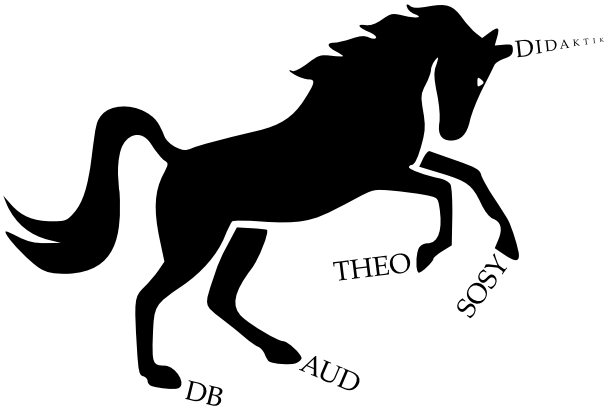
66115
Herbst
2019

Theoretische Informatik / Algorithmen (vertieft)

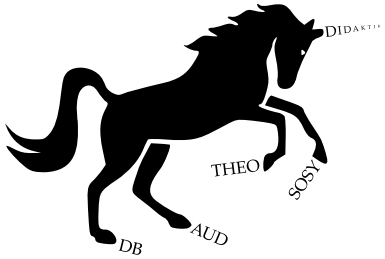
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 1

Aufgabe 4 [SAT DOPPELSAT]

Betrachten Sie die folgenden Probleme:

SAT
Gegeben: Aussagenlogische Formel F in KNF
Frage: Gibt es mindestens eine erfüllende Belegung für F?

DOPPELSAT
Gegeben: Aussagenlogische Formel F′ in KNF
Frage: Gibt es mindestens eine erfüllende Belegung für F, in der mindestens zwei Literale pro Klausel wahr sind?

(a) Führen Sie eine polynomielle Reduktion von SAT auf DOPPELSAT durch.

Lösungsvorschlag

https://courses.cs.washington.edu/courses/csep531/09wi/handouts/sol4.pdf

DOUBLE-SAT is in NP. The polynomial size certificate consists of two assignments f1 and f2. First, the verifier verifies if f1 6= f2. Then, it verifies if both assignments satisfy φ by subtituting the values for the variables and evaluate the clauses of φ. Both checks can be done in linear time. DOUBLE-SAT is NP-hard. We give a reduction from SAT. Given an instance φ of SAT which is a CNF formula of n variables x1,x2,...xn, we construct a new variable xn+1 and let $\psi = \varphi \wedge (\text{xn+1} \vee \neg \text{xn+1})$ be the corresponding instance of DOUBLE-SAT. We claim that φ has a satisfying assignment iff ψ has at least two satisfying assignments. On one hand, if φ has a satisfying assignment f, we can obtain two distinct satisfying assignments of ψ by extending f with xn+1 = T and xn+1 = F respectively. On the other hand, if ψ has at least two sastisyfing assignments then the restriction of any of them to the set x1,x2,...xn is a satisfying assignment for φ. Thus, DOUBLE-SAT is NP-complete.

https://cs.stackexchange.com/questions/6371/proving-double-sat-is-np-complete

Here is one solution:

Clearly Double-SAT belongs to NP, since a NTM can decide Double-SAT as follows: On a Boolean input formula $\phi(x_1, \ldots, x_n)$, nondeterministically guess 2 assignments and verify whether both satisfy ϕ.

To show that Double-SAT is NP-Complete, we give a reduction from SAT to Double-SAT, as follows:

On input $\phi(x_1, \ldots, x_n)$:

1. Introduce a new variable y. 2. Output formula $\phi'(x_1, \ldots, x_n, y) = \phi(x_1, \ldots, x_n) \wedge (y \vee \bar{y})$.

If $\phi(x_1, \ldots, x_n)$ belongs to SAT, then ϕ has at least 1 satisfying assignment, and therefore $\phi'(x_1, \ldots, x_n, y)$ has at least 2 satisfying assignments as we can satisfy the new clause $(y \vee \bar{y})$ by assigning either $y = 1$ or $y = 0$ to the new variable y, so $\phi'(x_1, \ldots, x_n, y) \in$ Double-SAT.

On the other hand, if $\phi(x_1, \ldots, x_n) \notin$ SAT, then clearly $\phi'(x_1, \ldots, x_n, y) = \phi(x_1, \ldots, x_n) \wedge (y \vee \bar{y})$ has no satisfying assignment either, so $\phi'(x_1, \ldots, x_n, y) \notin$ Double-SAT.

Therefore, SAT $\leq_p$ Double-SAT, and hence Double-SAT is NP-Complete.

(b) Zeigen Sie, dass DOPPELSAT NP-vollständig ist.

Aufgabe 5 [Notation des Informatik-Duden]

In der folgenden Aufgabe soll ein Feld A von ganzen Zahlen aufsteigend sortiert werden. Das Feld habe n Elemente A[0] bis A[n-1]. Der folgende Algorithmus (in der Notation des Informatik-Duden) sei gegeben:

```
procedure quicksort(links, rechts : integer)
var i, j, x : integer;
begin
  i := links;
  j := rechts;
  if j > i then begin
    x := A[links];
    repeat
      while A[i] < x do i := i+1;
      while A[j] > x do j := j-1;
      if i < j then begin
        tmp := A[i]; A[i] := A[j]; A[j] := tmp;
        i := i+1; j := j-1;:
      end
    until i > j;
    quicksort(links, j);
    quicksort(i, rechts);
  end
end
```

Umsetzung in Java:

```
public static void quicksort(int[] A, int links, int rechts) {
  System.out.println("quick");
  int i = links;
  int j = rechts;
  if (j > i) {
    int x = A[links];
    do {
      while (A[i] < x) {
        i = i + 1;
      }
```

```
      while (A[j] > x) {
        j = j - 1;
      }
      if (i <= j) {
        int tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;
        i = i + 1;
        j = j - 1;
      }
      // Java verfügt über keine do-until Schleife.
      // Wir verwenden eine do-while-Schelfe mit einem umgedrehten Test
      // unit i > j -> while (i <= j)
    } while (i <= j);
    quicksort(A, links, j);
    quicksort(A, i, rechts);
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/herbst/QuickSort.java`

Der initiale Aufruf der Prozedur lautet:

```
quicksort(0,n-1)
```

(a) Sortieren Sie das folgende Feld der Länge 7 mittels des Algorithmus. Notieren Sie jeweils alle Aufrufe der Prozedur quicksort mit den konkreten Parameterwerten. Geben Sie zudem für jeden Aufruf der Prozedur den Wert des in Zeile 7 gewählten Elements an.

27 13 21 3 6 17 44 42

Lösungsvorschlag

```
quicksort(0, 6)
27 32 3 6 17 44 42
x: 27
quicksort(0, 2)
17 6 3 32 27 44 42
x: 17
quicksort(0, 1)
3 6 17 32 27 44 42
x: 3
quicksort(0, -1)
3 6 17 32 27 44 42
quicksort(1, 1)
3 6 17 32 27 44 42
quicksort(2, 2)
3 6 17 32 27 44 42
quicksort(3, 6)
3 6 17 32 27 44 42
x: 32
quicksort(3, 3)
3 6 17 27 32 44 42
quicksort(4, 6)
3 6 17 27 32 44 42
x: 32
quicksort(4, 3)
```

```
3 6 17 27 32 44 42
quicksort(5, 6)
3 6 17 27 32 44 42
x: 44
quicksort(5, 5)
3 6 17 27 32 42 44
quicksort(6, 6)
3 6 17 27 32 42 44
3 6 17 27 32 42 44
```

(b) Angenommen, die Bedingung $j > i$ in Zeile 6 des Algorithmus wird ersetzt durch die Bedingung $j \geq i$. Ist der Algorithmus weiterhin korrekt? Begründen Sie Ihre Antwort.

Lösungsvorschlag

geht, dauert aber länger

```
quicksort(0, 6)
27 32 3 6 17 44 42
x: 27
quicksort(0, 2)
17 6 3 32 27 44 42
x: 17
quicksort(0, 1)
3 6 17 32 27 44 42
x: 3
quicksort(0, -1)
3 6 17 32 27 44 42
quicksort(1, 1)
3 6 17 32 27 44 42
x: 6
quicksort(1, 0)
3 6 17 32 27 44 42
quicksort(2, 1)
3 6 17 32 27 44 42
quicksort(2, 2)
3 6 17 32 27 44 42
x: 17
quicksort(2, 1)
3 6 17 32 27 44 42
quicksort(3, 2)
3 6 17 32 27 44 42
quicksort(3, 6)
3 6 17 32 27 44 42
x: 32
quicksort(3, 3)
3 6 17 27 32 44 42
x: 27
quicksort(3, 2)
3 6 17 27 32 44 42
quicksort(4, 3)
3 6 17 27 32 44 42
quicksort(4, 6)
3 6 17 27 32 44 42
x: 32
```

```
quicksort(4, 3)
3 6 17 27 32 44 42
quicksort(5, 6)
3 6 17 27 32 44 42
x: 44
quicksort(5, 5)
3 6 17 27 32 42 44
x: 42
quicksort(5, 4)
3 6 17 27 32 42 44
quicksort(6, 5)
3 6 17 27 32 42 44
quicksort(6, 6)
3 6 17 27 32 42 44
x: 44
quicksort(6, 5)
3 6 17 27 32 42 44
quicksort(7, 6)
3 6 17 27 32 42 44
3 6 17 27 32 42 44
```

(c) Angenommen, die Bedingung $i \leq j$ in Zeile 11 des Algorithmus wird ersetzt durch die Bedingung $i < j$. Ist der Algorithmus weiterhin korrekt? Begründen Sie Ihre Antwort.

Lösungsvorschlag

bleibt hängen

```
quicksort(0, 6)
27 32 3 6 17 44 42
x: 27
quicksort(0, 2)
17 6 3 32 27 44 42
x: 17
quicksort(0, 1)
3 6 17 32 27 44 42
x: 3
```

(d) Wie muss das Feld A gestaltet sein, damit der Algorithmus mit der geringsten Anzahl von Schritten terminiert? Betrachten Sie dazu vor allem Zeile 7. Begründen Sie Ihre Antwort und geben Sie ein Beispiel.

Lösungsvorschlag

Im Worst Case (schlechtesten Fall) wird das Pivotelement stets so gewählt, dass es das größte oder das kleinste Element der Liste ist. Dies ist etwa der Fall, wenn als Pivotelement stets das Element am Ende der Liste gewählt wird und die zu sortierende Liste bereits sortiert vorliegt. Die zu untersuchende Liste wird dann in jedem Rekursionsschritt nur um eins kleiner und die Zeitkomplexität wird beschrieben durch $\mathcal{O}(n^2)$. Die Anzahl der Vergleiche ist in diesem Fall $\frac{n\cdot(n+1)}{2} - 1 = \frac{n^2}{2} + \frac{n}{2} - 1$. Die Länge der jeweils längeren Teilliste beim rekursiven Aufrufe ist nämlich im Schnitt $\frac{2}{n} \sum_{i=\frac{n}{2}}^{n-1} i = \frac{3}{4}n - \frac{2}{4}$ und die Tiefe der Rekursion damit in $\mathcal{O}(\log(n))$. Im Avera-

ge Case ist die Anzahl der Vergleiche etwa $2 \cdot \log(2) \cdot (n+1) \cdot \log_2(n) \approx 1,39 \cdot (n+1) \cdot \log_2(n)$.

(e) Die rekursiven Aufrufe in den Zeilen 16 und 17 des Algorithmus werden zur Laufzeit des Computers auf dem Stack verwaltet. Die Anzahl der Aufrufe von quicksort auf dem Stack abhängig von der Eingabegröße n sei mit s(n) bezeichnet. Geben Sie die Komplexitätsklasse von s(n) für den schlimmsten möglichen Fall an. Begründen Sie Ihre Antwort.

Aufgabe 6 [Sortieren von O-Klassen]

(a) Sortieren Sie die unten angegebenen Funktionen der O-Klassen $\mathcal{O}(a(n))$, $\mathcal{O}(b(n))$, $\mathcal{O}(c(n))$, $\mathcal{O}(d(n))$ und $\mathcal{O}(e(n))$ bezüglich ihrer Teilmengenbeziehungen. Nutzen Sie ausschließlich die echte Teilmenge $\subset$ sowie die Gleichheit $=$ für die Beziehung zwischen den Mengen. Folgendes Beispiel illustriert diese Schreibweise für einige Funktionen f_1 bis f_5 (diese haben nichts mit den unten angegebenen Funktionen zu tun): [1]

$$\mathcal{O}(f_4(n)) \subset \mathcal{O}(f_3(n)) = \mathcal{O}(f_5(n)) \subset \mathcal{O}(f_1(n)) = \mathcal{O}(f_2(n))$$

Die angegebenen Beziehungen müssen weder bewiesen noch begründet werden.

- $a(n) = n^2 \cdot \log_2(n) + 42$
- $b(n) = 2^n + n^4$
- $c(n) = 2^{2 \cdot n}$
- $d(n) = 2^{n+3}$
- $e(n) = \sqrt{n^5}$

Lösungsvorschlag

$$\begin{aligned} a(n) &= n^2 \cdot \log_2(n) + 42 && = n \\ b(n) &= 2^n + n^4 && = 2^n \\ c(n) &= 2^{2 \cdot n} && = 2^{2 \cdot n} \\ d(n) &= 2^{n+3} && = 2^n \\ e(n) &= \sqrt{n^5} \end{aligned}$$

$$\mathcal{O}(a(n)) \subset \mathcal{O}(e(n)) \subset \mathcal{O}(b(n)) = \mathcal{O}(d(n)) \subset \mathcal{O}(c(n))$$

[1] http://www.s-inf.de/Skripte/DaStru.2012-SS-Katoen.(KK).Klausur1MitLoesung.pdf

$$\mathcal{O}(n^2 \cdot \log_2(n) + 42) \subset \mathcal{O}(\sqrt{n^5}) \subset \mathcal{O}(2^n + n^4) = \mathcal{O}(2^{n+3}) \subset \mathcal{O}(2^{2 \cdot n})$$

(b) Beweisen Sie die folgenden Aussagen formal nach den Definitionen der O-Notation oder widerlegen Sie sie.

(i) $\mathcal{O}(n \cdot \log_2 n) \subseteq \mathcal{O}(n \cdot (\log_2 n)^2)$

Lösungsvorschlag

Die Aussage gilt. Für $n \geq 16$ haben wir

$$(\log_2 n)^2 \leq n \Leftrightarrow \log_2 n \leq \sqrt{n}$$

und dies ist eine wahre Aussage für $n \geq 16$. Also gilt die Aussage mit $n_0 = 16$ und $c = 1$.

(ii) $2^{(n+1)} \in \mathcal{O}(n \cdot \log_2 n)$

(c) Bestimmen Sie eine asymptotische Lösung (in Θ-Schreibweise) für die folgende Rekursionsgleichung:

(i) $T(n) = 4 \cdot T(\frac{n}{2}) + n^2$

(ii) $T(n) = T(\frac{n}{2}) + \frac{n}{2}n^2 + n$

Aufgabe 7 [Binärer Baum mit Methode foo]

Gegeben sei die folgende Realisierung von binären Bäumen (in einer an Java angelehnten Notation):

```
class Node {
  Node left, right;
  int value;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/herbst/Node.java`

(a) Beschreiben Sie in möglichst wenigen Worten, was die folgende Methode `foo` auf einem nicht-leeren binären Baum berechnet.

```
int foo(Node node) {
  int b = node.value;
  if (b < 0) {
    b = -1 * b;
  }
  if (node.left != null) {
    b = b + foo(node.left);
  }
  if (node.right != null) {
    b = b + foo(node.right);
  }
  return b;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/herbst/Node.java`

Lösungsvorschlag

> Die Methode `foo(Node node)` berechnet die Summe aller Knoten des Unterbaums des als Parameter übergebenen Knotens `node`. Der Schüsselwert des Knotens `node` selbst wird in die Summenberechnung mit einbezogen.

(b) Die Laufzeit der Methode `foo(tree)` ist linear in n, der Anzahl von Knoten im übergebenen Baum `tree`. Begründen Sie kurz, warum `foo(tree)` eine lineare Laufzeit hat.

Lösungsvorschlag

> Die Methode `foo(Node node)` wird pro Knoten des Baums `tree` genau einmal aufgerufen. Es handelt sich um eine rekursive Methode, die auf den linken und rechten Kindknoten aufgerufen wird. Hat ein Knoten keine Kinder mehr, dann wir die Methode nicht aufgerufen.

(c) Betrachten Sie den folgenden Algorithmus für nicht-leere, binäre Bäume. Beschreiben Sie in möglichst wenigen Worten die Wirkung der Methode `magic(tree)`. Welche Rolle spielt dabei die Methode `max`?

```
void magic(Node node) {
  Node m = max(node);
  if (m.value > node.value) {
    // Werte von m und node vertauschen
    int tmp = m.value;
    m.value = node.value;
    node.value = tmp;
  }
  if (node.left != null)
    magic(node.left);
  if (node.right != null)
    magic(node.right);
}

Node max(Node node) {
  Node max = node;
  if (node.left != null) {
    Node tmp = max(node.left);
    if (tmp.value > max.value)
      max = tmp;
  }
  if (node.right != null) {
    Node tmp = max(node.right);
    if (tmp.value > max.value)
      max = tmp;
  }
  return max;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2019/herbst/Node.java`

Methode `magic(tree)` vertauscht für jeden Knoten des Unterbaums den Wert mit dem Maximal-Knoten. Die Methode `max` liefert dabei den Knoten mit der größten Schlüsselwert im Unterbaum des übergebenen Knoten (sich selbst eingeschlossen).

(d) Geben Sie in Abhängigkeit von n, der Anzahl von Knoten im übergebenen Baum `tree`, jeweils eine Rekursionsgleichung für die asymptotische Best-Case-Laufzeit $(B(n))$ und Worst-Case-Laufzeit $(W(n))$ des Aufrufs `magic(tree)` sowie die entsprechende Komplexitätsklasse (Θ) an. Begründen Sie Ihre Antwort.

Hinweis: Überlegen Sie, ob die Struktur des übergebenen Baumes Einfluss auf die Laufzeit hat. Die lineare Laufzeit von `max(t)` in der Anzahl der Knoten des Baumes `t` darf vorausgesetzt werden.

Thema Nr. 2

Aufgabe 1 [Multiplikation mit 3]

Gesucht ist eine Turing-Maschine mit genau einem beidseitig unendlichen Band, die die Funktion $f : \mathbb{N} \to \mathbb{N}$ mit $f(x) = 3x$ berechnet. Zu Beginn der Berechnungsteht die Eingabe binär codiert auf dem Band, wobei der Kopf auf die linkeste Ziffer (most significant bit) zeigt. Am Ende der Berechnung soll der Funktionswert binär codiert auf dem Band stehen, wobei der Kopf auf ein beliebiges Feld zeigen darf.

(a) Beschreiben Sie zunächst in Worten die Arbeitsweise Ihrer Maschine.

Lösungsvorschlag

$13 \cdot 3 = 0b1101 \cdot 0b11 = 39 = 0b100111$:

$$\begin{array}{cccccc} & 1 & 1 & 0 & 1 & \\ & & 1 & 1 & 0 & 1 \\ {\scriptstyle 1} & {\scriptstyle 1} & & & & \\ \hline 1 & 0 & 0 & 1 & 1 & 1 \end{array}$$

Die entworfene Turningmaschine imitierte der Vorgehensweise beim schriftlichen Multiplizieren. Die Maschine geht zunächst an das Leerzeichen am rechten Ende des Eingabewortes. Die Maschine bewegt sich nun zwei Schritte nach links und liest die Zahlen ein und addiert sie. Schließlich bewegt sich die Maschine einen Schritt nach rechts und schreibt das Ergebnis der Additium. Dabei wird das Eingabewort überschrieben allmählich überschrieben.

(b) Geben Sie dann das kommentierte Programm der Turing-Maschine an und erklären Sie die Bedeutung der verwendeten Zustände.

Lösungsvorschlag

z_0 An das rechte Ende des Eingabewortes gehen

z_1 Übertrag 0

z_2 Übertrag 1

z_3 1. Additionsschritt: $+0$

z_4 1. Additionsschritt: $+1$

z_5 1. Additionsschritt: $+2$

z_6 2. Additionsschritt: $+0$

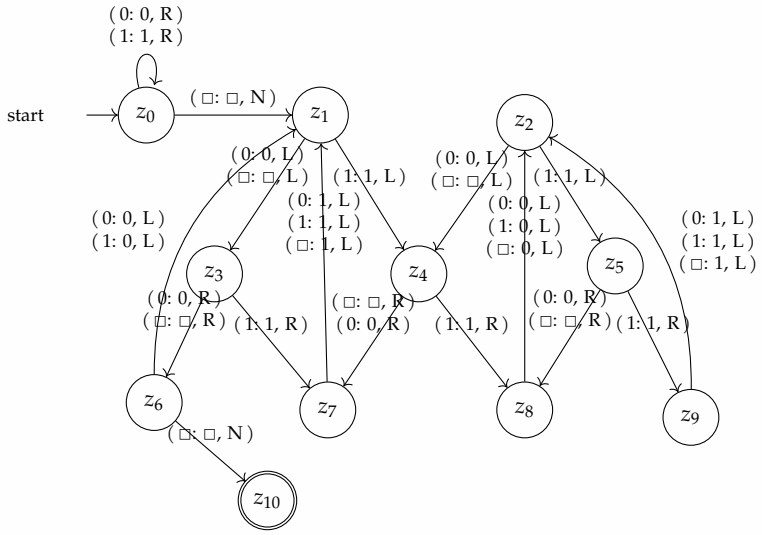
z_7 2. Additionsschritt: $+1$

z_8 2. Additionsschritt: $+2$

z_9 2. Additionsschritt: $+3$

z_{10} Endzustand

Nicht sehr übersichtlich hier: Im Anhang findet sich die JSON-Datei für flaci.com



Der Automat auf flaci.com (FLACI: Formale Sprachen, abstrakte Automaten, Compiler und Interpreter) Ein Projekt der Hochschule Zittau/Görlitz und der Pädagogischen Hochschule Schwyz: flaci.com/Ahjkw50kg

Aufgabe 6 [Mastertheorem]

Der Hauptsatz der Laufzeitfunktionen ist bekanntlich folgendermaßen definiert:

1. Fall: $T(n) \in \Theta\left(n^{\log_b a}\right)$

$$\text{falls } f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right) \text{für } \varepsilon > 0$$

2. Fall: $T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$

$$\text{falls } f(n) \in \Theta\left(n^{\log_b a}\right)$$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega\left(n^{\log_b a+\varepsilon}\right)$ für $\varepsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

Bestimmen und begründen Sie formal mit Hilfe dieses Satzes welche Komplexität folgende Laufzeitfunktionen haben.

(a) $T(n) = 8 \cdot T\left(\frac{n}{2}\right) + 5n^2$

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$

Anzahl der rekursiven Aufrufe (a):

8

Anteil Verkleinerung des Problems (b):

um $\frac{1}{2}$ also $b = 2$

Laufzeit der rekursiven Funktion ($f(n)$):

$5n^2$

Ergibt folgende Rekursionsgleichung:

$T(n) = 8 \cdot T\left(\frac{n}{2}\right) + 5n^2$

1. Fall: $f(n) \in \mathcal{O}\left(n^{\log_b a-\varepsilon}\right)$:

für $\varepsilon = 4$:

$f(n) = 5n^2 \in \mathcal{O}\left(n^{\log_2 8-4}\right) = \mathcal{O}\left(n^{\log_2 4}\right) = \mathcal{O}\left(n^2\right)$

2. Fall: $f(n) \in \Theta\left(n^{\log_b a}\right)$:

$f(n) = 5n^2 \notin \Theta\left(n^{\log_2 8}\right) = \Theta\left(n^3\right)$

3. Fall: $f(n) \in \Omega\left(n^{\log_b a+\varepsilon}\right)$:

$f(n) = 5n^2 \notin \mathcal{O}\left(n^{\log_2 8+\varepsilon}\right)$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

(b) $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + 5n^2$

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$

Anzahl der rekursiven Aufrufe (a):

9

Anteil Verkleinerung des Problems (b):

um $\frac{1}{3}$ also $b = 3$

Laufzeit der rekursiven Funktion ($f(n)$):

$5n^2$

Ergibt folgende Rekursionsgleichung:

$$T(n) = 9 \cdot T\left(\frac{n}{3}\right) + 5n^2$$

1. Fall: $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$:

$f(n) = 5n^2 \notin \mathcal{O}\left(n^{\log_3 9 - \varepsilon}\right)$ für $\varepsilon > 0$

2. Fall: $f(n) \in \Theta\left(n^{\log_b a}\right)$:

$f(n) = 5n^2 \in \Theta\left(n^{\log_3 9}\right) = \Theta\left(n^2\right)$

3. Fall: $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$:

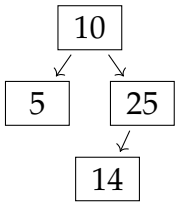
$f(n) = 5n^2 \notin \mathcal{O}\left(n^{\log_3 9 + \varepsilon}\right)$ für $\varepsilon > 0$

$\Rightarrow T(n) \in \Theta\left(n^2 \cdot \log n\right)$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

Aufgabe 7 [AVL-Baum 10,5,25,14 erweitern und Knoten entfernen]

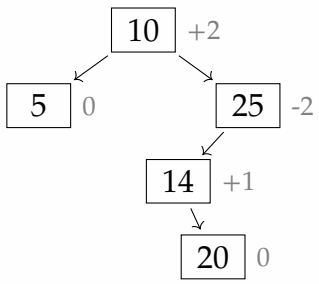
Fügen Sie (manuell) nacheinander die Zahlen $20, 31, 2, 17, 7$ in folgenden AVL-Baum ein. Löschen Sie anschließend aus dem entstandenen Baum nacheinander 14 und 25.



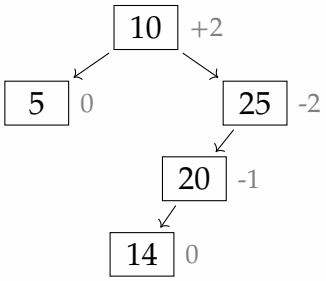
Zeichnen Sie jeweils direkt nach jeder einzelnen Operation zum Einfügen oder Löschen eines Knotens, sowie nach jeder elementaren Rotation den entstehenden Baum. Insbesondere sind evtl. anfallende Doppelrotationen in zwei Schritten darzustellen. Geben Sie zudem an jedem Knoten die Balancewerte an.

Lösungsvorschlag

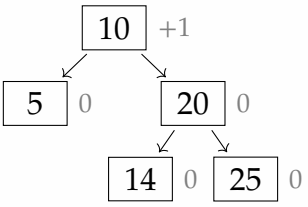
Nach dem Einfügen von „20“:



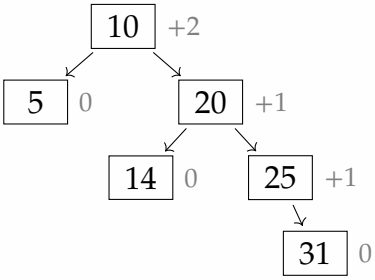
Nach der Linksrotation:



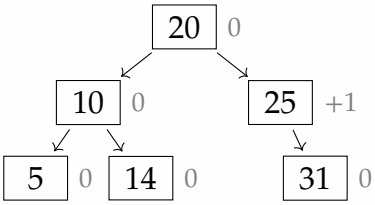
Nach der Rechtsrotation:



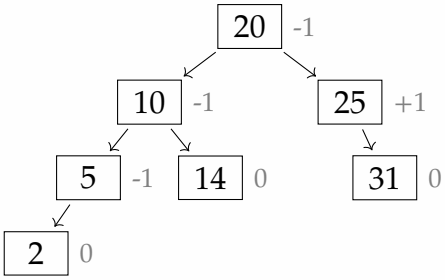
Nach dem Einfügen von „31“:



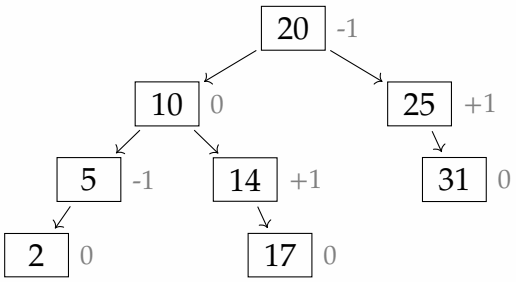
Nach der Linksrotation:



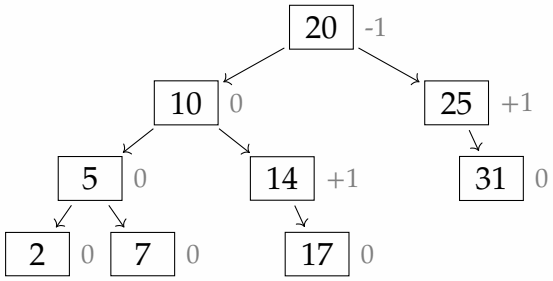
Nach dem Einfügen von „2“:



Nach dem Einfügen von „17“:

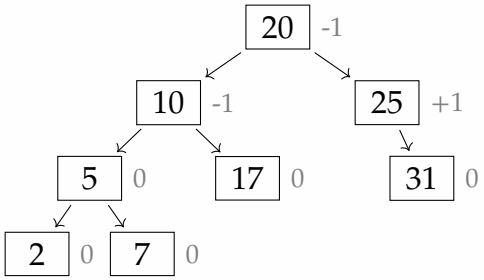


Nach dem Einfügen von „7“:

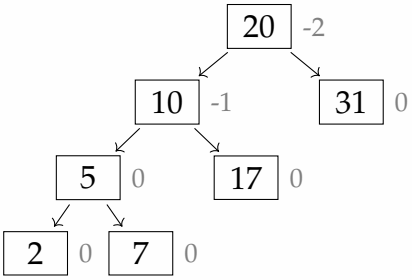


Löschen

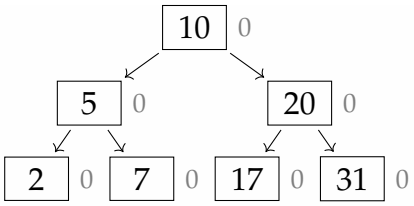
Nach dem Löschen von „14“:



Nach dem Löschen von „25“:

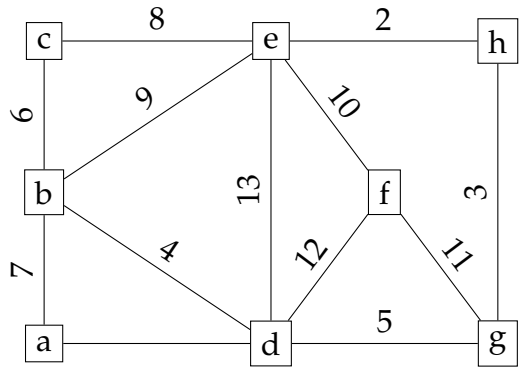


Nach der Rechtsrotation:



Aufgabe 8 [Graph a-h]

Gegeben Sei der folgende ungerichtete Graph mit Kantengewichten.



(a) Zeichnen Sie den (hier eindeutigen) minimalen Spannbaum.

(b) Geben Sie sowohl für den Algorithmus von Jarník-Prim als auch für den Algorithmus von Kruskal die Reihenfolge an, in der die Kanten hinzugefügt werden. Starten Sie für den Algorithmus von Jarník-Prim beim Knoten a.

Übernehmen Sie den Graph auf Ihre Bearbeitung und füllen Sie hierzu das Tupel jeder Kante - aus dem MST in der Form (n, m) aus, wobei die Kante e vom Algorithmus von Jarník-Prim als n'te Kante und vom Algorithmus von Kruskal als m'te Kante hinzugefügt wird. Lassen Sie andere Tupel unausgefüllt.

Aufgabe 9 (Hashing) [Hashing mit mod 11 und 13]

Verwenden Sie die Hashfunktion $h(k, i) = (h'(k) + i^2) \mod 11$ mit $h'(k) = k \mod 13$, um die Werte 12, 29 und 17 in die folgende Hashtabelle einzufügen. Geben Sie zudem jeweils an, auf welche Zellen der Hashtabelle zugegriffen wird.

0	1	2	3	4	5	6	7	8	9	10
			16		5				22	

Lösungsvorschlag

Einfügen des Wertes 12

$h'(12) = 12 \mod 13 = 12$
$h(12, 0) = 12 + 0^2 \mod 11 = 1$

0	1	2	3	4	5	6	7	8	9	10
	12		16		5				22	

Einfügen des Wertes 29

$h'(29) = 29 \mod 13 = 3$

$h(29,0) = 3 + 0^2 \mod 11 = 3$ (belegt von 16)

$h(29,1) = 3 + 1^2 \mod 11 = 4$

0	1	2	3	4	5	6	7	8	9	10
	12		16	29	5				22	

Einfügen des Wertes 17

$h'(17) = 17 \mod 13 = 4$

$h(17,0) = 4 + 0^2 \mod 11 = 4$ (belegt von 29)

$h(17,1) = 4 + 1^2 \mod 11 = 5$ (belegt von 5)

$h(17,2) = 4 + 2^2 \mod 11 = 8$

0	1	2	3	4	5	6	7	8	9	10
	12		16	29	5			17	22	