Probeklausur

Einführung in die Informatik - Vertiefung

Universität:	Technische Universität Berlin
Kurs/Modul:	Einführung in die Informatik - Vertiefung
Bearbeitungszeit:	120 Minuten
Erstellungsdatum:	September 20, 2025

Bearbeitungszeit: 120 Minuten.

Aufgabe 1.

(a) Skizzieren Sie eine generische Klasse `Stack<T>` in Java-Notation mit den Methoden `push(T x)`, `T pop()`, `boolean isEmpty()`, `int size()`. Beschreiben Sie kurz, wie ein Iterator über den Stack implementiert werden könnte, so dass Elemente in LIFO-Reihenfolge durchlaufen werden.

(b) Definieren Sie eine abstrakte Schnittstelle `IList<T>` mit den Methoden `void add(T x)`, `T get(int i)`, `int size()`. Diskutieren Sie, welche Vorteile ein Iterator bei der Verwendung dieser ADT gegenüber direktem Indexzugriff bietet und welche Anforderungen an die Reihenfolge der Elemente sinnvoll sind.

(c) Skizzieren Sie eine einfache generische Basisklasse `AbstractList<T>` mit einer gemeinsamen Hilfsmethode zur Größenverwaltung. Begründen Sie, warum eine solche Struktur in einer Bibliothek sinnvoll ist und welche Typen davon profitieren (*z. B.* konkrete Implementierungen wie `ArrayList<T>` oder `LinkedList<T>` hinzufügen).

Aufgabe 2.

(a) Beschreiben Sie die grundlegende Struktur einer einfach verketteten Liste. Geben Sie die typischen Laufzeiten der folgenden Operationen an: Einfügen am Anfang, Entfernen am Anfang, Zugriff auf das i-te Element.

(b) Skizzieren Sie das Prinzip eines Stacks, der auf einer verketteten Liste basiert. Erklären Sie, wie Sie `push`, `pop` und `top` implementieren würden und wie sich die Laufzeiten verhalten.

(c) Erklären Sie die beiden Grundformen von Graphen-Speicherungen: Adjazenzenliste und Adjazenzmatrix. Diskutieren Sie Vor- und Nachteile beider Repräsentationen hinsichtlich Speicherbedarf und typischer Graph-Operationen.

(d) Geben Sie die rekursive Inorder-Traversierung für einen Binärbaum an und erläutern Sie kurz, wann eine Rekursion sinnvoll ist gegenüber einer iterativen Implementierung.

Aufgabe 3.

(a) Analysieren Sie die Laufzeit der Breitensuche (Breadth-First Search) in einem ungewichteten Graphen mit Knoten und Kanten. Geben Sie eine allgemeine obere Schranke in $\mathcal{O}(\cdot)$ an und erläutern Sie die Abhängigkeiten von und .

(b) Beschreiben Sie den Dijkstra-Algorithmus zur Bestimmung der kürzesten Pfade von einer Quelle in einem Graphen mit nicht-negativen Kantengewichten. Geben Sie die zentrale Idee in zwei bis drei Sätzen wieder und nennen Sie die benötigte Datenstruktur zur Optimierung.

(c) Gegeben sei der Baum $\mathcal{T} = (V, E)$ mit Wurzel r. Zeigen Sie, wie mittels Tiefensuche (DFS) die Ebenen und die Eltern-Beziehung zuverlässig bestimmt werden können. Formulieren Sie eine kurze iterative Variante als Gedankenkonstruktion.

(d) Vergleichen Sie die Komplexität von Tiefensuche (DFS) und Breitensuche (BFS) in einem Baum bezüglich Laufzeit und Speicherbedarf.

Aufgabe 4.

(a) Geben Sie eine Boolesche Ausdruck φ in konjunktiver Normalform (KNF) an, indem Sie die folgenden Terme gegebenen Variablen symbolschematisch umformen:

$$\varphi = (A \vee B) \wedge (\neg A \vee C) \vee (B \wedge \neg C).$$

Führen Sie eine explizite Umformung zu KNF durch und notieren Sie jeden Zwischenschritt.

(b) Zeigen Sie, wie sich eine einfache Schaltungslogik mit drei Eingängen A, B, C als Boolescher Ausdruck $F(A, B, C)$ darstellen lässt, wobei F gleich $(A \wedge B) \vee (\neg C)$ ist. Skizzieren Sie die Schaltungsentwurf-Überlegungen (ohne Zeichnung).

(c) Bestimmen Sie eine Minimierungs- bzw. Normalform-Variante von F mit zwei Variablen.

Aufgabe 5.

(a) Gegeben sei ein gerichteter, gewichteter Graph $G = (V, E)$ mit Knoten $V = \{v_0, \ldots, v_{n-1}\}$ und gewichteten Kanten. Beschreiben Sie einen Algorithmus, der den kürzesten Pfad von v_0 nach v_{n-1} mithilfe von Dijkstra findet. Geben Sie die Hauptschritte als Fließtext wieder.

(b) Nehmen Sie an, dass der Graph zusätzlich Zyklen enthält. Erläutern Sie, warum Dijkstra in diesem Fall noch korrekt funktioniert, sofern alle Kantengewichte nicht negativ sind.

(c) Diskutieren Sie, wie Generika in der Java-Programmierung dabei helfen, eine generische Graph-API zu entwerfen, die Kanten mit generischen Gewichtstypen unterstützt. Geben Sie grob an, welche Klassen/interfaces sinnvoll wären und welche Typen dabei verwendet werden könnten.