Algorithmen und Datenstrukturen (für ET/IT)

Sommersemester 2014

Dr. Tobias Lasser

Computer Aided Medical Procedures
Technische Universität München

Organisatorisches

Weiterer Ablauf:

- Mittwoch, 2. Juli:
  - Zentralübung: zweite Probe-Klausur
- Donnerstag, 3. und Montag, 7. Juli:
  - Vorlesung: Kapitel 11 Datenkompression
- Mittwoch, 9. Juli:
  - Zentralübung: Fragestunde
- Donnerstag, 10. Juli:
  - Vorlesung: Kapitel 12 Kryptographie
- Freitag, 11. Juli:
  - letzte Tutorfragestunde
- Mittwoch, 16. Juli:
  - Klausur

Übersicht Inhalte

Übersicht Inhalte

Nicht Klausur-relevant:

11. Datenkompression
12. Kryptographie

Kapitel 1: Einführung

- Was ist ein Algorithmus?
- Was ist eine Datenstruktur?
- Wo kommen sie vor?
- Einordnung in Computer-Schema

Kapitel 2: Grundlagen von Algorithmen

2.1 Darstellung von Algorithmen

- Pseudocode, Flowchart, Struktogramm, Programmiersprachen
- Churchsche These

2.2 Elementare Bausteine

- Elementarer Verarbeitungsschritt, Sequenz, bedingter Verarbeitungsschritt, Wiederholung

2.3 Logische Ausdrücke

- Konjunktion, Disjunktion, Negation
- Rechenregeln
- Logische Ausdrücke in Pseudocode

Kapitel 3: Grundlagen von Datenstrukturen

3.1 Primitive Datentypen und Zahldarstellung

- Bits und Bytes, Primitive Datentypen
- Dezimal-, Binär-, Hexadezimalsystem, 2-Komplement Darstellung
- Fixed point und Floating point Darstellung

3.2 Felder als sequentielle Liste

- Feld als sequentielle Liste mit Operationen

3.3 Zeichen und Zeichenfolgen

- ASCII Code, Unicode, Zeichen und Strings

3.4 Felder als verkettete Liste

- Feld als (doppelt) verkettete Liste mit Operationen

Kapitel 3: Grundlagen von Datenstrukturen

3.5 Abstrakte Datentypen

- Abstrakter Datentyp, abstrakte Variable

3.6 Stacks

- Stack mit Operationen und Implementationen (sequentielle / verkettete Liste)

3.7 Queues

- Queue mit Operationen und Implementationen (verkettete Liste, (zirkuläre) sequentielle Liste)

Kapitel 4: Grundlagen der Korrektheit von Algorithmen

4.1 Motivation und Spezifikation

- Relative Korrektheit, Nachweis durch Verifikation, Validation

4.2 Verifikation

- Vor- und Nachbedingungen, partielle und totale Korrektheit
- Korrektheitsbeweis für 4 Bausteine

4.3 Beispiel: Insertion Sort

- Algorithmus und Verifikation

4.4 Validation

- Systematische Tests: Blackbox-, Whitebox-, Regression-, Integrations-Test
- Fehlerquellen, fehlertolerantes und fehlerpräventives Programmieren

Kapitel 5: Grundlagen der Effizienz von Algorithmen

5.1 Motivation

- Komplexität Insertion Sort, Wachstumsraten

5.2 RAM-Modell

- Rechnermodell zur Laufzeitanalyse

5.3 Landau-Symbole

- Landau-Symbole O und Θ, Konstanten und Rechenregeln
- Kategorisierung der Effizienz von Algorithmen
- Komplexität der 4 Bausteine

Kapitel 6: Grundlagen des Algorithmen-Entwurfs

6.1 Entwurfsprinzipien

- Prinzip Verfeinerung, Prinzip Algorithmen-Muster

6.2 Divide and Conquer

- Muster Divide and Conquer, MergeSort, QuickSort
- Ausblick auf später: Binäre Suche, Strassen, FFT

6.3 Greedy-Algorithmen

- Muster Greedy, Wechselgeld, minimaler Spannbaum
- Ausblick auf später: Dijkstra, Prim
  (Huffman Code - nicht Klausur-relevant)

6.4 Backtracking

- Muster Backtracking, Labyrinth, Traveling Salesman, Acht-Damen-Problem

6.5 Dynamisches Programmieren

- Muster dynamisches Programmieren, Fibonacci-Zahlen

Ende Teil 1

Ende Teil 1, nun Teil 2

Kapitel 7: Fortgeschrittene Datenstrukturen

Kapitel 7.1: Graphen

- Ungerichteter Graph:

Definition: Ungerichteter Graph

Ein ungerichteter Graph ist ein Paar $G = (V, E)$ mit

- V endliche Menge der Knoten
- $E \subseteq \big\{\{u, v\} : \ u, v \in V\big\}$ Menge der Kanten

- Gerichteter Graph:

Definition: Gerichteter Graph

Ein gerichteter Graph ist ein Paar $G = (V, E)$ mit

- V endliche Menge der Knoten
- $E \subseteq V \times V$ Menge der Kanten

- Gewichteter Graph:

Definition: Gewichteter Graph

Ein gewichteter Graph ist ein Graph $G = (V, E)$ mit einer Gewichtsfunktion $w : E \to \mathbb{R}$.

Kapitel 7.1: Graphen

- Eigenschaften von Graphen:
  - adjazente Knoten
  - gerichteter Graph: Eingangs- und Ausgangsgrad
  - ungerichteter Graph: Grad
  - Pfad zwischen Knoten
  - Zyklus in gerichteten und ungerichteten Graphen
  - (starker) Zusammenhang und (starke) Zusammenhangskomponenten
- Darstellung von Graphen:
  - Adjazenzmatrizen, Speicherkomplexität: $O(|V|^2)$
  - Adjazenzlisten, Speicherkomplexität: $O(|V| + |E|)$
  - Komplexitäten der Operationen
    - Kanten-Operationen einfach mit Adjazenzmatrix: $O(1)$
    - Knoten einfügen einfach mit Adjazenzliste: $O(1)$
    - Knoten löschen immer aufwendig: $O(|V|^2)$ bzw. $O(|V| + |E|)$

Kapitel 7.2: Bäume

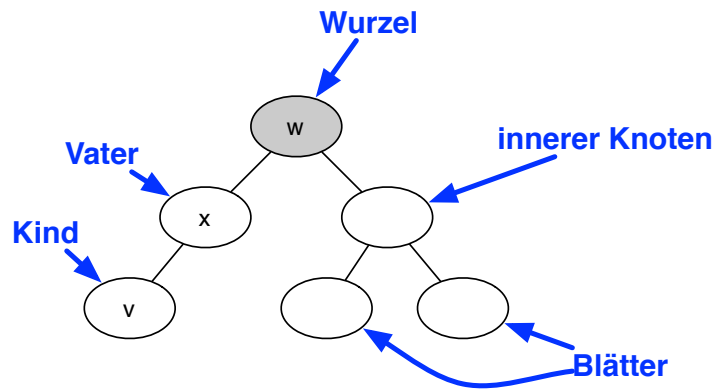
- Bäume und Wälder:

Definition: Wald und Baum

- Ein azyklischer ungerichteter Graph heißt auch Wald.
- Ein zusammenhängender, azyklischer ungerichteter Graph heißt auch Baum.

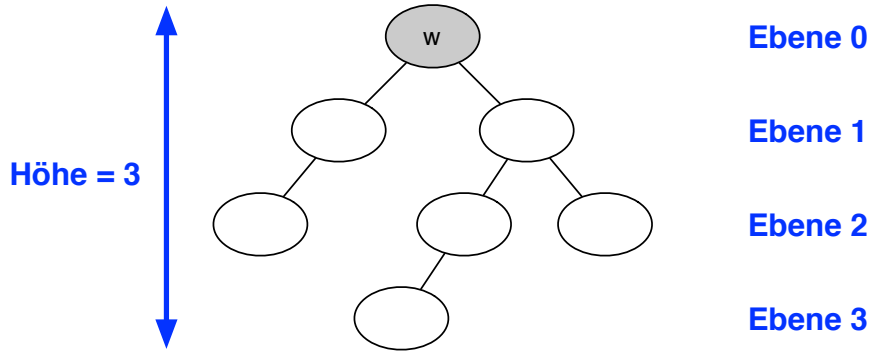
  - falls Kante entfernt: nicht mehr zusammenhängend
  - falls Kante hinzugefügt: nicht mehr azyklisch
  - $|E| = |V| - 1$

- Eigenschaften:
  - Wurzelknoten, Vaterknoten, innere Knoten, Blätter



Kapitel 7.2: Bäume

- Eigenschaften:
  - Tiefe, Ebene, Höhe



- Spezialfälle:
  - geordnete Bäume ($\rightarrow$ Suchbäume)
  - n-äre Bäume
  - Binärbäume
- Darstellung von Bäumen:
  - mit (doppelt) verketteten Listen
  - Binärbäume mit (doppelt) verketteten Listen (Kind links/rechts) bzw. als sequentielle Liste

Kapitel 7.2: Bäume

- Traversierung von Binärbäumen:
  - Tiefensuche/DFS: Pre-order (wlr), In-order (lwr), Post-order (lrw)
    Implementierung z.B. mit Stack
  - Breitensuche/BFS
    Implementierung z.B. mit Queue

Kapitel 7.3: Heaps

- Heap:

Definition Heap

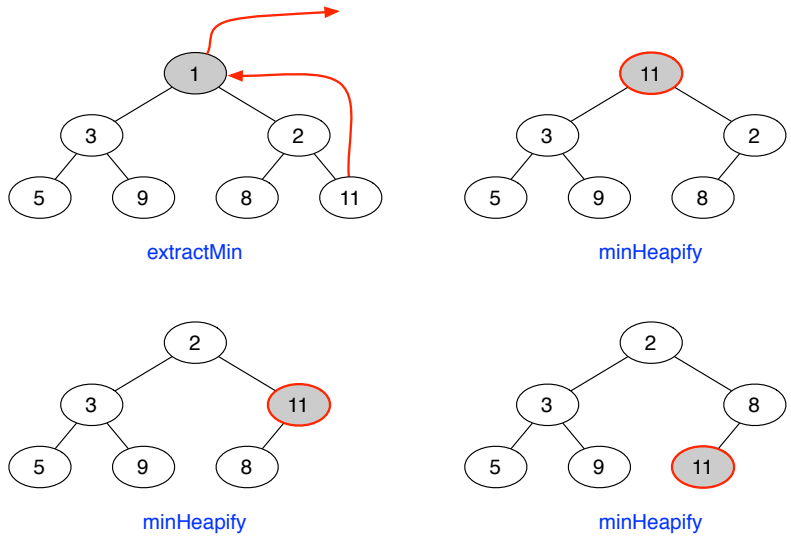
Sei $G = (V, E)$ ein Binärbaum mit Wurzel $w \in V$. Jeder Knoten $v \in V$ sei mit einem Wert $key(v)$ verknüpft, die Werte seien durch $\leq, \geq$ geordnet.
G heißt Heap, falls er folgende zwei Eigenschaften erfüllt:

- G ist fast vollständig, d.h. alle Ebenen sind vollständig gefüllt, ausser auf der untersten Ebene, die von links her nur bis zu einem bestimmten Punkt gefüllt sein muss.
- G erfüllt die Min-Heap-Eigenschaft bzw. die Max-Heap-Eigenschaft, d.h. für alle Knoten $v \in V$, $v \neq w$ gilt
  - Min-Heap: $key(v.vater) \leq key(v)$
  - Max-Heap: $key(v.vater) \geq key(v)$

Entsprechend der Heap-Eigenschaft heißt G Min-Heap bzw. Max-Heap.

Kapitel 7.3: Heaps

- Operationen:
  - minHeapify: $O(\log n)$
  - buildMinHeap: $O(n)$
  - extractMin: $O(\log n)$



- HeapSort:
  - buildMinHeap und extractMin bis leer
  - in-place, $O(n \log n)$, besser als QuickSort im worst case

Kapitel 7.4: Priority Queues

- Priority Queue:

Definition Priority Queue

Eine Priority Queue ist ein abstrakter Datentyp. Sie beschreibt einen Queue-artigen Datentyp für eine Menge von Elementen mit zugeordnetem Schlüssel und unterstützt die Operationen

- Einfügen von Elemente mit Schlüssel in die Queue,
- Entfernen von Element mit minimalem Schlüssel aus der Queue,
- Ansehen des Elementes mit minimalem Schlüssel in der Queue.

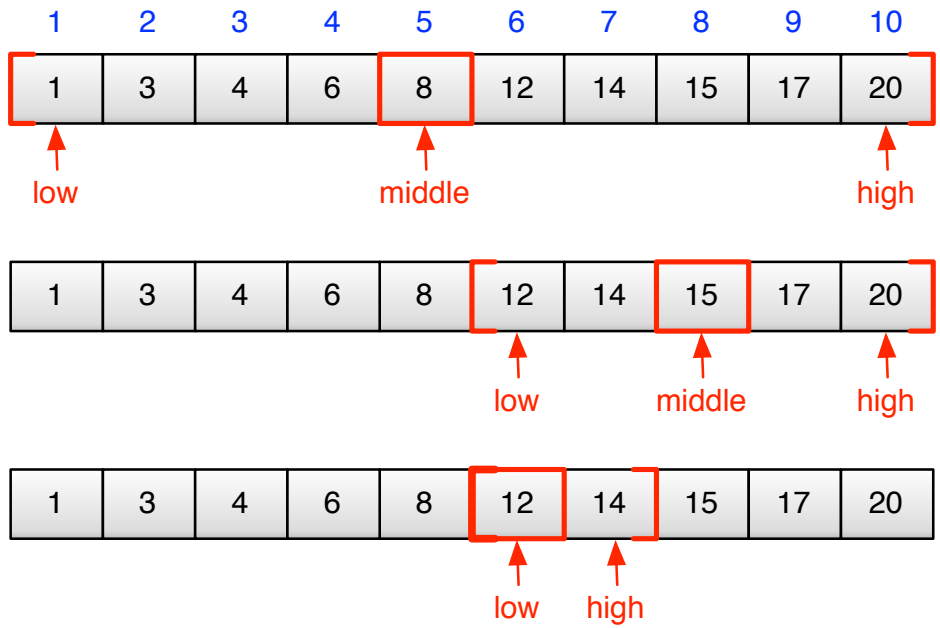
- Operationen auf Priority Queue als Heap:
  - extractMin und insert, beides $O(\log n)$
  - decreaseKey: $O(\log n)$

- auch implementierbar mit (un)sortierten Feldern

Kapitel 8: Such-Algorithmen

Kapitel 8.1 und 8.2: Lineare und binäre Suche

- Lineare Suche: durchlaufe Array bis gesuchter Schlüssel gefunden: $O(n)$
- Binäre Suche: sortiere Array, dann Divide and Conquer: $O(\log n)$



Kapitel 8.3: Binäre Suchbäume

- Binärer Suchbaum:

Definition binärer Suchbaum

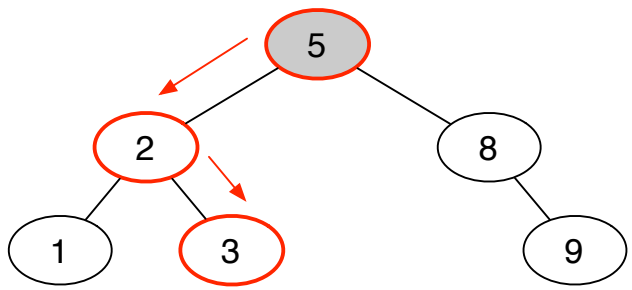
Sei $G = (V, E)$ ein Binärbaum mit Wurzel $w \in V$. Jeder Knoten $v \in V$ sei mit einem Wert $key(v)$ verknüpft, die Werte seien durch $\leq, \geq$ geordnet.
G heißt binärer Suchbaum, falls für alle inneren Knoten $v \in V$ gilt

- für alle Knoten x im linken Teilbaum $v.left$ gilt

$$key(x) \leq key(v)$$

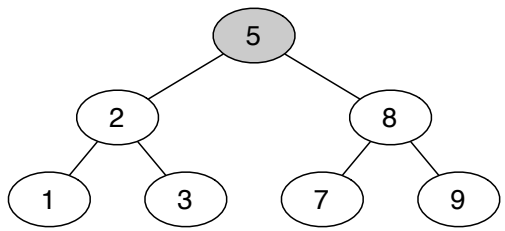
- für alle Knoten y im rechten Teilbaum $v.right$ gilt

$$key(y) \geq key(v)$$

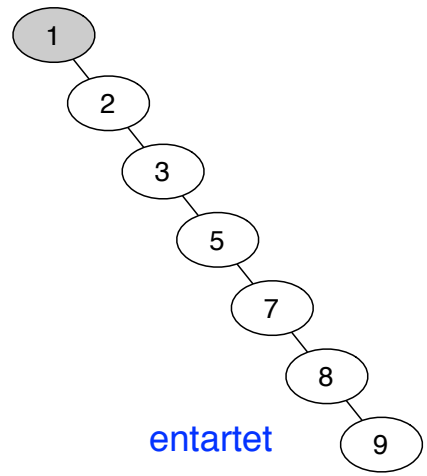


Kapitel 8.3: Binäre Suchbäume

- Operationen auf binären Suchbäumen mit Höhe h:
  - Suchen, Minimum, Maximum: $O(h)$
  - Einfügen, Löschen: $O(h)$
- Effizienz direkt abhängig von Höhe
  - best case: $O(\log n)$
  - worst case: $O(n)$



ausgeglichen



entartet

- Ausweg: balancierte Suchbäume

Kapitel 8.4: Balancierte Suchbäume

- Problem: binären Suchbaum balanciert zu halten ist aufwendig: $O(n)$
- Ausweg: Abschwächung der Kriterien
  - balanciert: AVL-Bäume
  - Binärbaum: B-Bäume, Rot-Schwarz-Bäume
- AVL-Bäume:

Definition AVL-Baum

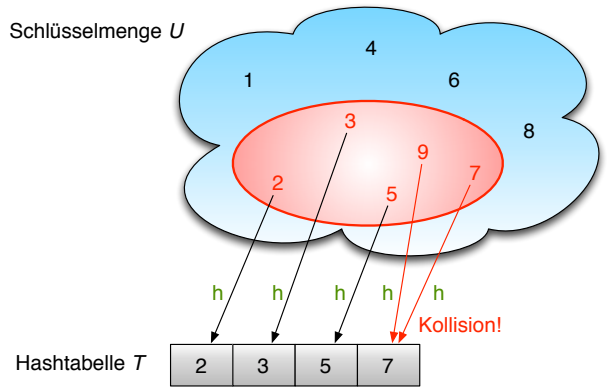
Ein binärer Suchbaum $G = (V, E)$ mit Wurzel $w \in V$ heißt AVL-Baum, falls er die AVL-Bedingung erfüllt:

- für jeden inneren Knoten $v \in V$ gilt: Höhe von linkem und rechtem Teilbaum von v unterscheidet sich maximal um 1.

  - Anpassung von Einfügen, Löschen mit Rotation und Doppelrotation

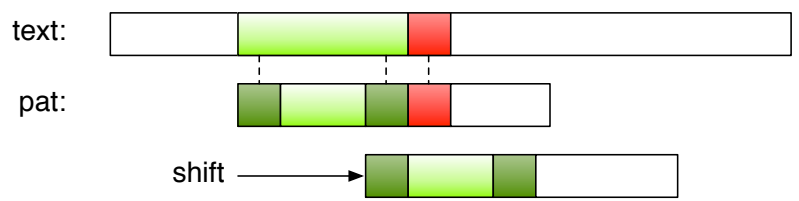
Kapitel 8.5: Suchen mit Hashtabellen

- Motivation: Such-Algorithmen für Wörterbücher mit Komplexität O(1)
  - Operationen: search, insert, erase
- Adresstabellen: Laufzeit O(1), aber gigantischer Speicherverbrauch!
- Hashtabellen: reduziere Speicherverbrauch über Hashfunktion, behalte Laufzeit O(1)
  - Kollisionsauflösung durch Verkettung, offene Adressierung
  - Hashfunktionen mittels Divisionsmethode, Multiplikationsmethode, universelles Hashing



Kapitel 8.6: Suchen in Zeichenketten

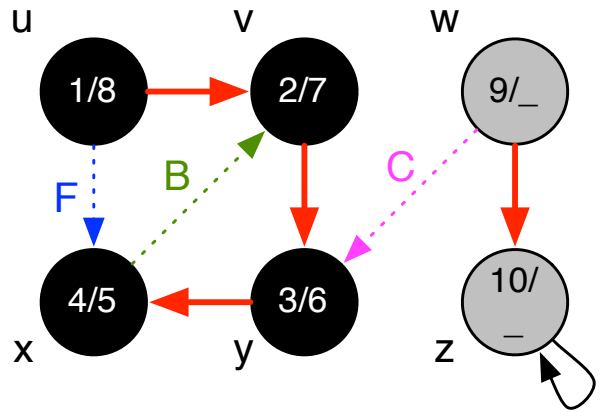
- Problem: suche Muster (Länge m) in Text (Länge n)
- Brute-Force Algorithmus: teste an jeder Textstelle $O(nm)$
- Knuth-Morris-Pratt Algorithmus: $O(n+m)$
  - Shift-Tabelle: Länge des eigentlichen Randes des Präfixes von Muster
  - modifizierter Brute-Force-Algorithmus:
    - 0 in Shift-Tabelle: direkt weitersuchen ohne zurückspringen
    - > 0 in Shift-Tabelle: mitten in Muster weitersuchen



Kapitel 9: Graph-Algorithmen

Kapitel 9.1: Tiefensuche

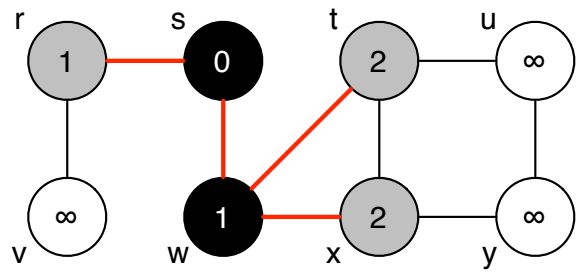
- Tiefensuche / DFS: $O(|V| + |E|)$



- Implementation rekursiv oder mit Stack
- Anwendungen:
  - berechnet Spannwald von Graph, markiert Rückkanten, Vorwärtskanten und Cross-Kanten
  - Test aus Zusammenhang
  - Test auf Zyklenfreiheit
  - Segmentierung von binären Bildern mit Nachbarschafts-Graph

Kapitel 9.2: Breitensuche

- Breitensuche / BFS: $O(|V| + |E|)$



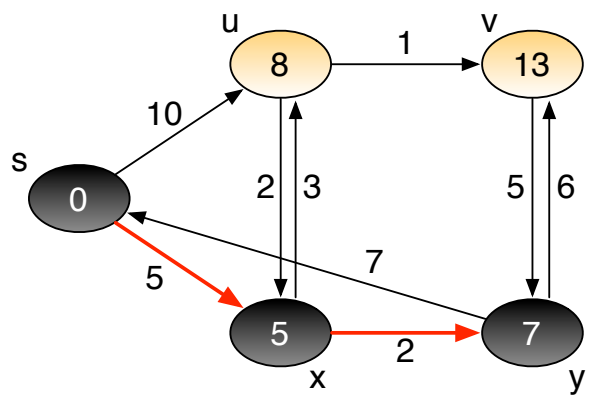
- Implementation mit Queue
- Anwendungen:
  - Besuch von Knoten in Zusammenhangskomponente
  - Berechnung Länge der kürzesten Pfade (d.h. Anzahl von Kanten)
  - Segmentierung von binären Bildern mit Nachbarschafts-Graph

Kapitel 9.3: Kürzeste Pfade

- Kürzester Pfad in gewichtetem Graph: Pfad mit minimaler Summe der Gewichte

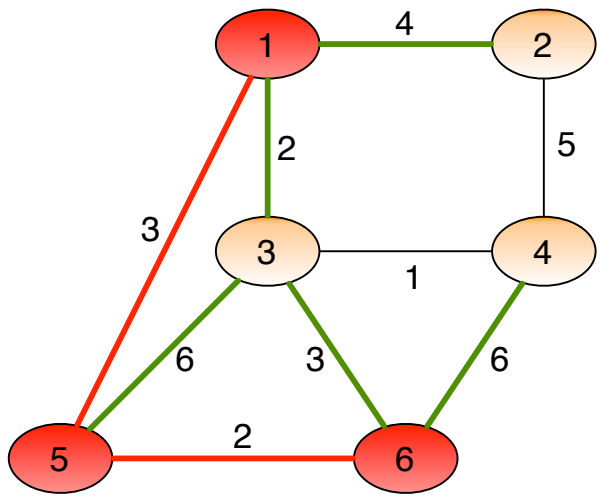
$$w_{min}(u,v) = \begin{cases} \min\{w(p) : p \text{ Pfad von } u \text{ nach } v\} & \text{Pfad existiert} \\ \infty & \text{sonst} \end{cases}$$

- Dijkstra Algoritmus: Greedy-Algorithmus, $O(|E| \log |V|)$
  - Voraussetzung: Kanten haben positives Gewicht
  - erweitere Pfad um kürzeste Kante heraus aus Menge aktueller Knoten
  - entscheidendes Hilfsmittel: Priority Queue



Kapitel 9.4: Minimale Spannbäume

- Spannbaum: Teilgraph, der Baum ist und alle Knoten enthält
- minimaler Spannbaum: Spannbaum mit minimalem Gewicht
- Prim Algorithmus: Greedy-Algorithmus, $O(|E| \log |V|)$
  - erweitere aktuelle Lösung um Kante mit minimalen Gewicht heraus aus Menge aktueller Knoten
  - entscheidendes Hilfsmittel: Priority Queue



Kapitel 10: Numerische Algorithmen

Kapitel 10.1: Matrizen

- Matrix als Element von Vektorraum, lineare Abbildung, Anordnung von Zahlen
- Matrix-Operationen: Addition, Skalar-Multiplikation, Matrix-Multiplikation
- Matrix-Multiplikation:
  - regulär: $O(n^3)$
  - Strassen (Divide and Conquer): $O(n^{2.807})$

$$A \cdot B = \begin{pmatrix} \mathbf{q}_1 + \mathbf{q}_4 - \mathbf{q}_5 + \mathbf{q}_7 & \mathbf{q}_3 + \mathbf{q}_5 \\ \mathbf{q}_2 + \mathbf{q}_4 & \mathbf{q}_1 + \mathbf{q}_3 - \mathbf{q}_2 + \mathbf{q}_6 \end{pmatrix}$$

Kapitel 10.2: Lineare Gleichungen

- Lineare Gleichung: $a^T x = b$
- Lineares Gleichungssystem: $Ax = b$
  - Anzahl Lösungen: keine, eine oder unendlich viele
- Lösen von linearen Systemen:
  - mit Inverser A^{-1} (numerisch nicht stabil)
  - mit Gauss-Elimination (numerisch nicht stabil)
  - mit Zerlegung und Vorwärts-/Rückwärts-Substitution (stabil mit geeigneter Zerlegung)
  - mit iterativen Verfahren

Kapitel 10.2: Lineare Gleichungen

- Matrix-Zerlegungen:
  - Cholesky-Zerlegung für symm. pos. def. Matrizen ($\frac{n^3}{3}$ FLOPs)
  - QR-Zerlegung für invertierbare Matrizen ($\frac{2n^3}{3}$ FLOPs)
  - LUP-Zerlegung für invertierbare Matrizen ($\frac{2n^3}{3}$ FLOPs)
  - SVD-Zerlegung für beliebige Matrizen ($2mn^2 + 2n^3$ FLOPs)

Kapitel 10.3: Die LUP-Zerlegung

- LU und LUP-Zerlegung
- Permutationsmatrizen und Darstellung
- Berechnung der LU- und LUP-Zerlegung:
  - rekursive Berechnung von Gauss-Eliminations-Schritten (Faktorisierung mittels Schur-Komplement)
  - partielles Pivoting zur Vermeiden von Division durch 0 und besserer Stabilität

Kapitel 10.4: Least squares Probleme

- minimiere Approximationsfehler

$$\eta_j = F(x_j) - y_j$$

für Datenpunkte (x_j, y_j) und Funktion $F(x) = \sum_i c_i f_i(x)$

Least-squares Lösung

Sei $A \in \mathbb{R}^{m \times n}$, $y \in \mathbb{R}^m$ mit $m \geq n$. Eine Lösung $c \in \mathbb{R}^n$ des Minimierungs-Problems

$$\min_c \|Ac - y\| \qquad \text{or} \qquad \min_c \|Ac - y\|^2$$

heißt Least-squares Lösung.

- Normalengleichung:

$$A^T A c = A^T y$$

- Pseudoinverse

$$A^+ = (A^T A)^{-1} A^T$$

und Berechnung via QR/SVD Zerlegung

Kapitel 10.5: Fast Fourier Transform

- Polynome in Koeffizientendarstellung:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

- Operationen:
  - Auswertung: $O(n)$ mit Horner-Schema
  - Addition: $O(n)$
  - Multiplikation: $O(n^2)$
- Faltung zweier Vektoren entspricht Polynom-Multiplikation

Kapitel 10.5: Fast Fourier Transform

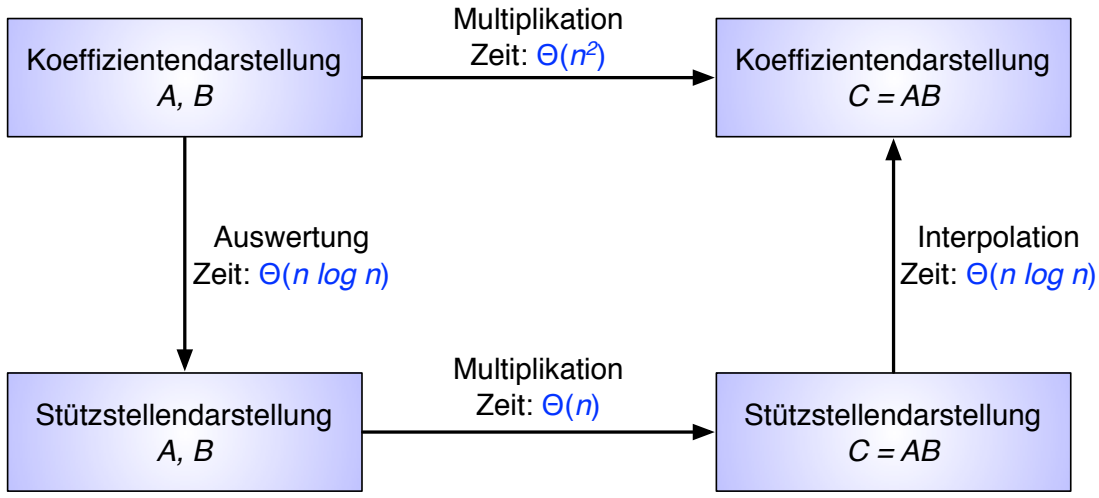
- Polynome in Stützstellendarstellung:

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

- Operationen:
  - Addition: $O(n)$
  - Multiplikation: $O(n)$
- Konversion zwischen Darstellungen:
  - Koeffizienten $\rightarrow$ Stützstellen: Auswertung mit FFT $O(n \log n)$
  - Stützstellen $\rightarrow$ Koeffizienten: Interpolation mit IFFT $O(n \log n)$

Kapitel 10.5: Fast Fourier Transform

- Effiziente Faltung / Polynom-Multiplikation:



- Diskrete Fourier Transformation: Polynom-Auswertung an n-ten komplexen Einheitswurzeln, $O(n^2)$
- Fast Fourier Transformation: DFT mit Divide and Conquer und Halbierungslemma, $O(n \log n)$
  - Implementierung z.B. entrekursiviert mit Butterfly-Schema

Ausblick: Kapitel 11 und 12

Nicht Klausur-relevant!

11. Datenkompression
    - 11.1 Einführung
    - 11.2 Grundlagen der Informationstheorie
    - 11.3 Huffman Codes
    - 11.4 LZ77 und LZ78
    - 11.5 JPEG
12. Kryptographie
    - 12.1 Einführung
    - 12.2 Symmetrische Verschlüsselung
    - 12.3 RSA

Anwendungs-Szenarien in der Tomographie

Nicht Klausur-relevant!

☞ separate Präsentation