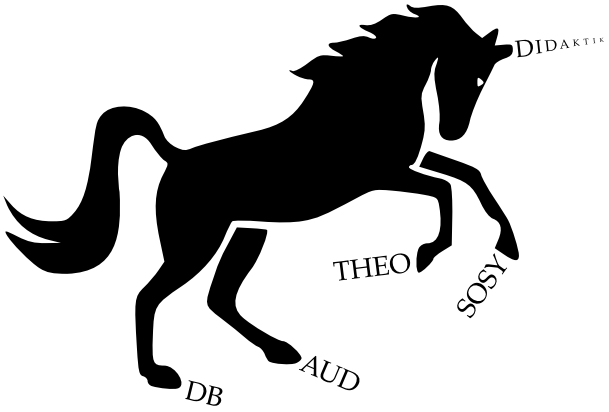
66115
Herbst
2018

Theoretische Informatik / Algorithmen (vertieft)

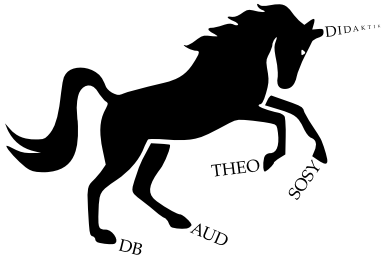
Aufgabenstellungen mit Lösungsvorschlägen



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 3 [Kontextfreie Sprachen]

(a) Entwerfen Sie eine kontextfreie Grammatik für die folgende kontextfreie Sprache über dem Alphabet $\Sigma = \{a, b, c\}$:

$$L = \{\, wb^{3k}c^{2k+1}v \mid k \in \mathbb{N}, |w|_c = |u|_a \,\}$$

(Hierbei bezeichnet $|u|$, die Anzahl des Zeichens x in dem Wort u, und es gilt $0 \in \mathbb{N}$.) Erklären Sie den Zweck der einzelnen Nichtterminale (Variablen) und der Grammatikregeln Ihrer Grammatik.

(b) Betrachten Sie die folgende kontextfreie Grammatik

$$G = (\{S, X, Y, Z\}, \{z, y\}, P, S)$$

mit den Produktionen

$$\mathrm{P} = \Big\{ \quad S \to ZX \mid y \quad X \to ZS \mid SS \mid x \quad Y \to SX \mid YZ \quad Z \to XX \mid XS \quad \Big\}$$

Benutzen Sie den Algorithmus von Cocke-Younger-Kasami (CYK) um zu zeigen, dass das Wort *xxxyx* zu der von G erzeugten Sprache $L(G)$ gehört.

Lösungsvorschlag

x	x	x	y	x
X	X	X	S	X
Z	Z	Z	Y	
S	X	S		
Z,X	Z			
X,S,Z				

$\Rightarrow xxxyx \in L(G)$

(c) Geben Sie eine Ableitung des Wortes *xxxyx* mit *G* an.

Aufgabe 5 [Binärer Suchbaum]

Hinweis: Wir betrachten in dieser Aufgabe binäre Suchbäume, bei denen jeder innere Knoten genau zwei Kinder hat. Schlüssel werden nur in den inneren Knoten gespeichert - die Blätter speichern keinerlei Informationen.

(a) Welche Eigenschaften muss ein binärer Suchbaum haben, damit er ein AVL-Baum ist?

Lösungsvorschlag

Er muss die zusätzliche Eigenschaft haben, dass sich an jedem Knoten die Höhe der beiden Teilbäume um höchstens eins unterscheidet

(b) Mit $n(h)$ bezeichnen wir die minimale Anzahl innerer Knoten eines AVL-Baums der Höhe h.

(i) Begründen Sie, dass $n(1) = 1$ und $n(2) = 2$.

(ii) Begründen Sie, dass $n(h) = 1 + n(h-1) + n(h-2)$.

(iii) Folgern Sie, dass $n(h) > 2^{\frac{h}{2}-1}$.

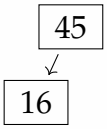
(c) Warum ist die Höhe jedes AVL-Baums mit n inneren Knoten $O(\log n)$?

(d) Fügen Sie die Elemente $(45, 16, 79, 31, 51, 87, 49, 61)$ in der angegebenen Reihenfolge in einen anfangs leeren binären Suchbaum ein (ohne Rebalancierungen). Zeichnen Sie den resultierenden Suchbaum nach jeder Einfügeoperation.

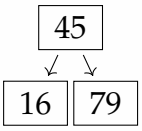
Lösungsvorschlag

Einfügen von „45“:

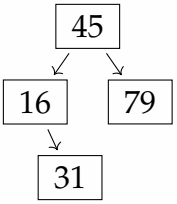
Einfügen von „16“:



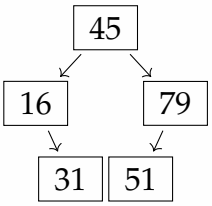
Einfügen von „79“:



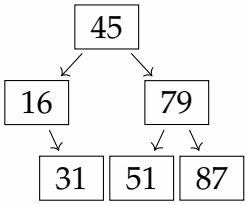
Einfügen von „31“:



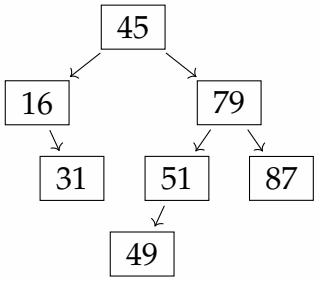
Einfügen von „51“:



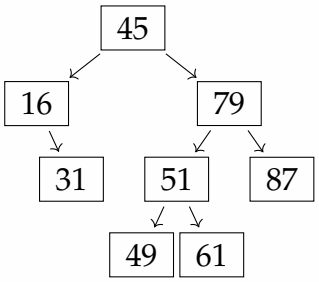
Einfügen von „87“:



Einfügen von „49“:



Einfügen von „61“:



(e) Ist der resultierende Suchbaum aus Teilaufgabe 5.4 ein AVL-Baum? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Ja, wie in der untenstehenden Grafik zu sehen ist, unterscheiden sich die Höhe der Teilbäume von allen Knoten nur um höchstens eins.

(f) Das Einfügen in einen AVL-Baum funktioniert (zunächst) wie beim binären Suchbaum durch Erweitern eines äußeren Knotens w:

vor dem Einfügen von 54 nach dem Einfügen von 54

Anschließend wird die AVL-Baum Eingenschaft (falls notwending) durch eine (Doppel-)Rotation wiederhergestellt: Wenn z der erste Knoten auf dem Pfad P von w zur Wurzel ist, der nicht balanciert ist, y das Kind von z auf P und r das Kind von y auf P, und wenn (a, b,c) die Inorder-Reihenfolge von x, y, 2 ist, dann führen wir die Rotation aus, die benötigt wird, um b zum obersten Knoten der drei zu machen.

Die folgende Illustration zeigt den Fall, dass key(y) < key(x) < key(z), ð(a,b,c) = (y,x,z), wobei w ein Knoten in Ty ist.

Sei Ah die Höhe des Teilbaums 73. Für i = 0,1,2 sei h; die Höhe des Teilbaums T; und für v = 2,y,z sei h, die Höhe des Teilbaums mit der Wurzel v vor der Restrukturierung. Begründen Sie, dass

(i) $h_0 = h$
(ii) $h_1 = h - 1$
(iii) $h_2 = h$
(iv) $h_x = h + 1$
(v) $h_y = h + 2$
(vi) $h_z = h + 3$

(g) Welche Höhe haben die Teilbäume mit den Wurzeln x, y, z nach der Restrukturierung? Begründen Sie Ihre Antworten.

(h) Begründen Sie, dass die oben gezeigte Doppelrotation die AVL-Baum-Eigenschaft wiederherstellt.

(i) Beschreiben Sie, wie ein binärer Baum der Höhe h in einem Array repräsentiert werden kann. Wie viel Speicherplatz ist für so eine Darstellung erforderlich?

(j) Warum verwendet man bei der Implementierung von AVL-Bäumen eine verzeigerte Struktur und nicht eine Array-basierte Repräsentation?

Thema Nr. 2

Aufgabe 6 [Rucksackproblem]

Ein sehr bekanntes Optimierungsproblem ist das sogenannte Rucksackproblem: Gegeben ist ein Rucksack mit der Tragfähigkeit B. Weiterhin ist eine endliche Menge von Gegenständen mit Werten und Gewichten gegeben. Nun soll eine Teilmenge der Gegenstände so ausgewählt werden, dass ihr Gesamtwert maximal ist, aber ihr Gesamtgewicht die Tragfähigkeit des Rucksacks nicht überschreitet.

Mathematisch exakt kann das Rucksackproblem wie folgt formuliert werden:

Gegeben ist eine endliche Menge von Objekten U. Durch eine Gewichtsfunktion $w\colon U \to \mathbb{R}^+$ wird den Objekten ein Gewicht und durch eine Nutzenfunktion $v\colon U \to \mathbb{R}^+$ ein festgelegter Nutzwert zugeordnet.

Des Weiteren gibt es eine vorgegebene Gewichtsschranke $B \in \mathbb{R}^+$. Gesucht ist eine Teilmenge $K \subseteq U$, die die Bedingung $\sum_{u \in K} w(u) \leq B$ einhält und die Zielfunktion $\sum_{u \in K} v(u)$ maximiert.

Das Rucksackproblem ist NP-vollständig (Problemgröße ist die Anzahl der Objekte), sodass es an dieser Stelle wenig Sinn macht, über eine effiziente Lösung nachzudenken. Lösen Sie das Rucksackproblem daher mittels Backtracking und formulieren Sie einen entsprechenden Algorithmus. Gehen Sie davon aus, dass die Gewichtsschranke B sowie die Anzahl an Objekten N beliebig, aber fest vorgegeben sind.

Das Programm soll folgende Ausgaben liefern:

(a) Maximaler Nutzwert, der durch eine Objektauswahl unter Einhaltung der Gewichtsschranke B erreicht werden kann.

(b) Das durch die maximierende Objektmenge erreichte Gesamtgewicht.

(c) Diejenigen Objekte (Objektnummern) aus U, die zur Maximierung des Nutzwerts beigetragen haben.

```java
/**
 * https://stackoverflow.com/a/14186622
 */
public class Rucksack {
  // static int[] werte = new int[] { 894, 260, 392, 281, 27 };
  // static int[] gewichte = new int[] { 8, 6, 4, 0, 21 };
  // static int[] werte = new int[] { 4, 2, 10, 1, 2 };
  // static int[] gewichte = new int[] { 12, 1, 4, 1, 2 };
  static int werte[] = new int[] { 60, 100, 120 };
  static int gewichte[] = new int[] { 10, 20, 30 };



  /**
   * Gewichtsschranke
   */
  //static int B = 30;
```

```java
  //static int B = 15;
  static int B = 50;

  /**
   * Diejenigen Objekte aus U, die zur Maximierung des Nutzwerts beigetragen
   * haben.
   */
  static boolean[] auswahl = new boolean[werte.length];

  private static int berechne(int i, int W) {
    if (i < 0) {
      return 0;
    }
    int alt = berechne(i - 1, W);
    if (gewichte[i] > W) {
      // Backtracking!
      auswahl[i] = false;
      return alt;
    } else {

      int neu = berechne(i - 1, W - gewichte[i]) + werte[i];
      if (alt >= neu) {
        // Backtracking!
        auswahl[i] = false;
        return alt;
      } else {
        auswahl[i] = true;
        return neu;
      }
    }
  }

  static void werteAus() {
    System.out.println(berechne(werte.length - 1, B));

    int gesamtGewicht = 0;
    int gesamtWert = 0;

    for (int i = 0; i < auswahl.length; i++) {
      if (auswahl[i]) {
        gesamtGewicht += gewichte[i];
        gesamtWert += werte[i];
        System.out.println("Objekt-Nr. " + i + " Gewicht: " + gewichte[i] + " Wert: " +
↪   werte[i]);
      }
    }

    System.out.println("Gesamtgewicht: " + gesamtGewicht);
    System.out.println("Gesamtwert: " + gesamtWert);
  }

  public static void main(String[] args) {
    werteAus();
  }
```

```
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2018/herbst/Rucksack.java`

Aufgabe 8 [Sortieren von 15,4,10,7,1,8,10 mit Bubble- und Selectionsort]

Gegeben sei das folgende Feld A mit 7 Schlüsseln:

$$[15, 4, 10, 7, 1, 8, 10]$$

(a) Sortieren Sie das Feld mittels des Sortierverfahrens *Bubblesort*. Markieren Sie jeweils, welche zwei Feldwerte verglichen werden und geben Sie den Zustand des gesamten Feldes jeweils neu an, wenn Sie eine Vertauschung durchgeführt haben.

Lösungsvorschlag

```
 15  4   10  7   1   8   10  Eingabe
 15  4   10  7   1   8   10  Durchlauf Nr. 1
>15  4<  10  7   1   8   10  vertausche (i 0<>1)
 4  >15  10< 7   1   8   10  vertausche (i 1<>2)
 4   10 >15  7<  1   8   10  vertausche (i 2<>3)
 4   10  7  >15  1<  8   10  vertausche (i 3<>4)
 4   10  7   1  >15  8<  10  vertausche (i 4<>5)
 4   10  7   1   8  >15  10< vertausche (i 5<>6)
 4   10  7   1   8   10  15  Durchlauf Nr. 2
 4  >10  7<  1   8   10  15  vertausche (i 1<>2)
 4   7  >10  1<  8   10  15  vertausche (i 2<>3)
 4   7   1  >10  8<  10  15  vertausche (i 3<>4)
 4   7   1   8   10  10  15  Durchlauf Nr. 3
 4  >7   1<  8   10  10  15  vertausche (i 1<>2)
 4   1   7   8   10  10  15  Durchlauf Nr. 4
>4   1<  7   8   10  10  15  vertausche (i 0<>1)
 1   4   7   8   10  10  15  Durchlauf Nr. 5
 1   4   7   8   10  10  15  Ausgabe
```

(b) Sortieren Sie das Feld mittels des Sortierverfahrens *Selectionsort*. Markieren Sie jeweils, welche zwei Feldwerte verglichen werden und geben Sie den Zustand des gesamten Feldes jeweils neu an, wenn Sie eine Vertauschung durchgeführt haben.

```
 15  4   10  7   1   8   10  Eingabe
 15  4   10  7   1   8   10* markiere (i 6)
>15  4   10  7   1   8   10< vertausche (i 0<>6)
 10  4   10  7   1   8*  15  markiere (i 5)
>10  4   10  7   1   8<  15  vertausche (i 0<>5)
 8   4   10  7   1*  10  15  markiere (i 4)
 8   4  >10  7   1<  10  15  vertausche (i 2<>4)
 8   4   1   7*  10  10  15  markiere (i 3)
>8   4   1   7<  10  10  15  vertausche (i 0<>3)
 7   4   1*  8   10  10  15  markiere (i 2)
```

```
>7   4   1<  8   10  10  15  vertausche (i 0<>2)
 1   4*  7   8   10  10  15  markiere (i 1)
 1  >4   7   8   10  10  15  vertausche (i 1<>1)
 1*  4   7   8   10  10  15  markiere (i 0)
>1   4   7   8   10  10  15  vertausche (i 0<>0)
 1   4   7   8   10  10  15  Ausgabe
```

(c) Vergleichen Sie beide Sortierverfahren hinsichtlich ihres Laufzeitverhaltens im *best case*. Welches Verfahren ist in dieser Hinsicht besser, wenn das zu sortierende Feld anfangs bereits sortiert ist? Begründen Sie Ihre Antwort.

Der Bubblesort-Algorithmus hat im *best case* eine Laufzeit von $\mathcal{O}(n)$, der Selectionsort-Algorithmus $\mathcal{O}(n^2)$.

Bubblesort steuert seine äußere bedingte Wiederholung in vielen Implementationen über eine boolsche Hilfvariable `getauscht`, die beim Betreten der Schleife erstmals auf falsch gesetzt wird. Erst wenn Vertauschungen vergenommen werden müssen, wird diese Varialbe auf wahr gesetzt und die äußere Schleife läuft ein weiteres Mal ab. Ist das zu sortierende Feld bereits sortiert, durchsucht der Algorithmus des Bubblesort das Feld einmal und terminiert dann.

Der Selectionsort-Algorithmus hingegen ist mit zwei ineinander verschränkten Schleifen umgesetzt, deren Wiederholungsanzahl sich starr nach der Anzahl der Elemente im Feld richtet.

Bubblesort

```java
int durchlaufNr = 0;
boolean getauscht;
do {
  durchlaufNr++;
  berichte.feld("Durchlauf Nr. " + durchlaufNr);
  getauscht = false;
  for (int i = 0; i < zahlen.length - 1; i++) {
    if (zahlen[i] > zahlen[i + 1]) {
      // Elemente vertauschen
      vertausche(i, i + 1);
      getauscht = true;
    }
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/sortier/BubbleIterativ.java`

Selectionsort

```java
// Am Anfang ist die Markierung das letzte Element im Zahlen-Array.
int markierung = zahlen.length - 1;
while (markierung >= 0) {
  berichte.feldMarkierung(markierung);
```

```java
// Bestimme das größtes Element.
// max ist der Index des größten Elements.
int max = 0;
// Wir vergleichen zuerst die Zahlen mit der Index-Number
// 0 und 1, dann 1 und 2, etc. bis zur Markierung
for (int i = 1; i <= markierung; i++) {
  if (zahlen[i] > zahlen[max]) {
    max = i;
  }
}

// Tausche zahlen[markierung] mit dem gefundenem Element.
vertausche(markierung, max);
// Die Markierung um eins nach vorne verlegen.
markierung--;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/sortier/SelectionRechtsIterativ.java`