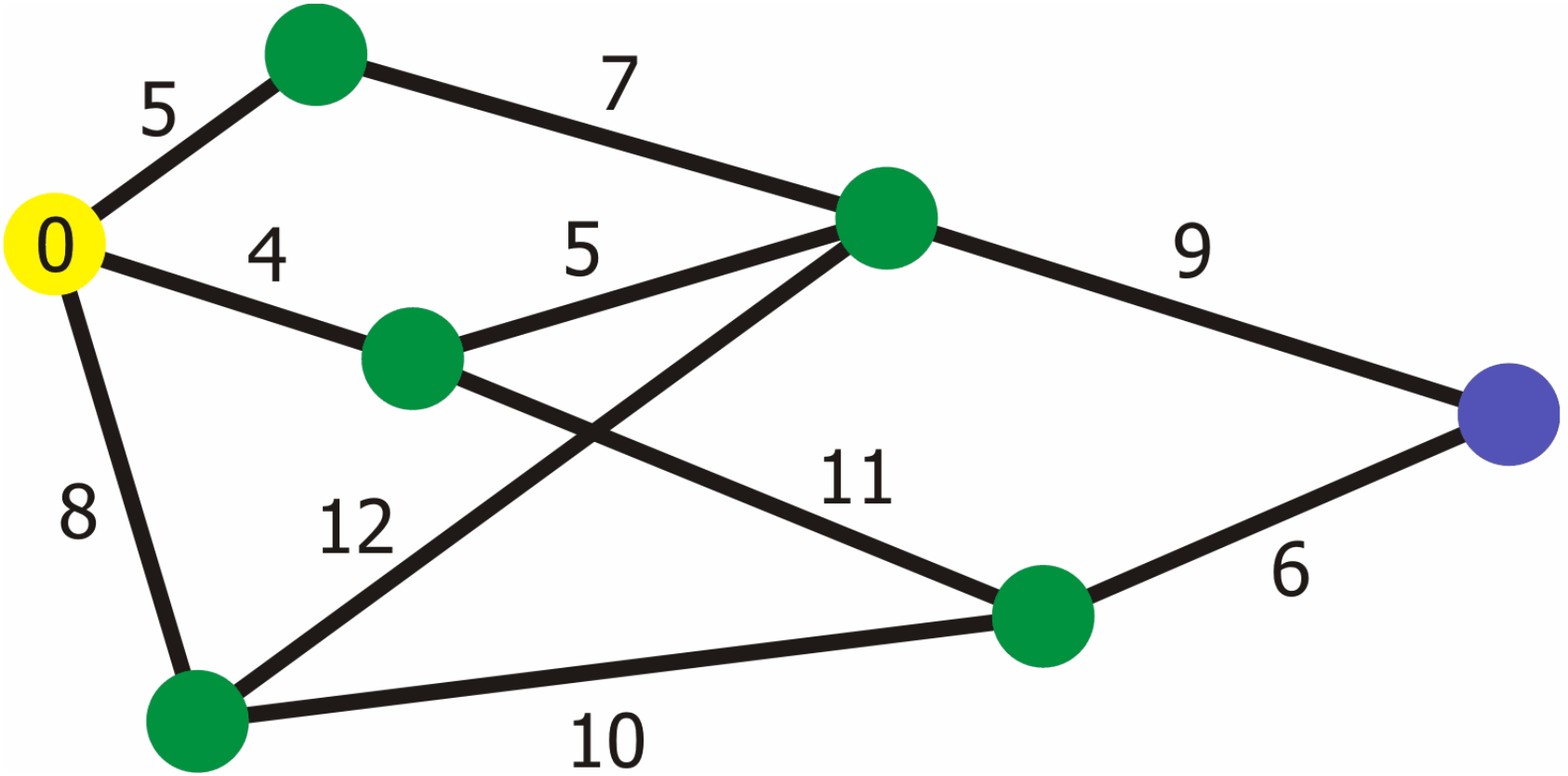
{

- alle Knoten liegen => setze ganz d auf ∞, d[*Startknoten*] = 0
- **while** (es gibt liegende Knoten) **do**
- v = liegender Knoten mit kleinstem d[v]
- mache v hängend
- **forall** (Fäden von v zu Nachbar u der Länge l) **do**
- **if** (d[v] + l) < d[u] **then** d[u] = d[v] + l

}

d[Knoten] – Gewicht des Knoten

Dijkstra-Algorithmus – Beispiel



Backtracking-Algorithmen – Motivation

- Sehr viele praktische Probleme lassen sich auf diese Weise lösen.
  - Spielprogramme
  - Planungsprobleme
  - Optimierungsprobleme
- Beim Backtracking wird der komplette Lösungsraum systematisch abgearbeitet.
- Der Algorithmus terminiert, wenn der Lösungsraum endlich ist.
- Es entsteht in Abhängigkeit der Möglichkeiten oft exponentieller Aufwand.
- Wenn ein Zweig in eine Sackgasse gerät, wird zur nächsten noch nicht bearbeiteten Abzweigung zurück gekehrt (Backtracking).

Backtracking-Algorithmen – Varianten

1. Alle Lösungen finden
2. Abbruch bei der ersten gefundenen Lösung
3. Es werden bewertete Lösungen berechnet und es wird die beste ausgewählt.
4. „Branch-and-Bound“ verfolgt nur Zweige, die eine Lösung prinzipiell zulassen. Es wird versucht, Wege in Sackgassen zu vermeiden.
5. Vorgabe einer maximalen Rekursionstiefe

Backtracking-Algorithmen – einleitendes Beispiel

- Labyrinth-Probleme lassen sich mittels Backtracking lösen
- In diesem Fall einfacher lösbar durch „Rechte-Hand-Regel“

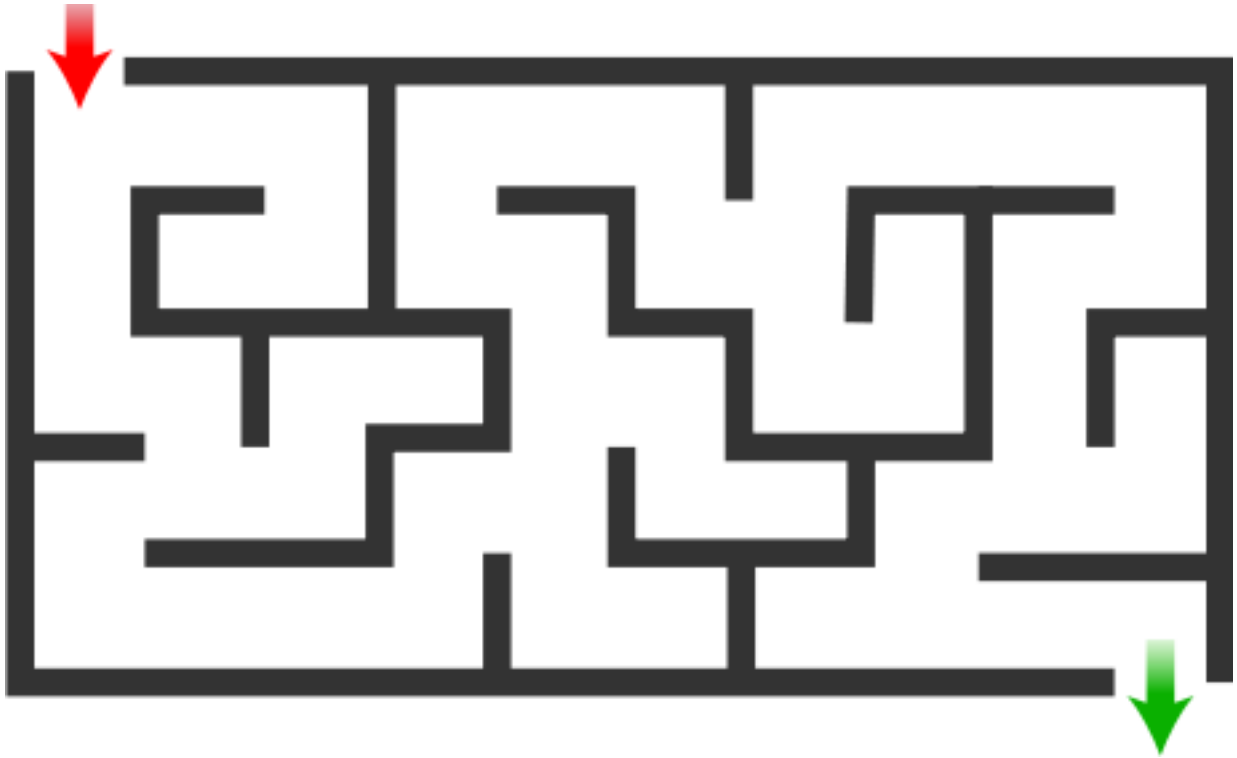


Bild aus wikipedia-Commons

Backtracking-Algorithmen – Labyrinth

- Abgewandeltes Labyrinth-Problem: Ausweg finden
- Bei Anwendung „Rechte-Hand-Regel“ wird Ausgang gefunden
- „Linke-Hand-Regel“ führt zum Eingang, nicht zum Ausgang

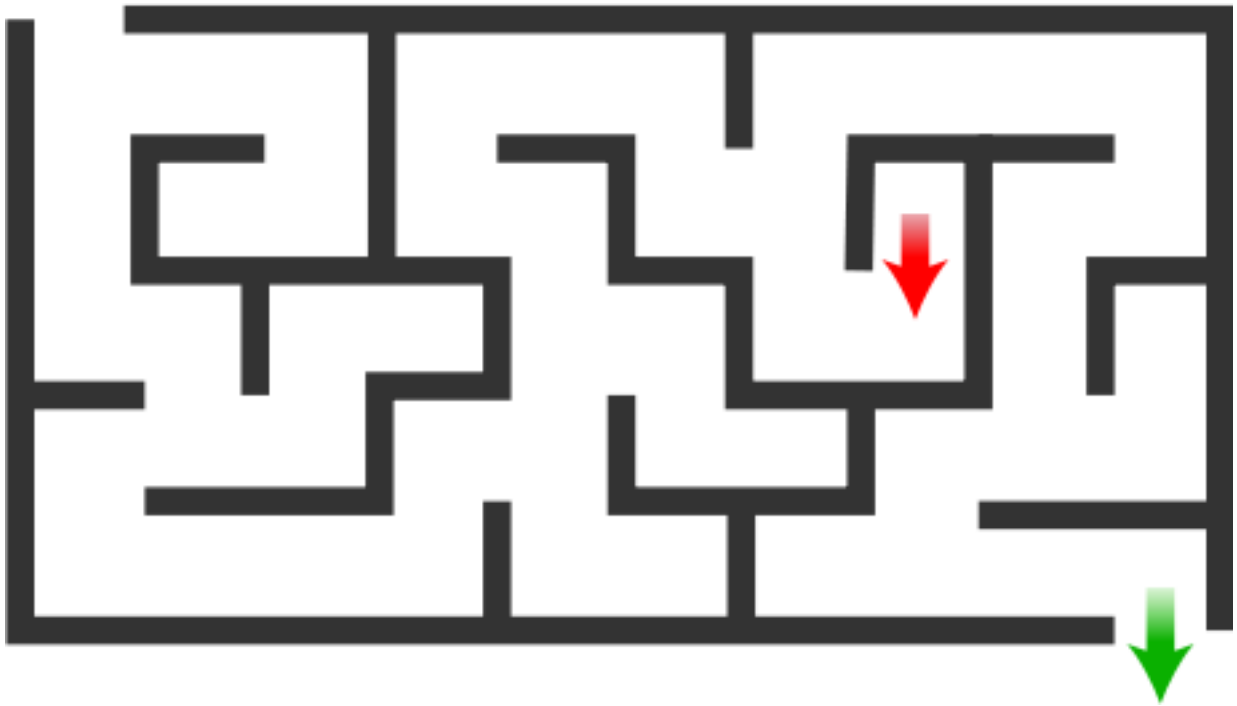
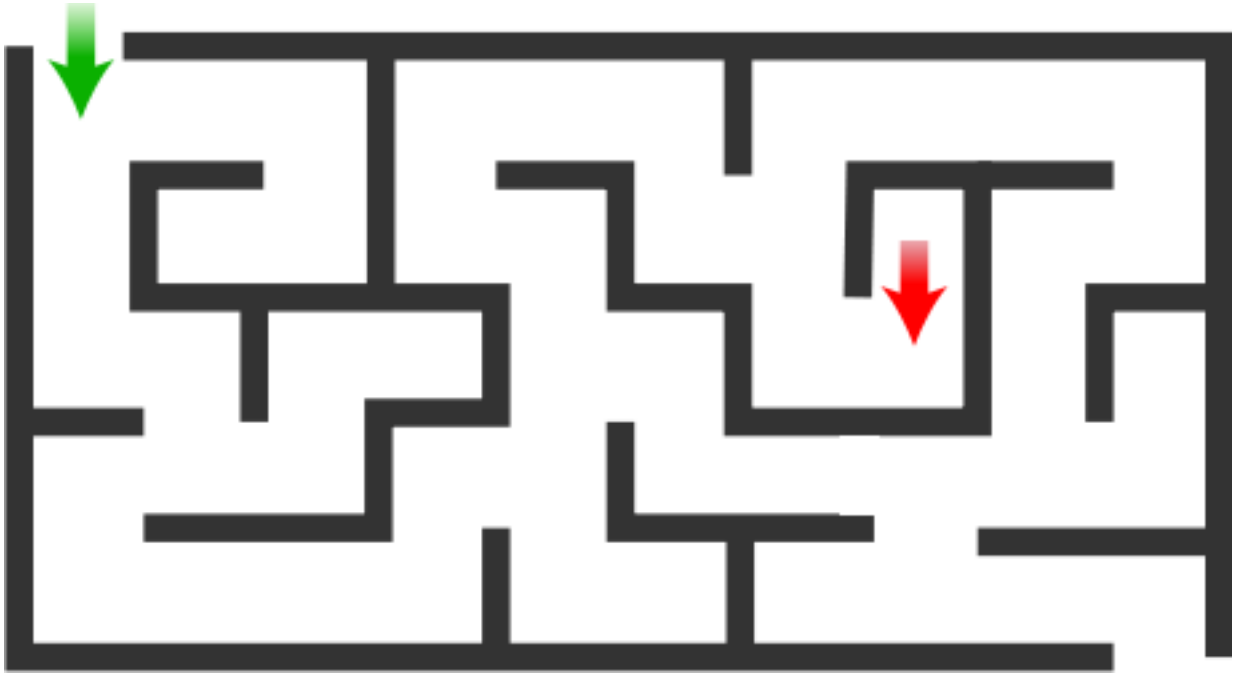


Bild aus wikipedia-Commons

Backtracking-Algorithmen – Labyrinth

- Abgewandeltes Labyrinth-Problem: Minotaurus finden und dann den Ausweg finden
- „Rechte-Hand-Regel“ und „Linke-Hand-Regel“ führen nicht zum Ziel
- Backtracking löst das Problem



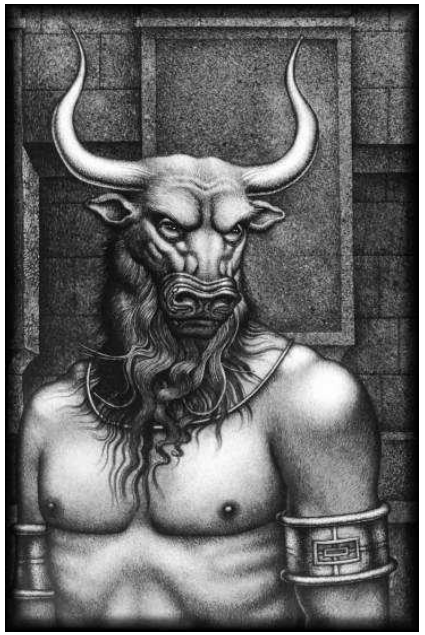


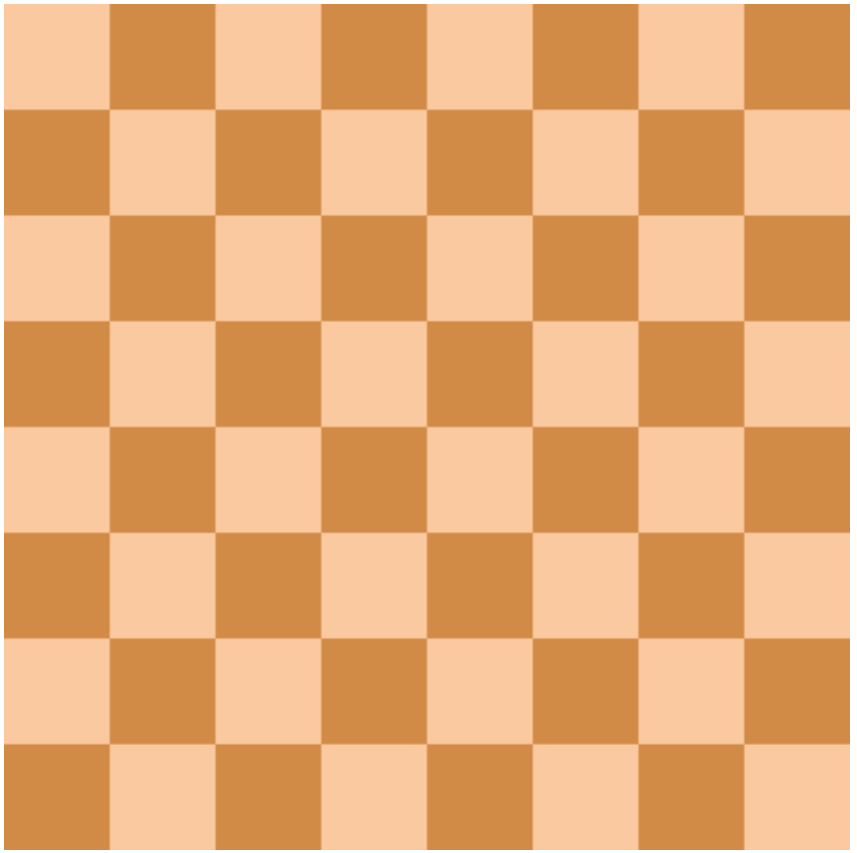
Bild aus wikipedia-Commons (abgewandelt)

Animation des 8-Damen-Problems

- Lösung durch Backtracking
- Es werden immer die am weitesten links liegenden Felder probiert.

- Es gibt 92 Lösungen

- Bei Berücksichtigung von Drehungen und Spiegelungen bleiben 12 Lösungen



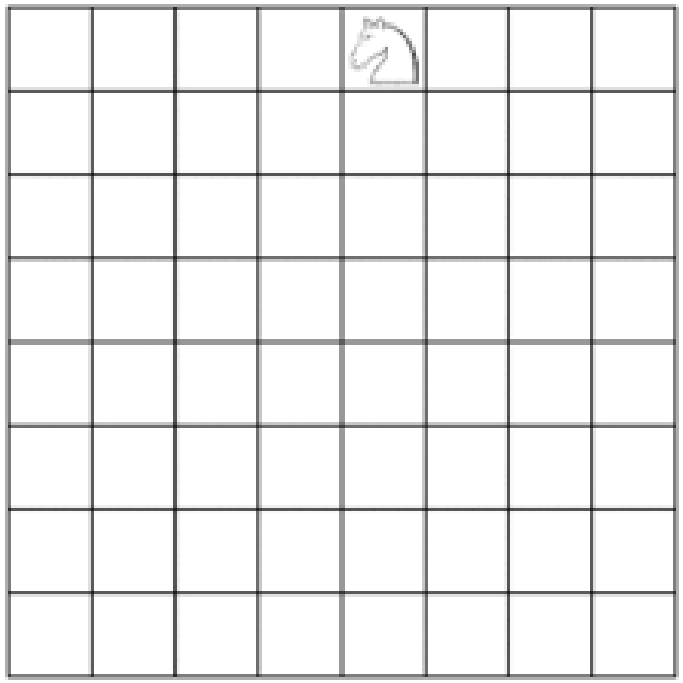
- Gif-Animation aus wikipedia-Commons

Animation einer Lösung des Springer-Problems

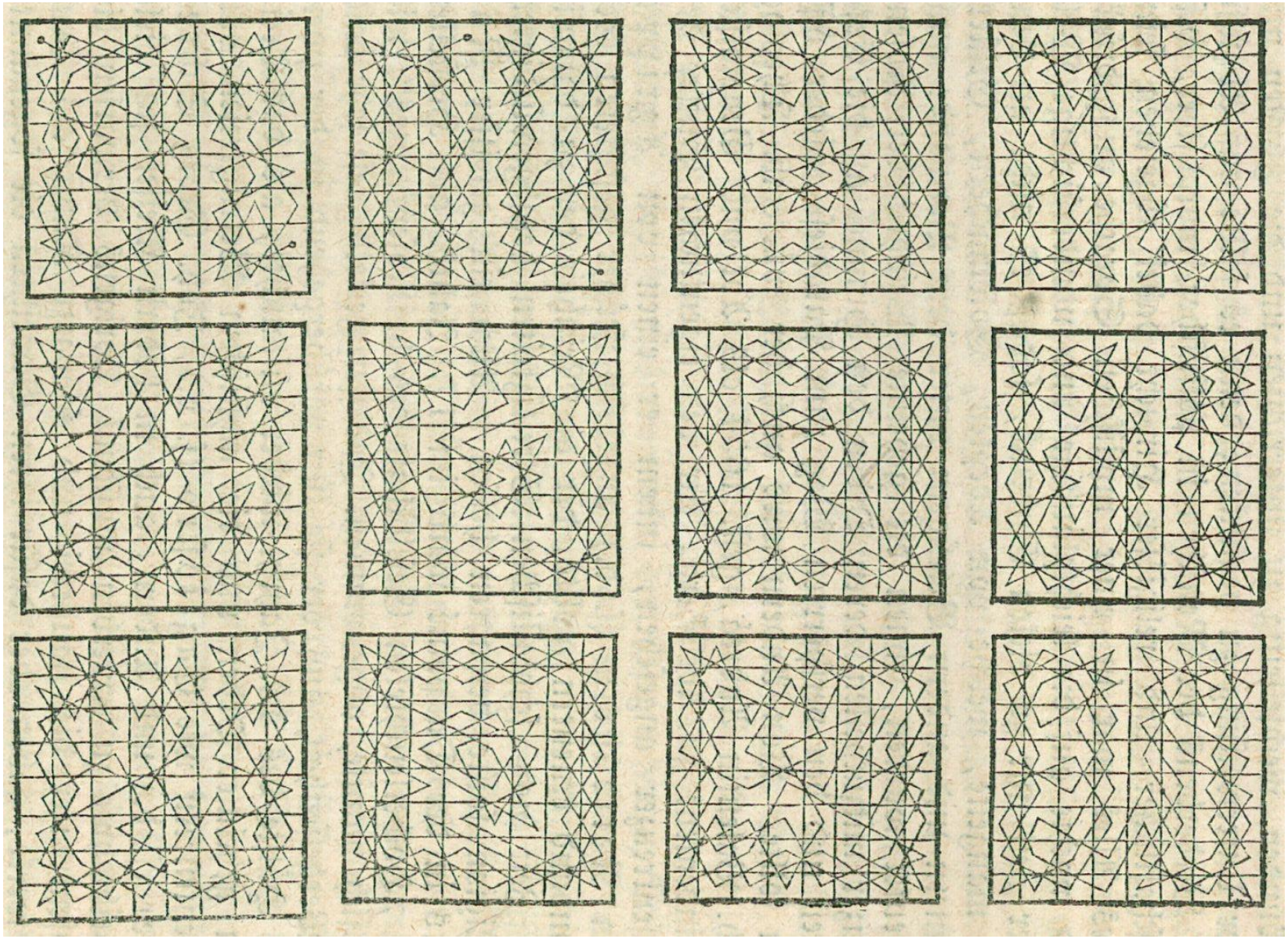
- Das Springerproblem ist verwandt mit dem 8-Damen-Problem

- Es kann auch mit Backtracking gelöst werden
- Der Springer soll jedes Feld genau einmal belegen
- Startfeld und Endfeld können verschieden sein
- Es gibt Lösungen, die beim Startfeld enden (Hamilton-Kreis)

- Es ist kein polynomialer Lösungsalgorithmus bekannt



Lösungen des Springer-Problems



Sudoku

Zurückführen lassen sich Sudokus auf die lateinischen Quadrate, die der Mathematiker Leonhard Euler (1707-1783) präsentierte. Die Quadrate, Euler nannte sie *carré latin*, waren aber nicht an die Größe 9 gebunden, konnten also größer oder kleiner sein und bestanden außerdem nicht aus Teilblöcken.
Frühe Versionen des Sudoku gehen auf französische Zeitungen um 1892 bzw. 1895 zurück. Dies wurde um 1979 unter dem Namen „number place“ in Indianapolis und ab 1984 in Japan unter dem Namen Sudoku wieder aufgegriffen. Der japanische Name bedeutet auf deutsch etwa „einzelne Ziffern“.
Anlässlich eines Japanbesuchs im Jahre 1997 wurde Wayne Gould, ein ehemaliger Richter in Hongkong, auf Sudoku aufmerksam.
Ihm gelang es, die Redaktion der Londoner Times von Sudoku zu überzeugen. Seit dem 12. November 2004 veröffentlicht die Times täglich diese Rätsel.

Sudoku – Lösungsstrategien

- Full House:

  Nur noch eine Zelle in Zeile, Spalte oder Block ist unbelegt

- Scannen (Naked Single):

  In einer Zelle ist nur eine Ziffer möglich.

- Ergänzen (Hidden Single):

  Es gibt in Zeile, Spalte oder Block nur eine Zelle, in der diese Ziffer möglich ist.

- Backtracking:

  Durchlaufen aller Möglichkeiten, bis Lösung gefunden ist.

Direkt polynomiell auf SAT reduzierbar (kombinatorisches Suchproblem).

Spiele Schach, Go

Vollständige Lösungen der Spiele Schach und GO werden in die Klasse PSPACE eingeordnet.
Sie enthält Probleme, zu deren Lösung eine polynomial wachsende Speichermenge und exponentiell wachsende Rechenzeit benötigt wird.

Genetische Algorithmen – Motivation

- Analogie zur Evolutionstheorie der Biologie
- Evolution ist eine erfolgreiche, robuste Methode für Anpassung biologischer Systeme
- GA können Räume von Hypothesen durchsuchen, die komplexe, interagierende Bestandteile enthalten, bei denen der Einfluss jedes Teils auf die Gesamthypothese unklar ist
- GA können leicht parallelisiert werden
- GA sind nicht deterministisch

Problemstellung

- Suche im Raum aller möglichen Hypothesen nach der „besten Hypothese“
- „beste Hypothese“ ist diejenige mit der größten Fitness

Fragen zur Implementierung

- Wie sieht die Fitness-Funktion aus?
- Wie sind die Individuen (Hypothesen) repräsentiert?
- Wie werden die Individuen selektiert?
- Wie reproduzieren sich die Individuen?

Fitness-Funktion

➔ Weist jeder Hypothese einen Fitness-Wert zu

Fitness

➔ Maß für die Güte der Hypothese

Mögliche Kriterien:

- Genauigkeit (z.B. Annähern einer unbekannten Funktion)
- Komplexität
- Allgemeingültigkeit

Genetische Operatoren

- Selektion
- Crossover (Rekombination)
- Mutation
- Erweiterte Operatoren

Creatures

Zusammenfassung

- GA führen eine zufällige, parallele Suche durch nach Hypothesen, die eine vordefinierte Fitness Funktion optimieren
- Dabei wird die natürliche Evolution simuliert
- GA sind „abrupt“ im Gegensatz zum „sanften“ Abstieg entlang des Gradienten wie z.B. bei Backpropagation
  - Nachkommen können radikal verschieden sein von den Eltern
  - Geringere Wahrscheinlichkeit, in einem lokalen Minimum „hängenzubleiben“

Quantencomputer

- Berechnungen beruhen ausschließlich auf Gesetzen der Quantenmechanik
- Bislang noch nicht realisierte universelle Quanten-Turing-maschine (bisher nur experimenteller Status)
- Leistungsfähiger als bisherige Computermodelle für bestimmte Probleme
- Stellen nicht die Church‘sche These in Frage
- Nichtberechenbares bleibt nichtberechenbar.
- Hat nur Auswirkungen auf die Komplexität bestimmter Probleme
- Nur für gewisse Arten von Fragestellungen geeignet

Algorithmen für Quantencomputer

Erster Quantenalgorithmus, 1985 durch David Deutsch (XOR-Problem, keine praktische Anwendung)

gegeben eine „bit to bit" Funktion **f**: $\{0,1\} \rightarrow \{0,1\}$

Aufgabe: ist die Funktion konstant, d.h. **f**(0) = **f**(1)
oder balanciert, d.h. **f**(0) ≠ **f**(1)

Klassisch: Es müssen sowohl **f**(0) als auch **f**(1) ausgewertet werden: → 2 Aufrufe

quantenmechanisch reicht ein einziger Aufruf!
Die Funktion **f** wird auf eine Superposition angewandt.

Verallgemeinerung: Deutsch-Josza (1992)

„n bits to one bit" **f**: $\{0,1\}^n \rightarrow \{0,1\}$
klassisch: worst case $2^{n-1}+1$ Aufrufe

Quantencomputer: 1 Aufruf

David Deutsch formulierte 1985 ein Konzept für eine Quanten-Turingmaschine.

Algorithmen für Quantencomputer

Shor-Algorithmus (Peter Shor 1994)

Aufgabe: Primfaktor-Zerlegung einer b-Bit Zahl (RSA-Krypographie)

541 * 1987 = ? (einfach)

1074967 = ? * ? (schwer)

klassisch: super-polynomial: $O\left(\exp\left(\left(\frac{64}{9}b\right)^{\frac{1}{3}}(\log b)^{\frac{2}{3}}\right)\right)$, bisheriges Optimum

quantenm.: sub-polynomial: $O(b^3)$, probabilistisch

Beispiel für b = 1000 (301-stellig) bei THz-Geschwindigkeit:

klassisch	quantenmechanisch
10^{24} Schritte	10^{10} Schritte
100.000 Jahre	< 1 Sekunde

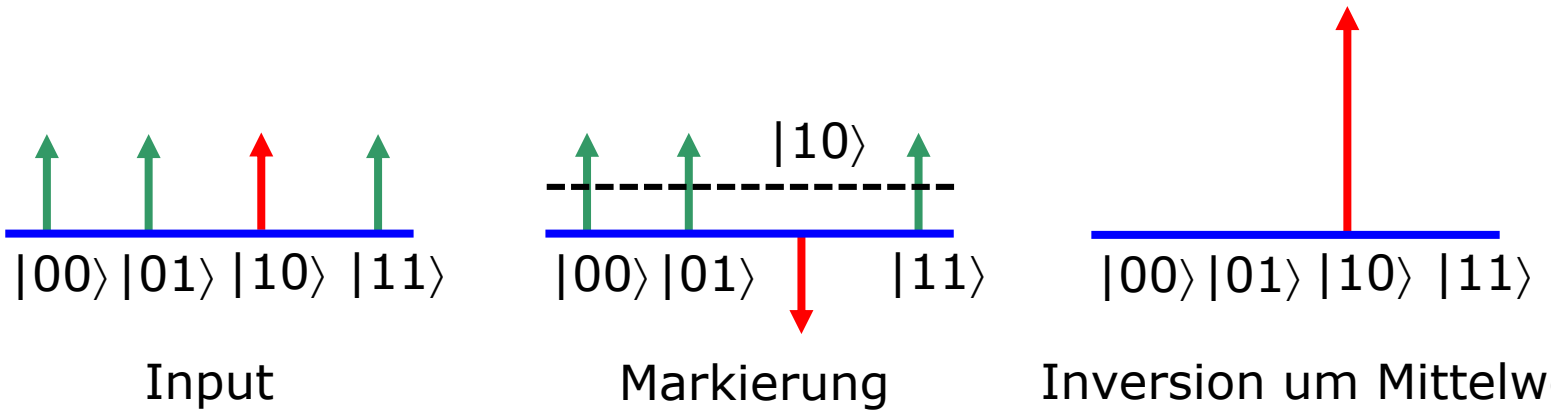
Algorithmen für Quantencomputer

Grover-Algorithmus (1996 durch Lov Grover)

Aufgabe: Datenbank-Suche in einer unsortierten Datenbank mit n Elementen (z.B. eine markierte Seite in einem Buch finden)

klassisch: O(n), es muss im Schnitt das halbe Buch durchblättert werden

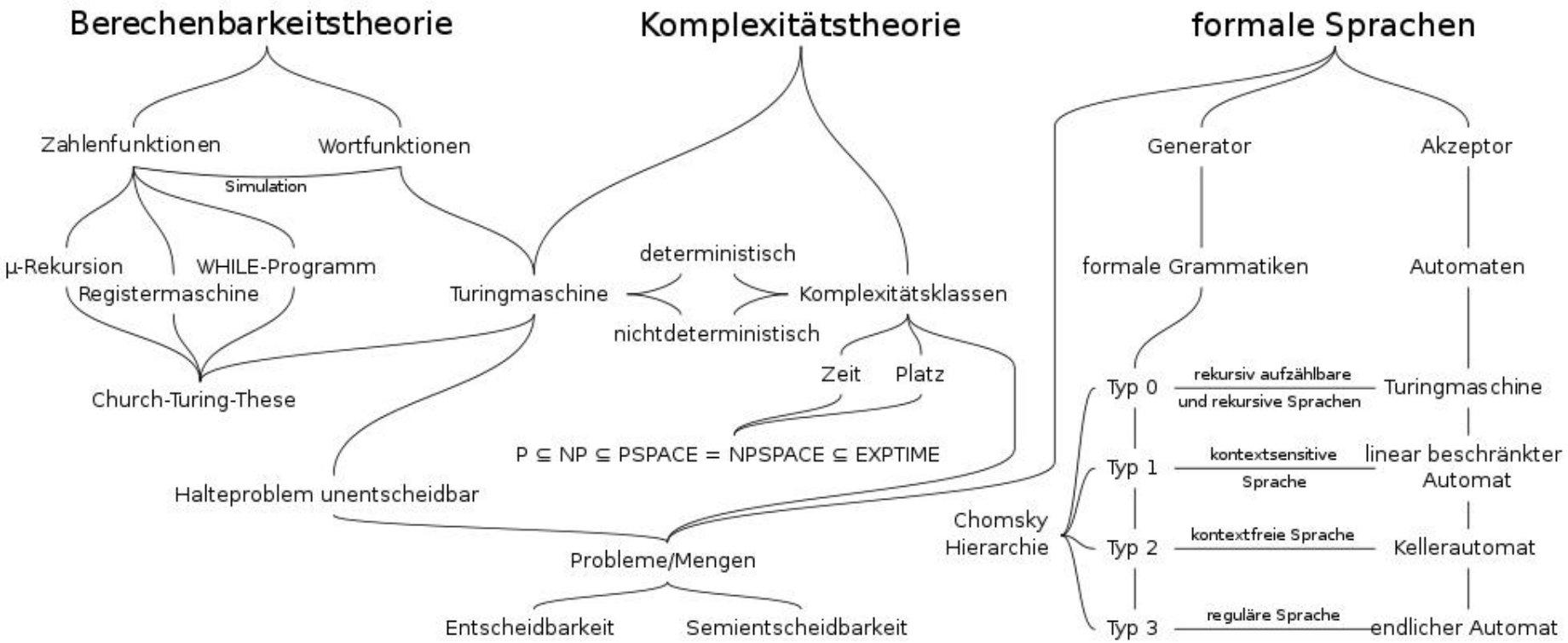
quantenm.: O(√n), „quadratic speed-up" (probabilistisch)



Es wurde bewiesen, dass der Grover-Algorithmus für dieses Problem optimal ist.

Mindmap zur Theoretischen Informatik

Mindmap aus „de.wikipedia.org“



Literaturangaben

[1] Prof. Dr.-Ing. habil. Hans-Ulrich Karl, Dr. rer. nat. Barbara Speck
Technische Universität Dresden, Fakultät Informatik
„Grundlagen der Informatik“,
VMS Verlag Modernes Studieren Hamburg – Dresden GmbH
Bestellnummer: 1039 02 0

[2] Dietmar Herrmann: *„Algorithmen Arbeitsbuch“,*
ADDISON-WESLEY Verlag, ISBN 3-89319-481-9

[3] Uwe Schöning: *„Theoretische Informatik – kurz gefasst“,*
Spektrum Akademischer Verlag, 5. Auflage, 2008

[4] Duden, *„Basiswissen Schule Informatik“*
paetec Gesellschaft für Bildung und Technik mbH, 2003

[5] Robert Sedgewick: *„Algorithmen in C“,* Addison-Wesley 1992
ISBN 3-89319-376-6

[6] Jochen Ziegenbalg: *„Algorithmen von Hammurapi bis Gödel“,*
Spektrum Akademischer Verlag GmbH, ISBN 3-8274-0114-3

Mathematische Symbole

Verdana: < = > + - / * ± ¬ ‹ › ∑ / √ ∞ ∫ ≈ ≠ ≤ ≥ Ø ø ∈ ∉
∪ ∩ ⊂ ⊆ △ ⇒ ⇐ ⇔ ∧ ∨ ∃ ∀ ≡ ↑ ↓ → ✕ ∋ ∀ ∃
α β ω μ ⊥ ⊗ ε ·

Arial: < = > + - / * ± ¬ ‹ › ∑ / √ ∞ ∫ ≈ ≠ ≤ ≥ Ø ø ∈ ∉
∪ ∩ ⊂ ⊆ △ ⇒ ⇐ ⇔ ∧ ∨ ∃ ∀ ≡ ↑ ↓ → ∋ ⊗ ε ·

∃ ∀