Lernzettel

Algorithmen und Datenstrukturen

Universität: Technische Universität Berlin
Kurs/Modul: Algorithmen und Datenstrukturen
Erstellungsdatum: September 6, 2025

Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

https://study.AllWeCanLearn.com

Algorithmen und Datenstrukturen

Lernzettel: Algorithmen und Datenstrukturen

(1) Grundlagen. Ein Algorithmus ist eine endliche Folge von Schritten zur Lösung eines Problems. Die Komplexität beschreibt, wie der Ressourcenbedarf (Zeit, Speicher) mit der Input-Größe n skaliert. Typische Größenordnungen sind:

$$O(f(n)) \quad \text{und} \quad \Theta(g(n)).$$

Beispiele:

$$O(n), \quad O(n \log n), \quad O(1), \quad \Theta(n^2).$$

(2) Aufwand- und Korrektheitsnachweise.

- Worst-case-Analyse, ggf. Average-case.
- Korrektheitsnachweis durch Invariante, Beleg durch Induktion.

Bei einer Invariante handelt es sich um eine Bedingung, die zu jedem Schritt des Algorithmus wahr ist und am Ende die Korrektheit sicherstellt.

(3) Relevante Datenstrukturen. Arrays: direkter Zugriff per Index, feste Größe, schnelle Abfragen, teils teure Einfügungen.

Verkettete Listen: flexible Größe, Einfügen am Anfang O(1), Zufallszugriff O(n).

Stapel (Stack) und **Warteschlange (Queue)**: LIFO bzw. FIFO; Grundoperationen Push/Pop bzw. Enqueue/Dequeue.

Bäume: Binärbäume, Suchbäume; balancierte Varianten wie AVL/RB-Bäume haben garantierte Baumhöhe O(log n).

Mengen und Karten (Hash-Tabellen): Durchschnittliche O(1) für Einfügen, Suchen; Kollisionsbehandlung notwendig.

Graphen-Repräsentationen: *Adjazenzliste* (sparsam, gut für spärlich besetzte Graphen) und *Adjazenzmatrix* ($\mathrm{O(n^2)} Speicher, direkter Zugriff$).

(4) Graphen und Graph-Repräsentationen. Gerichtete und ungerichtete Graphen, **gewichtete Graphen**. Wichtige Begriffe: Knoten (Vertices), Kanten (Edges), Kapazität c(u,v), Pfad, Fluss f(u,v).

(5) Flussprobleme. Max-Flow Problem: Gegeben ein Flussnetz, wähle Flüsse f(u,v) unter 0 f(u,v) c(u,v) und Flussbilanz an jedem Knoten außer Quelle s und Senke t, um den Gesamtfluss zu maximieren:

$$\max \sum_{v:(s,v)\in E} f(s,v)$$

unter

$$\sum_u f(u,v) = \sum_w f(v,w) \quad \text{für alle } v \neq s,t.$$

Min-Cut ist der minimal mögliche Kapazitätsschnitt, der den Graph in zwei Teile teilt, sodass s und t getrennt sind. Max-Flow = Min-Cut (Max-Flow-Min-Cut-Theorem).

Beispiele von Algorithmen:

- Ford-Fulkerson: wiederholtes Finden einer augmentierenden Kante;

- Edmonds-Karp: BFS-basiert, garantiert $O(|V||E|^2) Laufzeit$.

(6) Optimierungsalgorithmen. Branch and Bound: systematische Baumsuche mit Schranken für Pruning. **Backtracking**: rekursive Suche mit Zurückspringen bei Ungültigem.

Scheduling (Zeitplanung): typische Probleme wie Job-Shop oder DAG-basierte Aufgaben mit Zuweisung von Startzeiten, Minimierung von Gesamtdauer oder Verzögerungen. Wichtige Konzepte: Heuristiken, Basisoptimierung, Heuristik- und Exaktnachweise.

(7) Sortier- und Suchalgorithmen (Auswahl).

- Sortieralgorithmen: QuickSort (durchschnittlich O(n log n)), MergeSort, HeapSort.
- Suchalgorithmen: lineare Suche, binäre Suche in sortierten Arrays.

(8) Mengen- und Graphrepräsentationen. - Mengenoperationen (Vereinigen, Schnitt, Differenz) auf Sets. - Graphrepräsentationen wählen: Adjazenzliste bei vielen Kanten; Adjazenzmatrix bei dichten Graphen.

(9) Hinweise zur Java-Implementierung.

- Von Grund auf: Klassen für Knoten, Kanten, Graphen, Flüsse.
- Nutzung der Java-Collections (List, Map, Set) zur passenden Datenhaltung; Laufzeiten beachten.
- Für Graphen: zuerst Repräsentation wählen (Liste vs Matrix) und danach Algorithmen implementieren (BFS/DFS, Dijkstra, Flussalgorithmen).

Hinweis zur Prüfungsvorbereitung. - Verstehe die Definitionen und Beziehungen zwischen Max-Flow, Min-Cut und den relevanten Invarianten. - Übe das Ablesen von Laufzeitkomplexitäten anhand kleiner Beispiele. - Übe einfache Pseudocode-Schnipsel in Java-Form (ohne vollständigen Programmierschaden).