

Lösungen: Algorithmen und Datenstrukturen - 11.08.2022

Ausgangsdatei: 38-Klausur vom 11.08.2022_korr.pdf

Diese Ausarbeitung enthält Lösungen und Begründungen zu allen sechs Aufgaben. Bei Rekursionstiefen werden sowohl die Anzahl gleichzeitig aktiver Aufrufe als auch die Anzahl rekursiver Abstiege angegeben, damit unterschiedliche Zählkonventionen klar bleiben.

Aufgabe 1 - Wissen

a) Aussagen zur O-Notation

Nr.	Aussage	Lösung und Begründung
1	$(n + 1)(n + 1) = O(n^2)$	Richtig: $n^2 + 2n + 1 \leq 4n^2$ für $n \geq 1$.
2	$n^2 = O(n \log n)$	Falsch: $n^2 / (n \log n) = n / \log n$ wächst unbeschränkt.
3	$n^2 = O(2^n)$	Richtig: Die Exponentialfunktion wächst langfristig schneller als jedes feste Polynom.

b) Komplexität von Suchbäumen im schlechtesten Fall

Operation	Binärer Suchbaum	AVL-Baum
Einfügen	$O(n)$	$O(\log n)$
Suchen	$O(n)$	$O(\log n)$

Ein gewöhnlicher Suchbaum kann zu einer Kette der Länge n entarten. Ein AVL-Baum hält seine Höhe durch die Balancebedingung und Rotationen logarithmisch; Suchen und Einfügen folgen einem Wurzel-Blatt-Pfad.

c) List, Set und Map

List<E> speichert Elemente in einer geordneten, über Positionen zugänglichen Folge und erlaubt doppelte Elemente.

Set<E> speichert eine Menge von Elementen ohne Duplikate gemäß dem verwendeten Gleichheitsbegriff.

Map<K,V> ordnet jedem enthaltenen eindeutigen Schlüssel genau einen Wert zu; verschiedene Schlüssel dürfen denselben Wert besitzen.

d) Heap-Eigenschaften

Ein binärer Heap besitzt eine Formeigenschaft und eine Ordnungseigenschaft: Alle Ebenen außer eventuell der letzten sind vollständig besetzt, und die letzte Ebene wird ohne Lücken von links aufgefüllt. Beim Max-Heap ist jeder Elternschlüssel mindestens so groß wie die Schlüssel seiner Kinder; beim Min-Heap ist er höchstens so groß. Zwischen Geschwistern ist keine bestimmte Reihenfolge gefordert.

Aufgabe 2 - Komplexität und Rekursion

a) Anzahl der Aufrufe und Laufzeit

Für die Formeln sei $n \geq 1$.

Methode	Aufrufe bei $n = 4$	Aufrufe als Funktion von n	Zeitkomplexität
proz1	15	$(n - 1)(n + 1) = n^2 - 1$	$O(n^2)$
proz2	7	$2^{(n - 1)} - 1$	$O(2^n)$

Bei proz1 läuft die äußere Schleife $n - 1$ Mal. Pro Durchlauf erfolgen ein Aufruf vor und n Aufrufe innerhalb der inneren Schleife. Daher entstehen bei $n = 4$ genau $3 \cdot 5 = 15$ Aufrufe.

Für proz2 gilt $A(1) = 0$ und $A(n) = 2A(n - 1) + 1$. Auf den Ebenen vor dem Basisfall stehen 1, 2, 4, ..., $2^{(n - 2)}$ Aufrufe von tuwas. Die geometrische Summe beträgt $2^{(n - 1)} - 1$. Bei $n = 4$ sind es $1 + 2 + 4 = 7$. Einschließlich der Basisfälle werden $2^n - 1$ Methodenaufrufe ausgeführt, was ebenfalls $O(2^n)$ ergibt.

b) Rekursionstiefe für $n = 4$

Der längste Aufrufpfad lautet $\text{proz2}(4) \rightarrow \text{proz2}(3) \rightarrow \text{proz2}(2) \rightarrow \text{proz2}(1)$. Es gibt 4 gleichzeitig aktive Aufrufe, entsprechend 3 rekursive Abstiege. Die beiden Aufrufe mit $n - 1$ erfolgen nacheinander und verdoppeln daher nicht die Stapeltiefe.

c) Rekursionstiefe für allgemeines n

Für $n \geq 1$ sind maximal n Aufrufe gleichzeitig aktiv; bei Zählung nur der rekursiven Abstiege beträgt die Tiefe $n - 1$. Der zusätzliche Speicherbedarf ist $O(n)$.

Aufgabe 3 - Prioritätsliste

a) Sortiert einfügen

```
public void einfuegen(String aktion, int prioritaet) {
    assert(aktion != null);
    if (anfang == null || prioritaet > anfang.prioritaet) {
        anfang = new Link(aktion, prioritaet, anfang);
        return;
    }
    Link vorgaenger = anfang;
    while (vorgaenger.naechster != null
        && vorgaenger.naechster.prioritaet >= prioritaet) {
        vorgaenger = vorgaenger.naechster;
    }
    vorgaenger.naechster =
        new Link(aktion, prioritaet, vorgaenger.naechster);
}
```

Eine leere Liste und das Einfügen vor dem bisherigen Kopf werden gesondert behandelt. Ansonsten wird die Stelle hinter dem letzten Element mit mindestens ebenso hoher Priorität gesucht. Der neue Knoten verweist zuerst auf den bisherigen Nachfolger, anschließend wird der Vorgänger mit ihm verbunden. Gleiche Prioritäten werden hier hinter bereits vorhandenen gleichen Prioritäten eingefügt; das ist laut Aufgabe zulässig. Laufzeit im schlechtesten Fall: $O(n)$.

b) Aktion mit höchster Priorität entfernen

```
public String entferneAktionMitHoechsterPrioritaet() {
    if (anfang == null) {
        return null;
    }
    String ergebnis = anfang.aktion;
    anfang = anfang.naechster;
    return ergebnis;
}
```

Wegen der absteigenden Sortierung steht eine Aktion maximaler Priorität immer am Anfang. Nach dem Sichern des Rückgabewerts wird der Kopf auf den Nachfolger gesetzt. Das funktioniert auch für eine Liste mit genau einem Element. Laufzeit: $O(1)$.

Aufgabe 4 - Binärbäume

a) Höhe rekursiv bestimmen

```
private int hoehe(Knoten knoten) {
    if (knoten == null) {
        return 0;
    }
    return 1 + Math.max(hoehe(knoten.linkesKind),
                        hoehe(knoten.rechtesKind));
}
```

Die Höhe wird hier in Ebenen gezählt: leerer Baum 0, einzelnes Blatt 1. Ein nichtleerer Baum ist eine Ebene höher als sein höherer Teilbaum. Jeder Knoten wird einmal besucht; Laufzeit $O(n)$, Rekursionsspeicher $O(h)$.

b) Blatteigenschaft prüfen

```
private boolean istBlatt(Knoten knoten) {
    assert(knoten != null);
    return knoten.linkesKind == null
        && knoten.rechtesKind == null;
}
```

Ein Blatt besitzt weder ein linkes noch ein rechtes Kind. Ein Knoten mit nur einem Kind ist kein Blatt.

c) Blätter zählen

```
private int anzahlBlaetter(Knoten knoten) {
    if (knoten == null) {
        return 0;
    }
    if (istBlatt(knoten)) {
        return 1;
    }
    return anzahlBlaetter(knoten.linkesKind)
        + anzahlBlaetter(knoten.rechtesKind);
}
```

Ein leerer Teilbaum trägt kein Blatt bei, ein Blatt genau eines. Bei einem inneren Knoten addieren sich die Blattzahlen der disjunkten Teilbäume. Laufzeit $O(n)$.

d) Vollständigkeit prüfen

Der Begriff „vollständig“ wird in Lehrmaterialien unterschiedlich benutzt. Die vorgegebenen Hilfsmethoden Höhe und Blattzahl legen hier die Bedeutung „perfekter Binärbaum“ nahe: Jede Ebene ist vollständig besetzt, alle Blätter liegen auf derselben Ebene. Für diese Bedeutung lautet die verlangte Lösung:

```
public boolean istVollstaendig() {
    if (wurzel == null) {
        return true;
    }
    return anzahlBlaetter() == Math.pow(2, hoehe() - 1);
}
```

Ein Binärbaum mit Höhe h besitzt höchstens $2^{(h-1)}$ Blätter. Er erreicht diese Zahl genau dann, wenn beide Teilbäume jeweils die maximal mögliche Blattzahl erreichen, also alle Ebenen

vollständig besetzt sind. Math.pow vermeidet einen versehentlich zu kleinen beziehungsweise überlaufenden int-Shift.

Falls in der Vorlesung „vollständig“ stattdessen „bis auf die letzte Ebene voll, letzte Ebene linksbündig gefüllt“ bedeutet, genügt die obige Gleichheit nicht: Beispielsweise ist eine Wurzel mit nur einem linken Kind in dieser zweiten Bedeutung vollständig. Eine vollständige Lösung für diese alternative Definition ist:

```
public boolean istVollstaendig() {
    int n = anzahlKnoten(wurzel);
    return lueckenlos(wurzel, 0L, n);
}

private int anzahlKnoten(Knoten k) {
    if (k == null) {
        return 0;
    }
    return 1 + anzahlKnoten(k.linkesKind)
        + anzahlKnoten(k.rechtesKind);
}

private boolean lueckenlos(Knoten k, long index, int n) {
    if (k == null) {
        return true;
    }
    if (index >= n) {
        return false;
    }
    return lueckenlos(k.linkesKind, 2 * index + 1, n)
        && lueckenlos(k.rechtesKind, 2 * index + 2, n);
}
```

Bei der üblichen Feldnummerierung hat die Wurzel Index 0 und ein Knoten i die Kinder $2i + 1$ und $2i + 2$. Genau dann, wenn alle n Knoten die Indizes 0 bis $n - 1$ belegen, ist der Baum linksbündig vollständig. Beide gezeigten Varianten laufen in $O(n)$.

Aufgabe 5 - Suchen

a) Binäre Suche nach 9

Es werden inklusive Grenzen links und rechts und der nach unten abgerundete Mittelindex $m = (\text{links} + \text{rechts}) / 2$ verwendet.

Schritt	Indexbereich	Vergleichsin dex m	Feldwert	Entscheidung
1	[0, 8]	4	10	$9 < 10$: rechts = 3
2	[0, 3]	1	5	$9 > 5$: links = 2
3	[2, 3]	2	6	$9 > 6$: links = 3
4	[3, 3]	3	9	Gefunden an Index 3

Der Vergleich halbiert jeweils den noch möglichen Suchbereich. Bei anderer Rundung des Mittelindex kann die Folge abweichen; hier ist die verwendete Konvention explizit festgelegt.

b) Hash-Tabelle mit quadratischem Sondieren

Verwendete Standardkonvention: $h_i(k) = (k \bmod 11 + i^2) \bmod 11$, $i = 0, 1, 2, \dots$. Jeder neue Sonderversuch wird von der ursprünglichen Hash-Adresse aus berechnet.

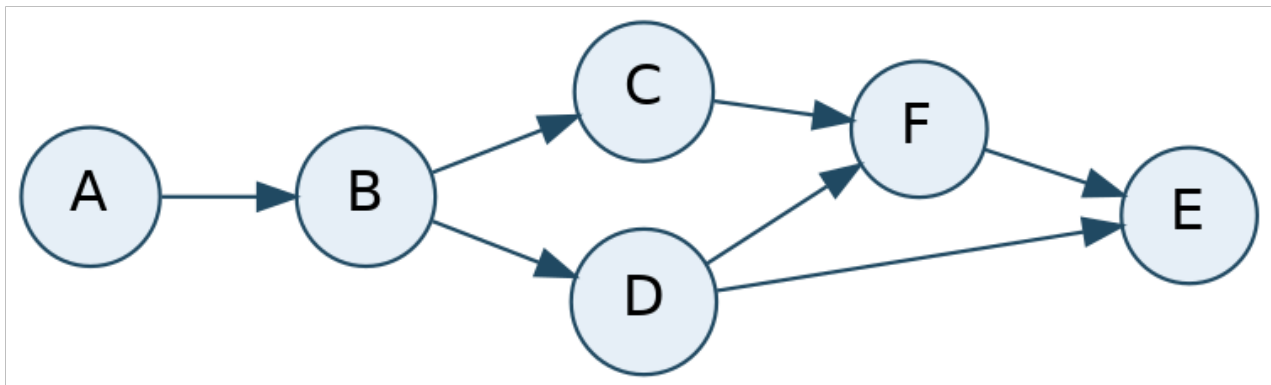
Schlüssel	Geprüfte Indizes und Rechnung	Endindex
4	$4 \bmod 11 = 4$	4
13	$13 \bmod 11 = 2$	2
26	$26 \bmod 11 = 4$ belegt; $(4 + 1^2) \bmod 11 = 5$	5
8	$8 \bmod 11 = 8$	8
24	$24 \bmod 11 = 2$ belegt; $(2 + 1^2) \bmod 11 = 3$	3
15	4 belegt; 5 belegt; $(4 + 2^2) \bmod 11 = 8$ belegt; $(4 + 3^2) \bmod 11 = 2$ belegt; $(4 + 4^2) \bmod 11 = 9$ frei	9

Index	0	1	2	3	4	5	6	7	8	9	10
Inhalt	leer	leer	13	24	4	26	leer	leer	8	15	leer

Die Aufgabe legt die genaue Sondierfolge nicht ausdrücklich fest. Falls die Vorlesung die ebenfalls verbreitete Variante mit Offsets $0, +1^2, -1^2, +2^2, -2^2, \dots$ verwendet, wird nur der letzte Eintrag anders: 15 prüft dann 4, 5, 3, 8, 0 und landet bei Index 0 statt 9. Die übrigen Einträge bleiben gleich.

Aufgabe 6 - Graphen

a) Graph aus der Adjazenzmatrix



Die Zeile ist der Ausgangsknoten, die Spalte der Zielknoten. Damit sind die Kanten $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow D$, $C \rightarrow F$, $D \rightarrow E$, $D \rightarrow F$ und $F \rightarrow E$ enthalten. Leere Matrixfelder bedeuten, dass keine entsprechende Kante existiert.

b) Topologische Sortierung

Ja. Der gerichtete Graph ist azyklisch. Eine mögliche topologische Reihenfolge ist A, B, C, D, F, E. Bei jeder Kante steht der Ausgangsknoten in dieser Folge vor dem Zielknoten. C und D können auch vertauscht werden.

c) Breitensuche im zugrunde liegenden ungerichteten Graphen ab E

Eine Kante darf nun in beiden Richtungen benutzt werden. Knoten werden bereits beim Einreihen markiert; dadurch wird ein Knoten niemals doppelt in die Warteschlange aufgenommen. In der folgenden Tabelle steht das nächste zu entnehmende Element jeweils links.

Schritt	Entnommener Knoten	Neu eingereihte Nachbarn, alphabetisch	Warteschlange danach
Start	-	E	[E]
1	E	D, F	[D, F]
2	D	B	[F, B]
3	F	C	[B, C]
4	B	A	[C, A]
5	C	keine	[A]
6	A	keine	[]

Die Besuchsreihenfolge lautet E, D, F, B, C, A. Die Entfernungsebenen sind $\{E\}$, $\{D, F\}$, $\{B, C\}$, $\{A\}$.