

Themenübersicht aller Klausuren und Lösungen

Vollständige, thematisch zusammengefasste Liste aus den **16 Aufgaben-PDFs und 16 zugehörigen Lösungs-PDFs** des Ordners. Wiederholte Inhalte werden gemeinsam aufgeführt. Stand: 11.10.2026.

Erfasst sind Programmieraufgaben, Wissens- und Multiple-Choice-Fragen sowie ergänzende Erklärungen der Lösungen. Ein Themename allein bedeutet nicht, dass dazu eine vollständige Implementierung verlangt wird. **„Ergänzungen aus den Lösungen“** kennzeichnet vertiefende Hinweise und Randfälle.

Belege lesen: „09: 3a, 3b“ bedeutet Klausur-ID 09, Aufgaben 3a und 3b; der Verweis umfasst jeweils Aufgabenblatt und Lösung. Die vollständigen Dateinamen stehen im Quellenverzeichnis.

A. Programmierung mit Java	2
A01 Java: Programmaufbau, Syntax und Ausführung	2
A02 Zahlensysteme und binäre Zahlendarstellungen	2
A03 Datentypen, Operatoren und sichere Auswertung	3
A04 Kontrollstrukturen, Konsoleneingabe und Ablaufspuren	3
A05 Zahlenalgorithmen, Häufigkeiten und Auswahl von Extremwerten	4
A06 Stringalgorithmen: Filtern, Wiederholen, Suchen und Löschen	4
A07 Arrays: Grundlagen, Änderungen an Ort und Stelle und Messdaten	5
A08 Matrizen, Bildtransformationen und zweidimensionale Muster	6
A09 Klassen, Konstruktoren, Kennungen und Zugriffsrechte	6
A10 Abstraktion, Interfaces, Vererbung und Polymorphie	7
A11 Referenztypen, Methodenauflösung und statische Typfehler	8
A12 Speicherabbilder, Referenzen und objektorientierte Ablaufanalyse	8
A13 UML und allgemeine Assoziationsverwaltung	9
A14 UML-Modelle: InternetRadio, Lager und MediaPlayer	9
A15 UML-Modelle: Garten, Regal und Gebäudeflächen	10
A16 UML-Modell: Putzplan, Fenster und Reinigungskosten	10
A17 UML-Modell: Veröffentlichungen und Themengebiete	11
A18 Polymorphe Anwendungsmodelle und Text-/Zahlenausgaben	11
B. Algorithmen und Datenstrukturen	12
B01 Algorithmusbegriff und Divide-and-Conquer	12
B02 Asymptotische Laufzeiten und Aufwand von Operationen	12
B03 Schleifenanalyse, Rekursion und Rekursionsspeicher	12
B04 Java-Implementierung, Referenzen und Vergleiche	13
B05 List, Set, Map und allgemeine Operationen verketteter Listen	13
B06 Prioritätslisten, Warteschlangen und Stapel	14
B07 Binärbäume, Suchbäume und Traversierungen	14
B08 AVL-Bäume und Balancebedingungen	15
B09 Binäre Heaps und Prüfung der Heapordnung	15
B10 B-Bäume: Suche, Ausgabe, Höhe und Ordnung	16
B11 Lineare, binäre und Interpolationssuche	16
B12 Hash-Tabellen und Kollisionsauflösung	16
B13 Sortierverfahren und Vergleich ihrer Laufzeiten	17
B14 Cocktail-Sort und Gnome-Sort schrittweise ausführen	17
B15 Graphdarstellung, Veränderungen und Zusammenhang	18
B16 Topologische Sortierung und gerichtete Kreisfreiheit	18
B17 Breitensuche mit Warteschlange	18
B18 Dijkstra und Rekonstruktion kürzester Wege	18
B19 Minimale Spannbäume, Kruskal und Union-Find	19
C. Quellenverzeichnis	20

A. Programmierung mit Java

A01 Java: Programmaufbau, Syntax und Ausführung

- Gültige Klassen-, Methoden- und Variablennamen; Groß-/Kleinschreibung, Schlüsselwörter und unzulässige Trennzeichen.
- Sprachliche Zulässigkeit von üblichen Namenskonventionen unterscheiden.
- Methodendeklaration, Rückgabetyt und return-Anweisung; Hilfsklassen benötigen nicht zwingend eine main-Methode.
- Syntaxfehler von Semantikfehlern unterscheiden, etwa ungültige Rückgabetypkombination gegenüber falscher Rechenoperation.
- Java-Quelltext, Compiler, Bytecode, Interpreter, JVM und Maschinencode unterscheiden; Programme können mehrere Quelldateien umfassen.
- Plattformunabhängiger Bytecode und plattformabhängige Ausführungsumgebung; Aufwand der Interpretation im vereinfachten Modell.

Ergänzungen aus den Lösungen

- JIT-Kompilierung häufig ausgeführten Bytecodes präzisiert das Ausführungsmodell; pauschale Aussagen über grundsätzlich langsamere Java-Ausführung einordnen.
- Öffentliche Typen in gleichnamigen Java-Dateien organisieren.
- Quellenfehler wie unpassende Zeilennummern und inkonsistente Schreibweisen von Methodennamen erkennen und ausdrücklich von der beabsichtigten Fassung unterscheiden.

Belege: 09: 1a · 10: 1a · 15: 4 · 16: 4 · 17: 4.4, 5, 4 · 20: 1a, 1c, 3a

A02 Zahlensysteme und binäre Zahlendarstellungen

- Dezimal-, Binär- und Hexadezimalzahlen anhand ihrer Stellengewichte umrechnen; Hexadezimalziffern und Vierergruppen von Bits zuordnen.
- Ziffern durch wiederholte Division und Restbildung gewinnen; Stellenzahl und führende Nullen unterscheiden.
- Hexadezimale Addition mit Überträgen durchführen und dieselbe Summe in verschiedenen Zahlensystemen darstellen.
- Negative ganze Zahlen im Zweierkomplement darstellen: Betrag binär notieren, Bits invertieren und eins addieren.
- Zweierkomplement über das negative höchstwertige Stellengewicht oder über die Differenz zur passenden Zweierpotenz rückprüfen.
- Bitbreite und Wertebereich vorzeichenbehafteter Ganzzahlen; achtbittige Darstellungen und Java-int mit 32 Bit.
- Hexadezimale Java-Literale mit 0x und ihr unterschiedlicher Wert als int beziehungsweise als long mit L-Suffix.
- Binäre Fließkommazahlen normalisieren; Vorzeichen, Exponent, Signifikand, Exponenten-Bias und implizite führende Eins unterscheiden.
- Aufteilung und Genauigkeit der Java-Formate float und double; exakt darstellbare Binärbrüche gegenüber periodischen Darstellungen von Dezimalbrüchen und notwendiger Rundung.

Ergänzungen aus den Lösungen

- Zweierkomplement hat nur eine Nulldarstellung; beim int-Überlauf bleiben die niederwertigen Bits erhalten.

Belege: 09: 1a, 3a, 3b · 10: 1b, 1a, 3a · 11: 1b, 1c, 2, 4b · 12: 1c, 3b, 4a, 4b · 15: 1 · 16: 1 · 17: 1

A03 Datentypen, Operatoren und sichere Auswertung

- Primitive Typen int, long, float, double, boolean und char in Variablen, Parametern, Feldern und Rückgaben verwenden.
- Erweiternde und verengende numerische Typumwandlungen, explizite Casts und Typanpassung bei gemischten Vergleichen; boolean ist nicht numerisch konvertierbar.
- Ganzzahlige Division von Gleitkommadivision unterscheiden; den Cast vor einer Division beziehungsweise Multiplikation platzieren.
- Zuweisung, Gleichheit, Ungleichheit und Größenvergleich unterscheiden; syntaktisch nicht vorhandene Operatoren erkennen.
- Arithmetik und Stringverkettung mit +: Linksassoziativität, Klammerung und ein führender leerer String beeinflussen die Auswertung.
- Prä-/Postinkrement und Prä-/Postdekrement; alte und neue Werte sowie Seiteneffekte in Bedingungen und Ausgaben.
- Zusammengesetzte Zuweisungen wie +=, -= und /=; die rechte Seite wird vor der abschließenden Aktualisierung ausgewertet.
- Boolesche Operatoren &&, ||, &, | und ^; kurzschließende Auswertung gegenüber Auswertung beider Operanden und exklusivem Oder.
- Statische Hilfsmethoden und der bedingte Operator ?: in kompakten Fallunterscheidungen.

Ergänzungen aus den Lösungen

- Darstellbarkeit von Summen, Produkten, Kosten und Dimensionen beachten; long-Akkumulatoren und Casts vor einer Multiplikation verhindern bestimmte int-Überläufe.
- Überlauf sichere Grenzen für Teiler- und Potenzprüfungen; long-Schleifenzähler bis zur oberen int-Grenze.
- Ein Häufigkeitsarray der Länge Maximum plus eins kann an der maximalen int-Grenze nicht angelegt werden.
- Aussagen über vollständige Operandenauswertung setzen normale Ausführung voraus; eine Ausnahme kann sie vorzeitig beenden.

Belege: 09: 3a, 3b, 5a, 7, 8, 4a · 10: 1a, 1b, 3a, 2, 7, 8 · 11: 2, 4b, 8, 5, 1a · 12: 3b, 4a, 4b, 1b, 8, 5, 7, 1a · 15: 4, 2i-2ii, 3ii · 16: 3ii, 4.2 · 17: 3ii, 4.3 · 20: 1d, 1e, 5b, 2b-2c, 3a-3d, 4b, 1f

A04 Kontrollstrukturen, Konsoleneingabe und Ablaufspuren

- Scanner, System.in, notwendige Importe und main; Ganzzahlen beziehungsweise Strings einlesen und mit print/println ausgeben.
- for-, while- und do-while-Schleifen, verschachtelte Durchläufe, if/else, Zustandsflags, break und frühe Rückgaben.
- Initialisierung außerhalb einer Schleife von wiederholter Verarbeitung trennen; vorhandene eigene Hilfsmethoden wiederverwenden.
- Auswahlmenü nach einmaliger Eingabe: switch/case/default, fallabschließendes break und Abbruchauswahl ohne if-Abfragen.
- Fußball-Punktetabelle aus drei Spielpaarungen: Ergebnisbewertung, Punkttakkumulation und vollständige Neueingabe bei Punktgleichheit.
- Sentinelgesteuerte Zahlenfolgen: Abschlusswert vor Verarbeitung erkennen und aus Summe, Anzahl oder Auswahl ausschließen.
- Summe und arithmetischen Mittelwert einer Eingabefolge bestimmen; Eingaben nach einer Vorzeichenbedingung beenden.
- Endwerte von Schleifenvariablen und Zahl der Durchläufe bestimmen; Halbierung, Schrittweiten und letzte Aktualisierung berücksichtigen.
- Spurentabellen mit Werten vor und nach Bedingungstests; Seiteneffekte bleiben auch beim letzten falschen Test wirksam.

Ergänzungen aus den Lösungen

- Scanner.next() für Wörter und nextLine() für vollständige Zeilen einschließlich Leerzeichen unterscheiden.
- Leere Eingabefolge beim Mittelwert gesondert behandeln: eine Summe ist möglich, ein Mittelwert ohne Messwerte nicht definiert.
- Die gemeinsam genutzte Standardeingabe durch Schließen eines Scanners nicht unbeabsichtigt unbrauchbar machen.

Belege: 09: 2 · 10: 2 · 11: 2 · 12: 2, 1b · 15: 2i-2ii, 3ii-3iii · 16: 2i-2iv, 3i-3ii · 17: 2i-2iv, 3i-3ii · 20: 2b-2c, 2a-2c, 3a-3d

A05 Zahlenalgorithmen, Häufigkeiten und Auswahl von Extremwerten

- Positive Teiler mit Modulo erkennen und zählen; Primzahlen über ihre positiven Teiler charakterisieren.
- Primfaktorzerlegung durch aufsteigende Kandidaten, wiederholtes Herausdividieren und Behandlung eines verbleibenden Primrests; Faktorfolge als String aufbauen.
- Dezimalziffernzahl durch Division ermitteln, Ziffern mit Restbildung abtrennen und in ein passend großes Array schreiben.
- Quersumme durch Zerlegung in Dezimalziffern berechnen.
- Häufigsten Wert im begrenzten Ziffernbereich und häufigste Dezimalziffer bestimmen; bei Gleichstand die kleinste wählen.
- Globales Maximum und wertindizierte Häufigkeitsverteilung eines zweidimensionalen Feldes bestimmen; Nichtnegativität als Voraussetzung für bestimmte Initialwerte.
- Maximalen Exponenten eines Dreierfaktors durch wiederholte Division bestimmen.
- Unter den Dreierexponenten eines Zahlenintervalls nur gerade Werte für die Maximumsuche berücksichtigen.
- Ein Maximum linear lokalisieren und genau einen maximalen Arrayeintrag nur dann ersetzen, wenn der neue Wert echt kleiner ist.
- Die zwei beziehungsweise k kleinsten negativen Eingaben in einem begrenzten Auswahlarray erhalten; Duplikate als getrennte Eingaben berücksichtigen.
- Summe beziehungsweise Produkt der ausgewählten Werte; weniger als k Werte und eine vollständig leere Eingabe berücksichtigen.

Ergänzungen aus den Lösungen

- Die Zahl eins besitzt keine Primfaktoren; das leere Produkt und dessen gewählte Textdarstellung als Randfall unterscheiden.
- Eine Dezimaldarstellung der Null umfasst eine Ziffer; innere Nullen werden ebenfalls gezählt, und für Häufigkeiten ist die Reihenfolge der Ziffern unerheblich.
- Ein häufigster tatsächlich vorkommender Wert ist für ein leeres Feld nicht definiert; dies von einer Konvention für lauter leere Häufigkeitszähler unterscheiden.
- Dreierfaktoren auch für negative Eingaben; bei Null fehlt ein maximaler Exponent. Eine alternative Maximumsuche verwendet die zu geraden Exponenten gehörenden Neunerpotenzen.
- Belegungszähler und neutrale Anfangswerte verhindern, dass freie Auswahlplätze in ein Produkt eingehen; Maximum-/Auswahlvarianten und leere beziehungsweise null-Arrays bei einer ergänzenden Schutzbehandlung.

Belege: 09: 3a, 3b · 10: 3a, 3c, 3b · 11: 3b, 3a · 12: 3a, 3b · 15: 2i · 16: 3i, 3i-3ii · 17: 2iv · 20: 2a-2c, 2a, 2b-2c

A06 Stringalgorithmen: Filtern, Wiederholen, Suchen und Löschen

- Strings zeichenweise mit length() und charAt() verarbeiten; Ergebnis durch Verkettung aufbauen und erlaubte Bibliotheksmethoden beachten.
- Strings nach gerader und ungerader Länge gruppieren und die Eingabereihenfolge innerhalb der Gruppen erhalten.

- Das erste Vorkommen eines Zeichens mit Zustandsflag oder alle Vorkommen durch Filtern entfernen; ohne Treffer unverändert lassen und sonst die Reihenfolge der übrigen Zeichen erhalten.
- Wörter wiederholt aneinanderfügen und Präfixe vorgegebener Länge bilden; Anzahl und Indexgrenzen beachten.
- Doppelte Wortwiederholungen über gleiche Hälften sowie mehrfache Wiederholungen über mögliche Grundwortlängen untersuchen.
- Teilbarkeit der Wortlänge für Wiederholungskandidaten nutzen; Präfix und Wiederholung kombinieren und Wortinhalte vergleichen.
- Prüfen, ob ein Suchwort ab einem bestimmten Index vollständig übereinstimmt; erstes Vorkommen durch aufsteigende Indexsuche finden.
- Ein Teilwortvorkommen durch Überspringen seines Indexintervalls entfernen; vereinbarte Rückgabewerte bei erfolgloser Suche beachten.
- Mehrere Löschwörter wiederholt anwenden; neu entstandene Treffer über Löschgrenzen erneut suchen und nach einem erfolglosen Gesamtdurchlauf aufhören.
- Verschiedene Löschreihenfolgen können unterschiedliche zulässige Endwörter erzeugen.

Ergänzungen aus den Lösungen

- String-Inhaltsvergleich mit equals von Referenzidentität mit == unterscheiden; null-Namen vor einem Methodenaufruf gesondert behandeln.
- Leeres Wort, leeres Präfix und nullfache Wiederholung; formale Leerwortkonvention gegenüber einer Forderung nach nichtleerem Grundwort.
- Ungültige Wiederholungszahlen und Präfixlängen sowie das leere Suchwort an gültigen Grenzpositionen ausdrücklich behandeln.
- Terminierung wiederholter Löschung durch strikt kürzere Wörter begründen; das leere Löschwort kann auf diese Weise nicht vollständig beseitigt werden.

Belege: 12: 2, 7 · 15: 2i-2ii, 2i · 16: 2i-2iv, 2i-2ii, 2iii-2iv, 3i · 17: 2i-2iv, 2i-2ii, 2iii-2iv, 2iv · 20: 2a-2c

A07 Arrays: Grundlagen, Änderungen an Ort und Stelle und Messdaten

- Nullbasierte Indizes, Längenattribut, feste Länge nach der Erzeugung und fehlende automatische Sortierung; Inkrementieren eines Werts vergrößert kein Array.
- Mehrdimensionale Arrays als Arrays von Arrays, auch mit mehr als zwei Dimensionen; Zeilenreferenzen und individuelle Zeilenlängen.
- Arrayelemente mit Schleifen traversieren und aufsummieren; Rückgabe eines berechneten Werts von Konsolenausgabe unterscheiden.
- Eindimensionales Array ohne neues Array umkehren: spiegelbildliche Positionen mit Hilfsvariable tauschen.
- Präfixsummen im vorhandenen Array von links nach rechts aufbauen und bereits aktualisierte Vorgängerwerte nutzen.
- Temperaturmesswerte nach Tagen in einem zweidimensionalen Array verwalten; zunächst leere Tageszeilen erzeugen.
- Einen Messwert anhängen, indem eine größere Tageszeile angelegt, befüllt und als neue Zeilenreferenz eingesetzt wird.
- Tagesindex und äußere Arrayform prüfen; Tagesmittelwerte aus Summe und Anzahl der Messungen bestimmen.
- Vorbedingungen für null, positive beziehungsweise nichtnegative Größen, rechteckige Felder und zulässige Wertebereiche; Assertions verwenden.

Ergänzungen aus den Lösungen

- Leere Zeile von einer vorhandenen Messung mit Zahlenwert 0 unterscheiden; null-Zeilen nur aufgrund einer ausdrücklich dokumentierten Konvention als leer behandeln.

- Mittelwert ohne Messwerte ist nicht definiert; eine Ersatzrückgabe muss kenntlich gemacht werden. Unterschiedlich lange und leere Zeilen bei Maximum und Häufigkeiten mitbehandeln.
- Leere oder einelementige Arrays und mittlere Elemente bei ungerader Länge erfordern bei Umkehrung beziehungsweise Präfixsummen keine Sonderänderung.
- Vor dem nächsten Präfixsummenschnitt steht im Vorgänger bereits die Summe des ursprünglichen Präfixes; diese Schleifeninvariante zur Begründung nutzen.
- Zurückgegebene Matrixzeile und neu erzeugte Spalte unterscheiden; Änderungen über die Zeilenreferenz beeinflussen die ursprüngliche Matrix.

Belege: 09: 4a, 4b, 3a, 3b · 10: 3a, 3c, 4a, 4b, 3b · 11: 4a, 4b, 3a, 3b · 12: 4a, 4b, 3a, 3b · 15: 3i-3iii, 3i-3ii, 3iii · 16: 3i-3ii, 4.1, 4.3 · 17: 3i-3ii, 4.1-4.2 · 20: 1d, 3a-3d, 3b-3d, 4c-4d

A08 Matrizen, Bildtransformationen und zweidimensionale Muster

- Vektoren und Matrizen als numerische Arrays darstellen und mit passenden Zeilen-/Spaltendimensionen traversieren.
- Skalarprodukt als Summe komponentenweiser Produkte; Matrixzeilen und Matrixspalten für die Weiterverarbeitung gewinnen.
- Matrixprodukt über Zeilen-Spalten-Skalarprodukte berechnen; innere Dimensionen müssen übereinstimmen, äußere bestimmen das Ergebnisformat.
- Transponieren durch Vertauschen der Indizes und Dimensionen; das ursprüngliche Array erhalten.
- Symmetrie mit der Transponierten vergleichen: quadratische Form, Spiegelpaare und Diagonale; unnötige Doppelvergleiche vermeiden.
- Graustufenbilder als Byte-Arrays; horizontale Symmetrie jeder Zeile durch spiegelbildliche Pixelvergleiche prüfen.
- Bild um einen rechten Winkel nach links drehen: Dimensionen vertauschen und eine Koordinate umkehren.
- Treppmuster aus einem je Zeile verschobenen Sternbereich in einem rechteckigen char-Array aufbauen.
- Gittermuster über jede zweite Zeile oder Spalte erzeugen; Parität der nullbasierten Indizes und logisches Oder kombinieren.
- Schachbrettartige Null-/Eins-Blöcke mit fester oder variabler Blockhöhe und -breite; Blockkoordinaten durch Ganzzahldivision und Modulo bestimmen.

Ergänzungen aus den Lösungen

- Java-byte ist vorzeichenbehaftet; Graustufen oberhalb seines positiven Bereichs durch dieselben Bitmuster speichern und mit & 0xff vorzeichenlos interpretieren.
- Ein leeres äußeres Array enthält keine ablesbare Spaltenzahl; Konventionen für leere Transposition und Symmetrie dokumentieren.
- char-Arrayzellen enthalten anfangs das Nullzeichen, nicht ein Leerzeichen. Feld- und Blockgrößen prüfen, abgeschnittene Randblöcke zulassen und Blockindexüberlauf durch frühere Paritätsreduktion vermeiden.
- Ein fehlerhaftes Zahlenbeispiel zum Skalarprodukt von der korrekten Definition unterscheiden.

Belege: 09: 4a, 4b · 10: 4a, 4b, 3a · 11: 4b, 4a · 12: 4b, 4a · 15: 4 · 16: 4.1, 4.3, 3ii, 4.2 · 17: 4.4, 5, 3i-3ii, 4.1-4.2, 3ii, 4.3 · 20: 1c, 3a, 3a-3d, 3b-3d, 4c-4d, 2b-2c, 4b

A09 Klassen, Konstruktoren, Kennungen und Zugriffsrechte

- Klassen und Objekte, Instanzattribute und gemeinsame Klassenattribute unterscheiden; Objekte durch new und passende Konstruktoren erzeugen.
- this zur Unterscheidung eines Objektfelds von gleichnamigen Parametern; eigene und geerbte Felder getrennt initialisieren.
- Konstruktoren überladen, Argumente durch mehrstufige Hierarchien weiterreichen und super(...) gezielt einsetzen.

- Reihenfolge von Oberklassenkonstruktoren und Feldinitialisierungen; implizite parameterlose Konstruktoraufrufe sowie vertauschte super-Argumente nachvollziehen.
- Fortlaufende Objektkennungen durch gemeinsamen static-Zähler vergeben und die individuelle Nummer getrennt speichern.
- Kennung über private final und fehlenden öffentlichen Setter unveränderlich halten; Postinkrement bei der Vergabe.
- public, protected, private und Paketsichtbarkeit; sichtbare Member innerhalb eines Packages und in Unterklassen außerhalb des Packages.
- Private Zustände durch geeignete Getter und Setter kapseln; ein Paketzugriff wird durch Weglassen eines Zugriffsmodifizierers ausgedrückt.
- Objekt- und Klassenfelder erhalten Standardwerte, lokale Variablen nicht; geerbte static-Felder sind gemeinsame Speicherstellen.

Ergänzungen aus den Lösungen

- protected-Zugriff aus fremden Packages hängt auch vom qualifizierenden Empfänger ab; ein öffentlicher Member allein garantiert noch keinen Zugriff, daher auch die Zugänglichkeit der verwendeten Typen prüfen.
- Eine package-private-Hilfsmethode kann die interne Pflege einer bidirektionalen Assoziation ermöglichen.
- Ein einfacher int-ID-Zähler besitzt einen endlichen Wertebereich; die gezeigte Kennungsvergabe setzt sequenzielle statt nebenläufiger Erzeugung voraus.

Belege: 09: 7, 5a, 5b, 6b, 8, 1a, 5, 6 · 10: 1a, 7, 5a, 5b, 6e, 6f, 8 · 11: 8, 5, 7, 1a · 12: 8, 5, 7, 1a · 15: 4 · 16: 4.2, 3ii, 5 · 17: 4.3, 3ii, 5 · 20: 1b, 4b, 4c, 2b-2c, 3a-3d, 5b, 5a-5b

A10 Abstraktion, Interfaces, Vererbung und Polymorphie

- Abstrakte Klassen und abstrakte Methoden; eine Klasse mit abstrakter Methode muss abstrakt sein, eine abstrakte Klasse benötigt nicht zwingend abstrakte Methoden.
- Abstrakte Klassen dürfen Konstruktoren und konkrete Methoden enthalten; unmittelbar instanziiert werden nur konkrete Klassen.
- Interfaces definieren und implementieren; Interfaces haben keine Konstruktoren und sind nicht unmittelbar instanziiierbar.
- Einzelne direkte Klassenoberklasse gegenüber mehreren implementierten Interfaces; Mehrfachvererbung durch mehrere Oberinterfaces.
- Konkrete Klassen müssen die erforderlichen abstrakten Operationen implementieren; Implementierungen können geerbt sein.
- Überschreiben und dynamische Bindung anhand des tatsächlichen Objekttyps, auch über Oberklassen- oder Interface-Referenzen.
- super.methode(...) wählt die Oberklassenimplementierung; ein weiterer unqualifizierter Aufruf darin kann erneut dynamisch gebunden werden.
- Polymorphe Arrays enthalten verschiedene konkrete Untertypen, darunter Pflanzen sowie Bücher/CDs, und nutzen gemeinsame Operationen für Berechnungen.

Ergänzungen aus den Lösungen

- Private Interface-Hilfsmethoden mit Rumpf sind versionsabhängig; private abstrakte Interface-Methoden sind unzulässig. Geeignete default- und geerbte Methoden können Implementierungspflichten erfüllen.
- Gewöhnliche abstrakte Interface-Methoden sind implizit öffentlich; ihre Implementierung muss öffentlich sein. Statische und private Interface-Methoden müssen nicht in jeder implementierenden Klasse nachgebildet werden.

- Ein gleichnamiger bedingter Selbstaufruf in der Basismethode kann auf einem Unterklassenobjekt die überschreibende Methode erreichen; dies von einer tatsächlich fortgesetzten Rekursion unterscheiden.

Belege: 09: 1a, 6, 1b, 7, 8 · 10: 5a, 6e, 6g, 1a, 5c, 6d, 7, 8 · 11: 1a, 5, 6, 7 · 12: 5, 7, 6 · 15: 4 · 16: 4.2-4.3, 4, 5 · 17: 4, 5 · 20: 5a-5b

A11 Referenztypen, Methodenauflösung und statische Typfehler

- Methodenüberladung anhand unterschiedlicher Parameterlisten, Parametertypen und deren Reihenfolge; nur verschiedene Rückgabetypen genügen nicht.
- Verfügbare Methoden und Überladungen am deklarierten Empfängertyp prüfen; dynamischer Typ bestimmt anschließend die überschreibende Implementierung.
- Gleichnamige Felder in Ober- und Unterklassen bleiben getrennt; Feldzugriff ist statisch an den Referenztyp gebunden.
- Getter einer Oberklasse lesen deren private Felder, auch wenn die Unterklasse ein gleichnamiges Feld besitzt.
- Eine Referenzvariable darf einen abstrakten Klassen- oder Interface-Typ besitzen; ein virtueller Aufruf einer abstrakt deklarierten Methode kann zulässig sein.
- Zuweisung einer Unterklasse an eine Oberklassenreferenz von der umgekehrten Richtung und von Zuweisungen nicht verwandter Klassen unterscheiden.
- Interface-Zuweisung setzt eine direkte oder geerbte Implementierung voraus; unbekannte Methoden am statischen Typ und unmittelbare Interface-Instanziierung als Übersetzungsfehler.
- Laufzeittyp mit instanceof prüfen und anschließend passend casten; konkrete Reinigungsarten beziehungsweise benötigte Unterklassenoperationen unterscheiden.

Ergänzungen aus den Lösungen

- Eine nur im Untertyp vorhandene Überladung wird nicht durch das tatsächliche Unterklassenobjekt über eine Basistypreferenz verfügbar; gezielten Empfängercast als minimale Korrektur prüfen.
- Wörtlich nicht kompilierbare Vorlage von einer korrigierten Fassung trennen. Ein Cast an einem Aufruf und eine Änderung des Variablentyps können wegen unterschiedlicher Feldbindung verschiedene Folgen haben.

Belege: 09: 7, 8 · 10: 8, 7 · 11: 8, 7, 6 · 12: 8, 6 · 16: 5 · 17: 4.4, 5 · 20: 1c, 3a, 1f

A12 Speicherabbilder, Referenzen und objektorientierte Ablaufanalyse

- Statischen Speicher, Objektheap und aktive Aufrufrahmen getrennt darstellen; eigene und geerbte Objektfelder vollständig erfassen.
- Referenzen in main, Methodenparameter, this und String[] args mit den jeweiligen Objekten verbinden.
- Vorgegebene UML-Objektdiagramme durch Erzeugungsreihenfolge und Referenzzuweisungen herstellen; null als fehlender Verweis.
- Empfängerobjekt von einem übergebenen Vergleichsobjekt unterscheiden; Getter und direkte Feldzugriffe bei booleschen Zustandsvergleichen nachverfolgen.
- Zustand vor return oder vor Verlassen einer Methode rekonstruieren; verschachtelte Ober-/Unterklassenaufrufe mit getrennten lokalen Variablen.
- Konstruktorfolgen, statische Zähler, Feldinitialisierungen und dynamische Methodenaufrufe für die spätere Konsolenausgabe gemeinsam verfolgen.
- Überladung, Feldverdeckung, super-Aufrufe und Inkrementausdrücke in mehrstufigen Hierarchien in eine vollständige Ausgabespur einbeziehen.

Ergänzungen aus den Lösungen

- Noch laufende rechte Seite einer Zuweisung: eine bisher uninitialisierte lokale Zielvariable ist noch nicht gesetzt, eine zuvor gesetzte behält ihren alten Wert.

- Bereits zurückgekehrte Getter oder Hilfsmethoden haben am betrachteten Zeitpunkt keinen aktiven Aufrufrahmen mehr; Akkumulatorzustände während Unteraufrufen gesondert erfassen.
- Mehrere Objekte können dasselbe String-Literal referenzieren; geteilte Referenz, eigenständiges Objekt und Kopie unterscheiden. Ein in einem Diagrammausschnitt ausgelassenes Objekt kann weiterhin erreichbar sein.
- Primitive Parameter werden als Wertkopien übergeben: Änderung eines lokalen Geschwindigkeitsparameters verändert kein gleichnamiges Objektfeld.

Belege: 09: 5a, 5b, 7, 8 · 10: 5b, 7, 8 · 11: 8 · 12: 8 · 16: 4.1, 4.3, 5 · 17: 4.1-4.2, 5 · 20: 3b-3d, 4c-4d, 5b

A13 UML und allgemeine Assoziationsverwaltung

- Klassendiagramme mit Typen, Attributen, Parametern, Rückgabetypen und Sichtbarkeitssymbolen in Java übertragen.
- Abstrakte beziehungsweise kursiv gekennzeichnete Klassen und Operationen, Interface-Realisierungen und Vererbungsbeziehungen unterscheiden.
- Unterstrichene Klassenattribute und ~ für Paketsichtbarkeit erkennen; Assoziationsrichtung und Multiplizitäten lesen.
- Referenzarrays mit fester Kapazität und Belegungszähler; nur benötigte Verwaltungsattribute und vorgegebene beziehungsweise ausdrücklich erlaubte Methoden ergänzen.
- Untergrenzen durch passende Konstruktorbelegung und Schutz des letzten Pflichtobjekts, Obergrenzen vor dem Einfügen sichern.
- Einfügen, Entfernen und Getter; bei lückenlosem Bestand Referenzen verschieben und den letzten Platz leeren.
- Doppelte Assoziationen nach Objektidentität vermeiden und null-Referenzen vor Änderungen zurückweisen.
- Rückgabekonventionen wie boolean, Einfügeindex, -1 oder null sowie Bibliotheksbeschränkungen bei eigenen Implementierungen einhalten.

Ergänzungen aus den Lösungen

- Defensive Kopien von Verwaltungsarrays verhindern fremde Änderungen der internen Referenzsammlung; Konstruktorargumente bei Bedarf in eigene Arrays übernehmen.
- Modellannahmen zu gültigen Konstruktorargumenten, nichtnegativen Größen und exaktem Ausschöpfen einer Kapazität ausdrücklich halten.

Belege: 09: 6a, 6b, 6c, 6 · 10: 6a, 6b, 6c, 6d, 6e, 6f, 6g · 11: 7 · 12: 7 · 15: 4 · 16: 4.1, 4.3, 4.1-4.3 · 17: 4.1-4.2, 4.1-4.4, 4.1-4.3 · 20: 3b-3d, 4c-4d, 4a-4d

A14 UML-Modelle: InternetRadio, Lager und MediaPlayer

- InternetRadio mit Geräteoberklasse und IPGerät-Interface; IP-Adresse und Senderinformationen über passende Methoden bereitstellen.
- Begrenzte Favoritenliste: Sender erzeugen, hinzufügen und zählen; ersten Eintrag lückenlos löschen und indexbasierten Zugriff prüfen.
- Lager mit Behältern: Aufnahmegrenze, Anzahl und Gesamtvolumen aus den dynamisch passenden Behälteroperationen bestimmen.
- Abstrakten Behälter durch Karton und Dose spezialisieren; Aufschrift und Abmessungen initialisieren und Kartenvolumen bestimmen.
- MediaPlayer mit abstraktem Medium sowie Audio/Video: Anzahl und freies Speicherbudget verwalten.
- Medium nur bei freiem Platz und genügend Speicher aufnehmen; beim Entfernen Budget zurückbuchen und die entfernte Referenz zurückgeben.
- Unter gleichen Namen das Medium mit kleinstem Index finden; im MediaPlayer erlaubte Arraylücken erneut nutzen.

Ergänzungen aus den Lösungen

- Einseitige Lagerzuordnung setzt in der gezeigten Schnittstelle voraus, dass ein Behälter noch keinem anderen Lager zugeordnet ist.
- Für korrektes Zurückbuchen wird ein während der Speicherung unveränderter, nichtnegativer Medienverbrauch vorausgesetzt.
- Lückenloser Favoritenbestand und lückenbehafteter Medienbestand verlangen unterschiedliche Verwaltung; null-Einträge und null-Namen gesondert behandeln.

Belege: 09: 6a, 6b, 6c, 6 · 10: 6a, 6b, 6c, 6d, 6e, 6f, 6g · 11: 7 · 12: 7

A15 UML-Modelle: Garten, Regal und Gebäudeflächen

- Garten und Pflanze-Interface mit konkretem Busch; Pflanzenassoziation innerhalb der vorgegebenen Mindest-/Höchstzahl verwalten.
- Gesamten Pflanzenflächenbedarf mit der rechteckigen Gartenfläche vergleichen; vollständige Ausnutzung ist erlaubt.
- Regal mit vorgeschriebener Mindestzahl und Obergrenze von Regalbrettern; jedes Brett verwaltet begrenzt viele Medien.
- Buch und CD über ein gemeinsames Medium-Interface modellieren; unterschiedliche Angaben zu Breite und Gewicht verwenden.
- Medienbreite und Gewicht aufsummieren und zugleich gegen Brettbreite und Tragkraft prüfen; erstes aufnahmefähiges Brett suchen.
- Gebäude und abstrakter Raum mit Wohnraum/Garage: Räume nur bei freiem Platz und ausreichender Gesamtfläche aufnehmen.
- Rechteckige Raumflächen, gesamte Raumfläche und ausschließlich die Wohnraumflächen berechnen; im UML geforderte numerische Typen beibehalten.

Ergänzungen aus den Lösungen

- Gültige, nichtnegative Abmessungen und Flächenkapazitäten voraussetzen; Kapazitätsgleichheit von Überschreitung unterscheiden.

Belege: 09: 6 · 10: 6c, 6g · 11: 7 · 12: 7 · 15: 4 · 16: 4.2-4.3, 4.1-4.3

A16 UML-Modell: Putzplan, Fenster und Reinigungskosten

- Gebäude mit genau einem Putzplan und begrenzter, nichtleerer Fenstersammlung modellieren.
- Reinigungseinträge im Putzplan bis zu seiner Obergrenze verwalten und über Indizes zugänglich machen.
- Abstrakten Reinigungseintrag mit Name, Firma und Kostenoperation durch Fenster- und Bodenreinigung konkretisieren.
- Fensterreinigungskosten aus Flächen und Kostensatz summieren; von einem konstanten Bodenreinigungspreis unterscheiden.
- Für jedes Gebäudefenster prüfen, ob es in mindestens einer Fensterreinigung enthalten ist.
- Nur passende Reinigungsarten auf ihre Fensterzuordnungen untersuchen und dabei Objektidentität berücksichtigen.

Ergänzungen aus den Lösungen

- Verschiedene Fenster bleiben bei gleicher Fläche verschiedene Objekte; Gleichheit von Eigenschaften ersetzt keine Identitätsprüfung.
- Inkonsistente Groß-/Kleinschreibung der vorgegebenen Prüfmethode als Quellenfehler kenntlich machen; zusätzliche Annahmen nur aus beschriebenen Vorbedingungen ableiten.

Belege: 17: 4.4, 5, 4.1-4.4 · 20: 1c, 3a, 4d

A17 UML-Modell: Veröffentlichungen und Themengebiete

- Nutzerkonten, Autoren, Gutachter und Herausgeber sowie die Vererbungskette Editor-Reviewer-UserAcct modellieren.
- Ober- und Untertypzugehörigkeit; Nutzernamen, E-Mail-Adresse, Getter/Setter und vorgegebene leere Methodenrumpfe.
- Assoziationen zwischen Author und Paper mit unterschiedlichen Mindestzahlen; notwendige Konstruktorargumente daraus ableiten.
- Unveränderliche Nutzererkennung mit gemeinsamem statischem Zähler, auch für Instanzen der Unterklassen.
- Bidirektionale Editor-Paper-Assoziation: begrenzte Papersammlung und höchstens ein Herausgeber je Paper; beide Richtungen gemeinsam aktualisieren.
- Doppelzuordnung an verschiedene Herausgeber verhindern, Herausgeberwechsel über Entfernen/Hinzufügen und geerbte Rollen getrennt verwalten.
- SubjectArea-Selbstassoziation mit begrenzten Unterthemen und Gettern; Selbstassoziation von Vererbung unterscheiden.

Ergänzungen aus den Lösungen

- Aus der Selbstassoziation folgen ohne zusätzliche Vorgabe weder Zyklusfreiheit noch eine begrenzte Zahl von Oberthemen.

Belege: 17: 4.1-4.4 · 20: 1b, 4c, 4a-4c, 4b, 5b, 4d

A18 Polymorphe Anwendungsmodelle und Text-/Zahlenausgaben

- toString als Textdarstellung bei println und Stringverkettung; referenzierte Person in die Fahrzeugausgabe einbeziehen.
- Tier/Freund-Modell mit Ziege und Tiger: gemeinsame Textdarstellung und dynamisch bestimmter Laut.
- Person mit Anwalt, Politiker und Manager: individuelle beziehungsweise konstante Gehälter, geschlechtsabhängige Berufsbezeichnung und nummernabhängige Namen.
- Vorgegebenes Person.toString unverändert lassen und das Ergebnis durch überschreibbare Beruf-/Gehaltsmethoden bestimmen; heterogenes Person-Array.
- Fahrzeug mit EBus, Pkw und Lkw: unterschiedliche Personenzahl, Treibstoffart, Verbrauch und Kilometer pro Liter bei begrenzter Attributzahl.
- Erstes EBus-Objekt dauerhaft kennzeichnen und Lkw nummerieren; gemeinsame Textdarstellung durch super.toString erweitern.
- Wurfmodell: Dauer begrenzen, mittlere Geschwindigkeit und Weite berechnen; dynamisch überschriebene Berechnung mit Widerstandsfaktor und double-Konsolenausgabe.

Ergänzungen aus den Lösungen

- Gleitkomma-Division durch einen Nullverbrauch liefert Infinity; eine verlangte EBus-Sonderausgabe erfordert eine passende Überschreibung.
- null als Treibstoffreferenz wird in Stringverkettung textuell dargestellt; vorgegebene Ausgabekonvention und physikalische Bedeutung unterscheiden.

Belege: 09: 5c · 10: 5c · 11: 5 · 12: 5 · 20: 5a-5b

B. Algorithmen und Datenstrukturen

B01 Algorithmusbegriff und Divide-and-Conquer

- Algorithmus als allgemeine Lösung einer Problemklasse; Abgrenzung von einer einzelnen Eingabe.
- Endliche, eindeutige und ausführbare Beschreibung elementarer Schritte statt einer vagen Problemlösung.
- Terminierung mit einem Ergebnis für jede zulässige Eingabe nach der in den Klausuren verwendeten Definition.
- Unabhängigkeit von einer konkreten Programmiersprache; ein korrekter Algorithmus muss nicht optimal effizient sein.
- Divide-and-Conquer: Problem zerlegen, kleinere gleichartige Teilprobleme rekursiv lösen, Basisfälle direkt behandeln und Teilergebnisse zusammensetzen.

Belege: 39: 1a, 1d, 1e · 40: 1a · 41: 1a · 42: 1a · 43: 1a

B02 Asymptotische Laufzeiten und Aufwand von Operationen

- O-Notation für konstante, logarithmische, lineare, linearithmische, quadratische, kubische und exponentielle Laufzeiten.
- Wachstumsraten vergleichen; dominante Summanden und konstante Faktoren erkennen.
- Polynomiale Ausdrücke umformen und Schranken durch Abschätzungen oder Quotienten begründen; Potenz und Multiplikation unterscheiden.
- Best Case, Average Case und Worst Case bei Such-, Sortier- und Datenstrukturoperationen unterscheiden.
- Elementzahl als Eingabegröße verwenden: Ein Schlüsselwert ist nicht die Länge des Arrays.
- Gesamtaufwand mehrerer Operationen bestimmen, insbesondere einmaliges Sortieren und anschließende viele Suchanfragen.
- Kostenmodelle für konstant oder linear teure Teilroutinen sowie höhenabhängige Baumoperationen berücksichtigen.

Ergänzungen aus den Lösungen

- Θ als enge Wachstumsklasse gegenüber einer bloßen O-Obergrenze; gröbere O-Angaben können ebenfalls mathematisch wahr sein.
- Gleiche Wachstumsklassen bedeuten keine identischen gemessenen Laufzeiten; konstante Faktoren können verschieden sein.
- Mehrdeutige Multiple-Choice-Vorgaben mit einer festen Anzahl erwarteter Antworten von der mathematisch wörtlichen O-Lesart unterscheiden.

Belege: 38: 1a, 1b · 39: 1b, 1c, 1f · 40: 1b, 1c, 2a, 5a · 41: 1b, 1c, 2a, 5b · 42: 1b, 2 · 43: 1b, 2 · 44: 1a, 1d, 2, 6 · 45: 1a, 1d, 2

B03 Schleifenanalyse, Rekursion und Rekursionsspeicher

- Exakte Anzahl von Teilroutinenaufrufen für konkrete Eingaben und als Funktion der Eingabegröße bestimmen.
- Inklusive und exklusive Schleifengrenzen, zusätzliche Aufrufe vor Schleifen und rechteckige Schleifennester berücksichtigen.
- Dreieckige Schleifennester über arithmetische Summen auswerten.
- Halbieren und Verdoppeln von Schleifenvariablen mit Zweierpotenzen und Logarithmen analysieren.
- Einen oder zwei rekursive Folgeaufrufe mit verringertem beziehungsweise halbiertem Argument in einem Rekursionsbaum darstellen.
- Zwischen Arbeit nur im Basisfall, nur in inneren Rekursionsknoten und in jedem Aufruf unterscheiden.
- Rekurrenzgleichungen und geometrische Summen zur Bestimmung der Gesamtaufrufzahl verwenden.

- Argumentabhängige Kosten je Aufruf und Gesamtaufwand je Rekursionsebene statt nur der Aufrufzahl bestimmen.
- Maximale Rekursionstiefe über den längsten Aufrufpfad bestimmen; nacheinander ausgeführte Geschwisteraufrufe erhöhen nicht die Tiefe.
- Rekursive und iterative Implementierbarkeit beurteilen, unter anderem für Fakultät, Suche, Sortierung und Baumtraversierung.

Ergänzungen aus den Lösungen

- Zählkonventionen der Rekursionstiefe offenlegen: gleichzeitig aktive Aufrufe einschließlich Erstaufruf oder nur rekursive Abstiege.
- Zusatzspeicher des Aufrufstapels bei linearer, logarithmischer oder von der Baumhöhe abhängiger Rekursion beurteilen.
- Einen expliziten Stapel zum Speichern ausstehender Besuche als Ersatz für den impliziten Rekursionsstapel erklären.

Belege: 38: 2a, 2b, 2c, 4a · 39: 2a, 2b, 2c · 40: 2a, 2b, 2c, 4c · 41: 1d, 2a, 2b, 2c, 4a · 42: 2 · 43: 1d, 2 · 44: 2 · 45: 2

B04 Java-Implementierung, Referenzen und Vergleiche

- Knoten- und Datenklassen mit Feldern, Konstruktoren, Objektreferenzen und Initialisierung leerer Strukturen verwenden.
- Öffentliche Schnittstellen, private Hilfsmethoden, Parameter, Rückgabewerte und parameterlose Wurzelmethoden unterscheiden.
- Generische Typen und Knoten mit Typparametern für unterschiedliche Datentypen einsetzen.
- Kontrollstrukturen, frühe Rückgaben, assert-Vorbedingungen, null sowie vereinbarte Rückgabesignale korrekt verwenden.
- Referenzidentität und inhaltliche equals-Vergleiche auseinanderhalten; Datenobjekte nach den vorgegebenen Feldern vergleichen.
- Comparable und compareTo für Gleichwertigkeit gemäß einer Ordnung einordnen.

Ergänzungen aus den Lösungen

- Sortiergleichheit durch compareTo ist nicht zwingend equals-Gleichheit oder Referenzidentität.
- Kurzschlussauswertung verhindert ungültige null-Zugriffe oder negative Feldindizes.
- Math.min, Math.max, Math.abs und Math.pow passend einsetzen; Ganzzahlüberläufe in Laufzeitmodellen und Zwischenrechnungen berücksichtigen, gegebenenfalls long verwenden.
- Private Methoden sind auch auf einer anderen Instanz derselben Klasse aufrufbar; aktivierte Assertions können zusätzlichen Aufwand verursachen.
- Fehlerhafte for-Schleifenköpfe und widersprüchliche Antwortvorgaben erkennen und notwendige Konventionen der Vorlage ausdrücklich benennen.

Belege: 38: 1c, 3a, 3b, 4a, 4b, 4c, 4d · 39: 3a, 3b, 3c, 4a, 4b, 4c · 40: 3a, 3b, 4a, 4b, 4c · 41: 3a, 3b, 4a, 4b, 5a · 42: 2, 3, 4, 5, 6 · 43: 2, 4, 5, 6 · 44: 1c, 2, 5 · 45: 1d, 2, 4, 5, 6

B05 List, Set, Map und allgemeine Operationen verketteter Listen

- List als positionsbezogene Folge mit Duplikaten, Set als Menge ohne Duplikate und Map als Zuordnung eindeutiger Schlüssel zu Werten unterscheiden.
- ArrayList und LinkedList hinsichtlich Anhängen und Indexzugriff vergleichen; vorhandene Arraykapazität beachten.
- Anfangs-, Nachfolger- und Endverweise sowie das null-Ende einer gültigen einfach verketteten Liste verstehen.
- Am Anfang, zwischen Knoten und am Ende einfügen; Kopf oder Nachfolger ausketten und den Rückgabewert vorher sichern.
- Das letzte Element beziehungsweise den Vorgänger eines Zielknotens finden und die Listenlänge zählen.

- Zwei Listen ohne Kopieren ihrer Knoten verketteten und die zweite Liste anschließend leeren.
- Eine Liste in zwei Hälften teilen; bei ungerader Länge das zusätzliche Element der vorderen Hälfte zuordnen.
- Nachfolger entfernen und jedes zweite Element einer Liste löschen.
- Durch Einfügen am Kopf eine neue Liste in umgekehrter Reihenfolge erzeugen.
- Leere Listen, Einzelknoten, getrennte Ketten und Ausschluss von Selbstverkettung berücksichtigen.
- Den Aufwand mit und ohne vorhandenen Endzeiger vergleichen; korrekt gepflegte Anfangs- und Endverweise erhalten.

Ergänzungen aus den Lösungen

- Die Java-LinkedList mit Endzeiger von einer eigenen einfach verketteten Liste ohne Endzeiger unterscheiden.
- Flache gegenüber tiefer Listenkopie unterscheiden: Neue Knoten können dieselben Datenobjekte referenzieren.

Belege: 38: 1c, 3a, 3b · 39: 1c, 3a, 3b, 3c · 40: 3a, 3b · 41: 3a, 3b · 42: 1c, 3 · 43: 1c, 4 · 44: 1a, 1c, 4 · 45: 1a, 4

B06 Prioritätslisten, Warteschlangen und Stapel

- Eine Prioritätsliste absteigend nach Prioritäten halten und die passende Einfügestelle einschließlich gleicher Prioritäten finden.
- In einer sortierten Prioritätsliste die Aktion höchster Priorität am Kopf entfernen.
- In einer unsortierten Prioritätsliste vorn einfügen und durch vollständigen Durchlauf das erste maximale Element finden.
- Das gefundene Maximum mithilfe seines Vorgängers entfernen; Kopf und inneren Knoten getrennt behandeln.
- FIFO-Warteschlange als einfach verkettete Liste implementieren: hinten enqueue, vorn dequeue.
- Eine Warteschlange ohne Endzeiger um eine noch nicht enthaltene Person erweitern; Duplikate über equals erkennen.
- LIFO-Stapel mit push und pop implementieren; einen bereits enthaltenen gleichen Auftrag vor erneutem Ablegen oben entfernen.
- Eindeutigkeit der gespeicherten Aufträge sowie konstante beziehungsweise lineare Operationskosten begründen.

Ergänzungen aus den Lösungen

- Bei der Maximumsuche das erste Maximum des bisherigen Präfixes als Schleifeninvariante nutzen; Initialisierung ohne willkürlichen numerischen Startwert behandelt auch negative Prioritäten.

Belege: 38: 3a, 3b, 6c · 39: 3a, 3b, 3c · 40: 3a, 3b · 41: 3a, 3b

B07 Binärbäume, Suchbäume und Traversierungen

- Wurzel, Eltern, linke und rechte Kinder, Teilbäume, Blätter und innere Knoten unterscheiden; Kinder auch über ein Referenzfeld darstellen.
- Höhe rekursiv aus Teilbaumhöhen berechnen und die Konvention Ebenen gegenüber Kanten beachten.
- Blätter und innere Knoten erkennen und rekursiv zählen; leere Teilbäume als Basisfälle behandeln.
- Beziehungen zwischen Höhe, maximaler Knotenzahl und maximaler Blattzahl sowie perfekte Bäume untersuchen.
- Minimum und Maximum eines beliebigen Binärbaums durch Untersuchung beider Teilbäume bestimmen.
- Suchbaumordnung als globale Einschränkung aller Schlüssel der Teilbäume verstehen; Aussagen über Kind- und Vorfahrenschlüssel prüfen.
- Die Suchbaumeigenschaft rekursiv mit linkem Teilbaummaximum und rechtem Teilbaumminimum überprüfen; reine Eltern-Kind-Vergleiche reichen nicht aus.

- Im Suchbaum Minimum beziehungsweise Maximum iterativ über den äußersten linken beziehungsweise rechten Pfad finden.
- Höhenabhängige Such- und Einfügekosten sowie die mögliche Entartung zu einer linearen Kette berücksichtigen.
- Iterativ prüfen, ob jeder Knoten höchstens ein Kind hat; auch Richtungswechsel entlang einer Kette sind zulässig.
- Preorder, Inorder und Postorder einschließlich ihres Laufzeitvergleichs einordnen.
- Schlüssel mit umgekehrter Inorder-Traversierung absteigend ausgeben und Leerzeichen zwischen Teilfolgen korrekt zusammensetzen.

Ergänzungen aus den Lösungen

- Perfekte Bäume von linksbündig vollständigen Bäumen unterscheiden; die alternative Prüfung zählt Knoten und kontrolliert lückenlose Feldindizes.
- Konventionen für gleiche Suchbaumschlüssel festlegen und Vergleichsoperatoren daran anpassen.
- Wiederholte Minimum-/Maximumdurchläufe können quadratische Prüfkosten erzeugen; weitergegebene Wertebereiche erlauben eine lineare Alternative.
- Unveränderliche Strings können trotz einmaliger Traversierung durch wiederholtes Kopieren quadratischen Zeichenkopieraufwand verursachen.

Belege: 38: 1b, 4a, 4b, 4c, 4d · 39: 4a, 4b, 4c · 40: 4a, 4b, 4c · 41: 1d, 4a, 4b · 42: 1c · 43: 1c, 1d · 44: 1c, 1d · 45: 1b, 1c, 1d, 5

B08 AVL-Bäume und Balancebedingungen

- AVL-Eigenschaft als Suchbaumordnung mit einer Höhendifferenz von höchstens eins an jedem Knoten verstehen.
- Balancefaktoren -1, 0 und +1 sowie gültige beziehungsweise ungültige Beispielsbäume beurteilen; die Wurzel besitzt keinen Elternknoten.
- Logarithmische Baumhöhe und daraus folgende Such- und Einfügekosten begründen.
- Rekursiv zugleich Höhe und AVL-Gültigkeit berechnen, verletzte Balance über ein Fehlersignal weitergeben und Teilbaumhöhen nicht mehrfach berechnen.
- Bäume mit überall ausgeglichenen Teilbaumhöhen sowie den Zusammenhang zu perfekten Bäumen und deren Knotenzahl untersuchen.

Ergänzungen aus den Lösungen

- Rotationen werden als Mittel zur Erhaltung der AVL-Balance erläutert; eine konkrete Rotationsfolge wird in diesen Klausuren nicht ausgeführt.

Belege: 38: 1b · 40: 4b, 4c · 42: 1d · 44: 1b · 45: 1b

B09 Binäre Heaps und Prüfung der Heapordnung

- Formeigenschaft eines Heaps von seiner Schlüsselordnung unterscheiden: vollständige obere Ebenen und linksbündige letzte Ebene.
- Min-Heap: Elternschlüssel höchstens so groß wie Kindschlüssel; Max-Heap: mindestens so groß; Geschwister müssen nicht sortiert sein.
- Heapordnung von Suchbaumordnung abgrenzen.
- Den größten Schlüssel eines gültigen Max-Heaps an der Wurzel ablesen.
- Die lokale Schlüsselordnung eines Knotens und anschließend die Ordnung aller Teilbäume rekursiv prüfen.
- Gleiche Schlüssel, fehlende Kinder und leere Teilbäume bei der Prüfung berücksichtigen.

Ergänzungen aus den Lösungen

- Eine reine Prüfung der Schlüsselordnung beweist nicht automatisch die Formeigenschaft; der Aufwand vorgeschalteter Assertions ist getrennt vom eigentlichen Wurzelzugriff zu betrachten.

Belege: 38: 1d · 42: 1c · 44: 1c, 5

B10 B-Bäume: Suche, Ausgabe, Höhe und Ordnung

- Mehrere sortierte Schlüssel je Knoten und zugehörige Kindintervalle darstellen.
- Ein Kindfeld mit einem Eintrag mehr als Schlüssel und null-Kinder der Blätter beachten; Knoten dürfen unterschiedlich viele Schlüssel enthalten.
- Iterativ und rekursiv nach einem Schlüssel suchen und das passende Kindintervall auswählen.
- Schlüssel rekursiv in absteigender Reihenfolge über eine verallgemeinerte umgekehrte Inorder-Traversierung ausgeben.
- Höhe iterativ entlang eines Pfads bestimmen, weil die Blätter eines gültigen B-Baums auf derselben Ebene liegen.
- Die vorgegebene Ordnung m rekursiv prüfen: Wurzel mit einem bis 2m Schlüsseln, übrige Knoten mit m bis 2m Schlüsseln.
- Leere Bäume sowie Leerzeichen beim Zusammensetzen der Ausgabe behandeln.

Ergänzungen aus den Lösungen

- Bei einer verlangten Belegungsprüfung bereits vorausgesetzte B-Baum-Eigenschaften von tatsächlich neu geprüften Bedingungen trennen; Zwischenrechnungen für Obergrenzen gegen Überlauf absichern.

Belege: 42: 4 · 43: 5

B11 Lineare, binäre und Interpolationssuche

- Lineare Suche auf geordneten und ungeordneten Schlüsseln einschließlich des Aufwands bis zum letzten Element analysieren.
- Sortierung als Voraussetzung der binären Suche erkennen.
- Inklusive Suchgrenzen, Mittelindex und Rundung festlegen und nach jedem Vergleich den Bereich verkleinern.
- Ablaufspuren erfolgreicher und erfolgloser Suche mit Indexbereichen und Vergleichselementen führen; bei leerem Intervall abbrechen.
- Maximale Vergleichszahl und Schlüssel mit maximalem Suchaufwand bestimmen.
- Interpolationssuche mit wertabhängiger Positionsschätzung, Abrundung und anschließender Intervallfortschreibung ausführen.
- Durchschnitts- und Worst-Case-Verhalten der Verfahren vergleichen; für die Interpolationssuche passende Schlüsselverteilungen berücksichtigen.
- Mathematische Bruchauswertung von vorzeitiger Ganzzahldivision unterscheiden.

Ergänzungen aus den Lösungen

- Eine andere zulässige Rundung des Mittelindex kann eine andere korrekte Vergleichsfolge ergeben.
- Bloße Sortierung garantiert nicht die günstige mittlere Interpolationslaufzeit; geeignete Gleichverteilung ist eine zusätzliche Annahme.

Belege: 38: 5a · 39: 5a · 40: 1c · 41: 1c, 1d · 42: 1b, 5a · 43: 1b, 1d, 6a · 44: 1d, 6 · 45: 1d

B12 Hash-Tabellen und Kollisionsauflösung

- Hash-Funktion, berechnete Hash-Adresse und tatsächliche Speicherposition unterscheiden; Modulo-Funktionen verwenden.
- Kollisionen erkennen und bei offenem Adressieren freie Plätze durch Sondieren suchen.
- Quadratische Sondierfolgen von der ursprünglichen Hash-Adresse aus berechnen.
- Linear mit vorgegebener Schrittweite sondieren und am Tabellenende zyklisch fortsetzen.
- Schlüssel in vorgegebener Reihenfolge einfügen, alle geprüften Adressen und die endgültige Belegung nachvollziehen.
- Aus einer gefüllten Tabelle auf zwingend zuvor aufgetretene Kollisionen schließen.
- Belegungsgrad und Kollisionswahrscheinlichkeit bei gleichverteilten Hashwerten untersuchen.

- Einfügen bei nur einem freien Platz sowie erwartete und schlechteste Suchkosten analysieren; Hashing von binärer und Interpolationssuche abgrenzen.

Ergänzungen aus den Lösungen

- Einseitige und alternierende Quadratfolgen als unterschiedliche Konventionen unterscheiden; ein in einer MC-Option genannter Hash-Puffer ist keine dort definierte Kollisionsstrategie.
- Teilerfremdheit von Schrittweite und Tabellengröße erklärt, wann lineares Sondieren alle Plätze erreichen kann.

Belege: 38: 5b · 39: 5b · 40: 1d · 42: 5b · 43: 1b, 6b · 45: 6

B13 Sortiervverfahren und Vergleich ihrer Laufzeiten

- Average-Case- und Worst-Case-Komplexität der in den Wissensfragen genannten Sortiervverfahren vergleichen.
- Mergesort als Divide-and-Conquer-Verfahren mit Vereinigung sortierter Teilfolgen und linearer Arbeit je Ebene verstehen; die Worst-Case-Komplexität mit Bottom-Up-Mergesort vergleichen.
- Quicksort mit Pivot, Partitionierung und rekursiven Teilbereichen einordnen; mittleres und schlechtestes Verhalten unterscheiden.
- Heapsort durch Heapaufbau, Entnahmen und Wiederherstellung der Heapordnung erklären; iterative und rekursive Umsetzung einordnen.
- Bottom-Up-Heapsort hinsichtlich seiner asymptotischen Klasse und möglicher Konstantenunterschiede mit Heapsort und Quicksort vergleichen.
- Bubblesort und Insertionsort insbesondere im durchschnittlichen beziehungsweise schlechtesten Fall vergleichen.
- Direktes Insertionsort auf bereits sortierten Feldern anhand seiner Schlüsselvergleiche untersuchen.
- Selectionsort in die Laufzeitvergleiche einbeziehen.
- Schlüsselvergleiche von Schleifenprüfungen unterscheiden und Sortieren als Vorbereitung wiederholter Suchanfragen bewerten.

Ergänzungen aus den Lösungen

- Bottom-Up-Mergesort vereinigt iterativ zunehmend längere sortierte Läufe und widerlegt die Beschränkung von Mergesort auf Rekursion.

Belege: 39: 1e, 1f · 40: 1b · 41: 1b, 1d · 42: 1b · 43: 1b, 1d · 44: 1d, 6.3 · 45: 1d

B14 Cocktail-Sort und Gnome-Sort schrittweise ausführen

- Cocktail-Sort mit abwechselndem Vorwärts- und Rückwärtsdurchlauf über benachbarte Elemente ausführen.
- Den aktiven Feldbereich durch start und ende verkleinern, sortierte Teilfelder abgrenzen und ein Tauschkennzeichen für den Abbruch verwenden.
- Den Feldzustand nach jeder for-Iteration und jeder Grenzänderung dokumentieren, auch ohne Tausch oder bei einer Schleife ohne Durchlauf.
- Gnome-Sort durch Vorwärtsgehen bei geordnetem Paar und Tauschen mit anschließendem Rückwärtsschritt ausführen.
- Index und Feldinhalt nach jeder while-Iteration festhalten; Feldanfang und Abbruch an der Feldlänge korrekt behandeln.
- Nachbartausch mithilfe einer temporären Variable nachvollziehen.
- Best- und Worst-Case-Vergleichszahlen beider Verfahren aus dem Verhalten auf geordneten beziehungsweise ungünstigen Folgen begründen.

Ergänzungen aus den Lösungen

- Inversionen einer rückwärts sortierten Folge erklären den quadratischen Worst Case; jeder Nachbartausch beseitigt genau eine Fehlordnung.

Belege: 40: 5a, 5b · 41: 5a, 5b

B15 Graphdarstellung, Veränderungen und Zusammenhang

- Knoten, gerichtete oder ungerichtete Kanten, Gewichte und isolierte Knoten unterscheiden.
- Adjazenzmatrix lesen: Zeile als Ausgangsknoten, Spalte als Zielknoten, leeres Feld als fehlende Kante.
- Gerichtetheit anhand der Symmetrie prüfen und gegenläufige Kanten mit unterschiedlichen Gewichten getrennt behandeln.
- Adjazenzlisten lesen, zwischen Listen und Matrix wechseln und vollständige Graphzeichnungen anfertigen.
- Parametrisierte Graphen aus einer Kantenregel aufbauen und ihre Kanten zählen.
- Speicherbedarf von Adjazenzmatrix und Adjazenzlisten in Abhängigkeit von Knoten- und Kantenzahl vergleichen.
- Knoten samt ein- und ausgehenden Kanten entfernen, Kanten ergänzen und resultierende Zusammenhangskomponenten bestimmen.
- Den zugrunde liegenden ungerichteten Graphen durch Entfernen der Richtungen bilden; gleich gewichtete Gegenkanten zu einer Verbindung zusammenfassen.

Ergänzungen aus den Lösungen

- Listenverkettungen in einer Adjazenzlistenzeichnung sind keine zusätzlichen Graphkanten; bei gerichteten Graphen schwachen und starken Zusammenhang ausdrücklich unterscheiden.

Belege: 38: 6a, 6c · 39: 6a · 40: 6a, 6b, 6d · 41: 6a, 6b · 42: 6a, 6b, 6c · 43: 1c, 3a · 44: 3 · 45: 3

B16 Topologische Sortierung und gerichtete Kreisfreiheit

- Eine topologische Reihenfolge so angeben, dass jeder Ausgangsknoten vor dem Ziel seiner Kanten steht.
- Existenz an die Kreisfreiheit des gerichteten Graphen knüpfen und mehrere zulässige Reihenfolgen erkennen.
- Einen gerichteten Zyklus als Widerspruch zu einer topologischen Reihenfolge nachweisen.
- Kreisfreiheit mithilfe streng wachsender Knotennummern entlang der Kanten begründen.
- Durch Ergänzen einer geeigneten Rückkante gezielt einen gerichteten Zyklus erzeugen.

Belege: 38: 6b · 40: 6c · 43: 3b · 44: 3b, 3c, 3d · 45: 3c, 3e

B17 Breitensuche mit Warteschlange

- Breitensuche von einem vorgegebenen Startknoten im zugrunde liegenden ungerichteten Graphen ausführen.
- Den nächsten Knoten vorne aus einer FIFO-Warteschlange entnehmen und unbesuchte Nachbarn hinten einreihen.
- Vorgegebene alphabetische Wahlregeln bei mehreren Nachbarn einhalten.
- Den Warteschlangeninhalt nach jedem Bearbeitungsschritt dokumentieren.
- Die resultierende Besuchsreihenfolge und Entfernungsebenen vom Start bestimmen.

Ergänzungen aus den Lösungen

- Bereits beim Einreihen markieren, damit ein Knoten nicht mehrfach in der Warteschlange landet.

Belege: 38: 6c

B18 Dijkstra und Rekonstruktion kürzester Wege

- Dijkstra auf gewichteten Graphen mit nichtnegativen Kantengewichten anwenden.
- Distanzen und Vorgänger initialisieren: Startdistanz, vorläufige Unerreichbarkeit und fehlender Vorgänger.
- Den noch unmarkierten Knoten mit kleinster vorläufiger Distanz auswählen und markieren.
- Ausgehende Kanten entspannen, kürzere Entfernungen übernehmen und Vorgänger aktualisieren.

- Distanz-/Vorgängertabellen und Markierungsreihenfolge schrittweise fortführen.
- Den Weg durch Rückwärtsverfolgung und Umkehrung der Vorgängerkette rekonstruieren, Gewichte summieren und mit dem endgültigen Distanzwert vergleichen.

Belege: 39: 6b, 6c · 41: 6c, 6d · 43: 3c, 3d

B19 Minimale Spannbäume, Kruskal und Union-Find

- Einen minimalen Spannbaum eines ungerichteten gewichteten Graphen bestimmen.
- Kruskal-Kanten nach aufsteigendem Gewicht bearbeiten und Verbindungen zwischen verschiedenen Komponenten aufnehmen.
- Kanten verwerfen, die einen Kreis schließen würden; nach der erforderlichen Zahl aufgenommener Baumkanten beenden.
- Bearbeitungsreihenfolge, vollständigen Spannbaum und Gesamtgewicht darstellen.
- Union-Find zur Verwaltung der Komponenten mit Repräsentanten und Elternzeigern nachvollziehen.
- Die geforderte Union-Find-Endstruktur zur gewählten Kantenfolge aufstellen und ihre Komponenten mit dem Spannbaum abgleichen.

Ergänzungen aus den Lösungen

- Vereinigung nach Größe, Gleichstandsregeln und Pfadkompression können unterschiedliche korrekte Endbaumformen erzeugen, ohne die dargestellte Komponentenpartition zu verändern.

Belege: 40: 6d · 42: 6d

C. Quellenverzeichnis

Die Klausur-ID entspricht der Nummer am Anfang des Originaldateinamens. Alle 32 Dateien wurden berücksichtigt. Die Aufgaben liegen im Hauptordner, die Lösungen in **output/pdf/**.

09 | 19.02.2018 | Einführung in die Programmierung

Aufgaben: 09_EidP_Klausur_19.02.2018.pdf (16 S.)

Lösung: 09_EidP_Klausur_19.02.2018_Loesungen.pdf (10 S.)

10 | 18.07.2018 | Einführung in die Programmierung

Aufgaben: 10_EidP_Klausur_18.07.2018.pdf (18 S.)

Lösung: 10_EidP_Klausur_18.07.2018_Loesungen.pdf (10 S.)

11 | 19.02.2019 | Einführung in die Programmierung

Aufgaben: 11_EidP_Klausur_19.02.2019.pdf (18 S.)

Lösung: 11_EidP_Klausur_19.02.2019_Loesungen.pdf (11 S.)

12 | 22.07.2019 | Einführung in die Programmierung

Aufgaben: 12_EidP_Klausur_22.07.2019.pdf (16 S.)

Lösung: 12_EidP_Klausur_22.07.2019_Loesungen.pdf (12 S.)

15 | 10.03.2021 | Einführung in die Programmierung

Aufgaben: 15_EidP_Klausur_A_10_03_2021.pdf (3 S.)

Lösung: 15_EidP_Klausur_A_10_03_2021_Loesungen.pdf (6 S.)

16 | 27.08.2021 | Einführung in die Programmierung

Aufgaben: 16_EidP_Klausur_27.08.2021.pdf (5 S.)

Lösung: 16_EidP_Klausur_27.08.2021_Loesungen.pdf (9 S.)

17 | 21.02.2022 | Einführung in die Programmierung

Aufgaben: 17_EidP_Klausur_A_21.02.2022.pdf (6 S.)

Lösung: 17_EidP_Klausur_A_21.02.2022_Loesungen.pdf (9 S.)

20 | 28.07.2023 | Einführung in die Programmierung

Aufgaben: 20_EidP_Klausur_28.07.2023 (MI und INF).pdf (11 S.)

Lösung: 20_EidP_Klausur_28.07.2023 (MI und INF)_Loesungen.pdf (13 S.)

Quellenverzeichnis - Fortsetzung

38 | 11.08.2022 | Algorithmen und Datenstrukturen

Aufgaben: 38-Klausur vom 11.08.2022_korr.pdf (10 S.)

Lösung: 38-Klausur vom 11.08.2022_korr_Loesungen.pdf (7 S.)

39 | 08.02.2023 | Algorithmen und Datenstrukturen

Aufgaben: 39-Klausur vom 08.02.2023_final.pdf (11 S.)

Lösung: 39-Klausur vom 08.02.2023_final_Loesungen.pdf (7 S.)

40 | 02.08.2023 | Algorithmen und Datenstrukturen

Aufgaben: 40-Klausur vom 02.08.2023_korr.pdf (15 S.)

Lösung: 40-Klausur vom 02.08.2023_korr_Loesungen.pdf (8 S.)

41 | 09.02.2024 | Algorithmen und Datenstrukturen

Aufgaben: 41-Klausur vom 09.02.2024_final.pdf (15 S.)

Lösung: 41-Klausur vom 09.02.2024_final_Loesungen.pdf (7 S.)

42 | 20.08.2024 | Algorithmen und Datenstrukturen

Aufgaben: 42-Klausur vom 20.08.2024_final.pdf (15 S.)

Lösung: 42-Klausur vom 20.08.2024_final_Loesungen.pdf (7 S.)

43 | 31.01.2025 | Algorithmen und Datenstrukturen

Aufgaben: 43-Klausur vom 31.01.2025_final.pdf (15 S.)

Lösung: 43-Klausur vom 31.01.2025_final_Loesungen.pdf (6 S.)

44 | 14.08.2025 | Algorithmen und Datenstrukturen

Aufgaben: 44-Klausur vom 14.08.2025_final.pdf (13 S.)

Lösung: 44-Klausur vom 14.08.2025_final_Loesungen.pdf (6 S.)

45 | 06.02.2026 | Algorithmen und Datenstrukturen

Aufgaben: 45-Klausur vom 06.02.2026_final_korr.pdf (15 S.)

Lösung: 45-Klausur vom 06.02.2026_final_korr_Loesungen.pdf (6 S.)