Lernzettel

Einführung in die Informatik - Vertiefung

Universität: Technische Universität Berlin
Kurs/Modul: Einführung in die Informatik - Vertiefung
Erstellungsdatum: September 20, 2025

Lernzettel: Einführung in die Informatik - Vertiefung

(1) Überblick und Zielsetzung. Die Vertiefung baut auf den Grundlagen der Informatik auf und vertieft zentrale Themen der Programmierung, der Datenstrukturen und der Algorithmen. Ziel ist es, Probleme algorithmisch zu modellieren, fortgeschrittene Java-Programmierung zu beherrschen und die Effizienz sowie Korrektheit von Algorithmen im jeweiligen Anwendungsgebiet zu beurteilen. Anwendungen aus dem Fachgebiet und dem späteren Berufsleben sollen durch praktische Implementierungen gefestigt werden.

(2) Generische Datentypen, Iteratoren und Abstrakte Datentypen. - Generische Datentypen ermöglichen Typ-Parameterisierung von Strukturen wie `List<T>` oder `Map<K,V>`. - Iteratoren erlauben den sequentiellen Zugriff auf Elemente, z. B. `Iterator<T>` in Java. - Abstrakte Datentypen (ADT) definieren Verhalten unabhängig von der konkreten Implementierung (z. B. Stack, Queue, Liste als Spezifikationen).

(3) Datenstrukturen. Verkettete Listen (Linked List). Vorteile beim Einfügen/Entfernen, Zugriff | Zeitkomplexität typischer Operationen: $\mathcal{O}(1)$ am Anfang, $\mathcal{O}(n)$ für den Zugriff.

Stacks und Queues. LIFO vs. FIFO; Implementierungen als Array oder Liste; typische Operationen: `push/pop` (Stack), `enqueue/dequeue` (Queue).

Bäume, binäre Suchbäume (BST) und Heaps. Eigenschaften, Such- und Insertion-Operationen im Durchschnitt $\mathcal{O}(\log n)$; Heaps als Grundlage von Prioritätswarteschlangen.

Graphen. Knoten und Kanten, gerichtete/ungerichtete Graphen; Repräsentationen als Adjazenzliste oder Adjazenmatrix.

(4) Algorithmen. Suchen und Sortieren von Feldern. Sortierverfahren, Zeitkomplexität und Unterschiede zwischen stabilen und instabilen Sortierverfahren. Beispiele der Größenordnung: $\mathcal{O}(n \log n)$ für effiziente Sorten wie Quicksort/Heapsort, $\mathcal{O}(n^2)$ für einfache Sortierverfahren wie Bubble-Sort.

Komplexität von Algorithmen. Beurteilung von Laufzeit- und Speicherbedarf: Worst-Case, Durchschnittsfall, Raumkomplexität.

Durchsuchen und Rekonfigurieren von Bäumen. Traversierungen: Inorder, Preorder, Postorder; rekursive vs. iterative Implementierung; Zeitkomplexität $\mathcal{O}(n)$ für vollständige Durchläufe.

Suchen nach Elementen und kürzesten Wegen in Graphen. Suchstrategien: DFS, BFS; kürzeste Wege in gewichteten Graphen z. B. Dijkstra; Komplexitäten je nach Repräsentation des Graphen.

(5) Boolesche Algebra, Schaltungsentwurf, Normalformen und Vereinfachungsverfahren. Grundlagen der Booleschen Algebra, Umformung von logischen Ausdrücken, Normalformen (DNF, CNF) und Vereinfachungsverfahren (z. B. Karnaugh-Maps, Quine–McCluskey). Anwendungen im Schaltungsentwurf und in der digitalen Logik.

(6) Beispielhafte Abbildungen und Notationen. - Generische Typen: `class Box<T>` oder `List<T>` in Java. - Iteratoren: `Iterator<T>` als Zugriffsmittel auf Elemente einer Sammlung. - Graphdarstellungen: Adjazenzliste vs. Adjazenzmatrix, je nach Anwendungsfall.

Hinweise zur Umsetzung in Java. - Typisierung und Generics korrekt verwenden, z. B. `ArrayList<T>` statt `ArrayList`. - Abstrakte Datentypen definieren Interfaces (z. B. `StackInterface<T>`)

und konkrete Klassen implementieren diese. - Komplexitätsabschätzungen vor jeder Optimierung prüfen (z. B. $\mathcal{O}(n \log n)$ vs. $\mathcal{O}(n^2)$).