Algorithmen und Programmierung 3, WS 2019/2020 — 13. Übungsblatt

Abgabe bis Freitag, 31. Januar 2020, 12:00 Uhr, in die Fächer der Tutoren

---

81. Das Wörterbuchproblem, 0 Punkte

    Lassen Sie die Datenstrukturen, die Sie für das Wörterbuchproblem kennengelernt haben, Revue passieren.

    Stellen Sie die Vor- und Nachteile gegenüber. Welche Operationen werden von welchen Datenstrukturen effizient unterstützt?

82. Hashing, 0 Punkte

    Welche Möglichkeiten der Kollisionsbehandlung gibt es bei Hashtabellen?

83. 1-IN-3-SAT, 0 Punkte

    Beim 1-IN-3-SAT-Problem ist eine Menge von Klauseln gegeben, die aus je drei Literalen bestehen. Gesucht ist eine Belegung, in der in jeder Klausel *genau ein* Literal wahr ist (im Gegensatz zum gewöhnlichen Erfüllbarkeitsproblem SAT, wo in jeder Klausel *mindestens ein* Literal wahr sein muss). Wir schreiben diese Klauseln in der Form $l_1 + l_2 + l_3 = 1$.

    Zeigen Sie, dass dieses Problem NP-schwer ist, indem Sie 3SAT darauf reduzieren. Die Reduktion ersetzt jede 3SAT-Klausel $l_1 \vee l_2 \vee l_3$ durch 3 Klauseln

    $$\begin{aligned} \bar{l}_1 + b_1 + c_1 &= 1 \\ \bar{l}_2 + b_2 + c_2 &= 1 \\ c_1 + c_2 + l_3 &= 1 \end{aligned}$$

    mit neuen Variablen b_1, c_1, b_2, c_2, die sonst nirgends vorkommen. (3SAT ist der Spezialfall von SAT, in dem jede Klausel drei Literale enthält.)

    Liegt das Proplem 1-IN-3-SAT in NP?

84. Unabhängige Knotenmenge, 5 Punkte

    Gegeben ist ein ungerichteter Graph G. Gesucht ist eine möglichst große Teilmenge von Knoten, in der keine zwei Knoten benachbart sind. Zeigen Sie dass dieses Problem NP-schwer ist, indem Sie 3-Färbbarkeit darauf reduzieren.

    Anleitung: Nehmen Sie drei disjunkte Kopien (eine „rote“, eine „grüne“, und eine „blaue“) des Graphen, dessen 3-Färbbarkeit getestet werden soll, und verbinden je drei entsprechende Knoten in den verschiedenen Kopien durch drei Kanten, die ein Dreieck bilden.

85. Huffman-Bäume, 0 Punkte

    (a) Bestimmen Sie den Huffman-Baum für die Häufigkeiten 1300, 1900, 2000, 2400, 3000, 3300, 3400, 3900, 4900, 7000, 7200, 9900.

        Zeigen Sie in einzelnen Schritten, wie der Algorithmus abläuft, und zeichnen Sie den optimalen CodeBaum.

    (b) Ist der optimale binäre Code für diese Häufigkeiten eindeutig?

86. Huffman-Bäume, 0 Punkte

    Zeigen Sie, dass man einen Huffman-Baum in $O(n)$ Zeit bauen kann, wenn die Häufigkeiten $h_1 \leq \cdots \leq h_n$ in sortierter Reihenfolge gegeben sind.

$z = h(2^{-l_i}) + h(2^{l_i})$ $\quad e^z \geq 1+z$

$e^{h(2^{-l_i})+h(2^{l_i})} \geq 1 + h(2^{-l_i}) + h(2^{l_i})$

87. Entropie, 5 Punkte

Im Huffman-Baum für die Häufigkeiten $p_1, \ldots, p_n$ mit $p_1 + \cdots + p_n = 1$ seien die n Blätter auf Tiefe $l_1, \ldots l_n$. Aus Aufgabe 32b vom 4. Übungsblatt wissen wir, dass $\sum 2^{-l_i} = 1$ ist. Zeigen Sie, dass die mittlere Codewortlänge $\sum p_i l_i$ durch die *Entropie*

$$H(p_1, \ldots, p_n) := p_1 \log_2 \tfrac{1}{p_1} + p_2 \log_2 \tfrac{1}{p_2} + \cdots + p_n \log_2 \tfrac{1}{p_n} \tag{1}$$

der Verteilung $(p_1, \ldots, p_n)$ von unten beschränkt ist.

Anleitung: Leiten Sie aus der Ungleichung $e^z \geq 1 + z$, die für alle $z \in \mathbb{R}$ gilt, durch Umformung die Beziehung $2^{-l_i} \geq 2^{-L_i}(1 - \ln 2 \cdot (l_i - L_i))$ her. Setzen Sie dann $L_i := \log_2 \frac{1}{p_i}$, summieren Sie über i, und verwenden Sie $\sum 2^{-l_i} = 1$ für die linke Seite.

88. Huffman-Bäume, 0 Punkte

Berechnen Sie die Entropie der Verteilung, die sich aus den Häufigkeiten von Aufgabe 85 ergibt. Vergleichen Sie sie mit der mittleren Codewortlänge des Huffman-Codes.

89. Existenz eines guten Codes, 5 freiwillige Zusatzpunkte als Bonus

Für die Folge $l_i := \lceil \log_2 \frac{1}{p_i} \rceil$ gilt $\sum 2^{-l_i} \leq 1$. Wir können nun die Längen l_i schrittweise reduzieren, dass $\sum 2^{-l_i} = 1$ wird. Wegen Aufgabe 32c erhalten wir auf diese Weise einen Code, dessen mittlere Codewortlänge $\sum p_i l_i$ die Entropieschranke (1) um höchstens 1 Bit überschreitet. Führen Sie die Einzelheiten aus.

90. Datenkompression, Programmieraufgabe, 10 Punkte

(a) Implementieren Sie die Huffman-Codierung in PYTHON oder JAVA. Dabei soll eine Textdatei *byteweise* (nicht als *String*) gelesen werden. Zunächst soll in einem Durchlauf die Häufigkeit der Zeichen bestimmt, danach ein Huffman-Baum konstruiert und schließlich die Codierung des Texts ausgegeben werden. Überlegen Sie sich dazu eine geeignete Codierung für den Huffman-Baum. Um das Ende der Datei festzulegen, muss man entweder die Länge der Datei separat übermitteln, oder man muss ein zusätzliches Zeichen EOF in das Eingabealphabet aufnehmen. Sie können selbst entscheiden, ob Sie die Ausgabe auf zwei Dateien aufteilen oder in eine Datei zusammenpacken.

(b) Bestimmen Sie experimentell den Kompressionsfaktor für die Texte *Deutschland, ein Wintermärchen*[1] und *Sternstunden der Menschheit*[2] im Vergleich zum 8-Bit-ISO-Latin1-Code.

(c) Implementieren Sie auch die Decodierung. Die Eingabe besteht den im Schritt (a) produzierten Daten: aus dem Huffman-Code (das heißt, einem beliebigen präfixfreien Codes über dem Alphabet $\{0, 1\}$) und dem codierten Text.

91. Radixsort, 0 Punkte

Die Fachverteilung bei einem Durchlauf von Radixsort kann durch verkettete Listen erfolgen. Eine andere Möglichkeit speichert die n Elemente direkt in ein Zielfeld $z[0 \ldots n-1]$: Zunächst muss man *zählen*, wieviele Elemente für jedes Fach $0, \ldots, B-1$ bestimmt sind. Anschließend bestimmt man den Beginn des Intervalls, das jedes Fach im Feld z einnimmt. Danach kann man die Elemente durchlaufen und direkt an die richtige Stelle in z setzen.

Schreiben Sie ein Programm dafür in Pseudocode.[3],

[1] `http://www.inf.fu-berlin.de/lehre/WS15/ALP3/Deutschland.txt`, 77550 Bytes. Die Originalquelle ist `http://www.gutenberg.org/files/6079/6079-8.txt`, mit einem zusätzlichen Vor- und Nachspann.

[2] `http://www.inf.fu-berlin.de/lehre/WS13/Datenkompression/Daten/buch1`, 420.825 Bytes.

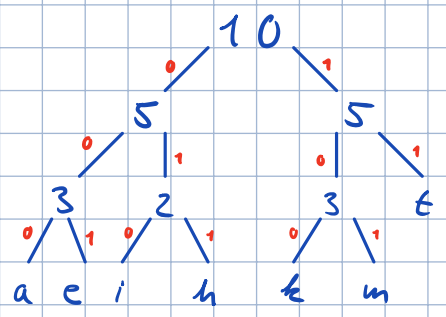
[3] Zur Lösung siehe das Unterprogramm `RadixDurchlauf` im Programm `http://www.inf.fu-berlin.de/lehre/WS13/Datenkompression/Suffixfeld-KS.py`

Ü 30 von Thore Brehmer und David Seg

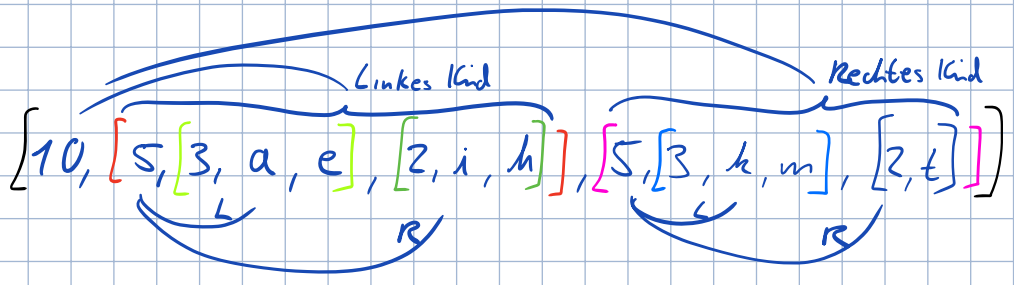
Bsp.

Mathematik

m = 2
a = 2
t = 2
h = 1
e = 1
i = 1
k = 1

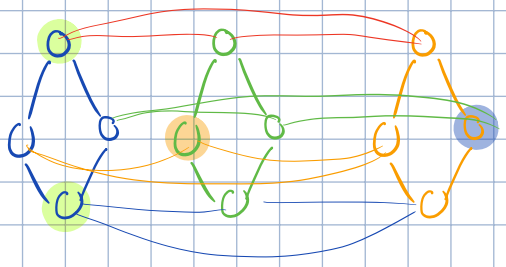


code ⇒




```
('Tree =', [10, [5, [3, 'a', 'e'], [2, 'i', 'h']], [5, [3, 'k', 'm'], [2, 't']]])
('Code =', [1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0])
('String =', ['m', 'a', 't', 'h', 'e', 'm', 'a', 't', 'i', 'k'])
thob97@andorra:~/alp3/u13$
```

84.



ALP3

Dynamisches Programmieren
- Editierabstand

Sortierverfahren
- Radixsort
- Quicksort
- Mergesort
- Bubblesort

P/NP (schwer/voll.)
- Definition (P = Polyn. Zeit lösbar, NP = P.Zeit überprüfbar)
- Reduktion
- 3COL / SAT / Kürzeste einfacher Weg -> Hamilton Kreis
  mächtigste Knotenmenge
- Teilwortsuche: Robin Karb, Boyer-Moore, Knuth Morris Pratt

Graphen
- Topologisches Sortieren
- BFS / DFS
- MST / Kruskal, Boruvka, Prim
- Dijkstra / (Kürzester Weg)
- Dantzig
- UNION FIND
- Backtracking
- Perfekte codes (präfixfrei)
  Quickselect
  Stückweise konstant
  Huffmann Schnittpunkte
- optimale Suchbäume

Datenstrukturen
- Abstraktionsfunktion
- Darstellungsinvariante
- Skipliste
- Bäume: AVL
  AVL, AB, Digitale Suchb.
  suffixb., präfixb.
- Hashtabellen
  Kollisionsbehandlung
  Kuckucksuche, lineares Sondieren
  offene Adressierung
  Optimale Hashfunktion.
  Kryptographische
  verkettete Liste
- Halden <- Henkel

Abstraktedatentypen
- spezifikation
  algebraisch
  beschreibend
- Wörterbuch
  insert(Key)
  delete(Key)
  get(k), size, isEmpty...
- Prio Queue
- Geometrische Alg.
  Konvexe Hülle

Skipliste: i - $O(\log n)$, d - $O(\log n)$, s - $O(\log n)$
Sp. ↳ $O(n)$

AVL Baum: i - $O(\log n)$, d - $O(\log n)$, s - $O(\log n)$
Sp. ↳ $O(n)$