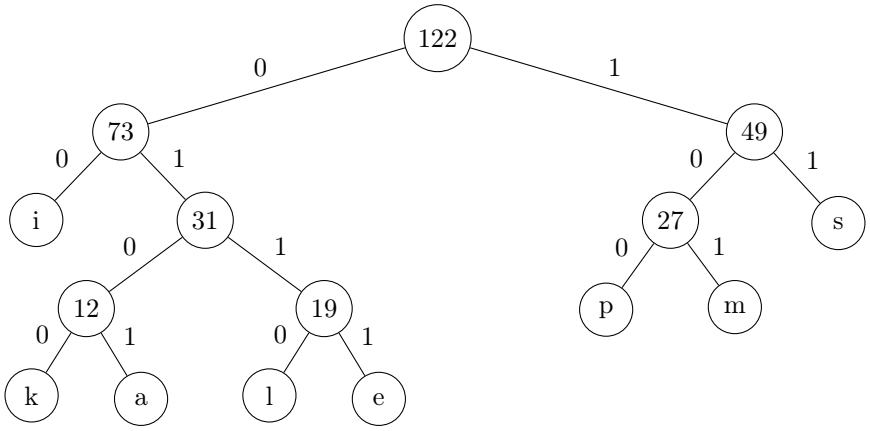
Algorithmen und Datenstrukturen: Übung 7

Tanja Zast, Alexander Waldenmaier

7. Januar 2021

Aufgabe 7.1

a)



b Präfixcode:

k	a	l	e	p	m	s	i
0100	0101	0110	0111	100	101	11	00

`mississippi` = 101 00 11 11 00 11 11 00 100 100 00

c) Die acht Buchstaben können auch in 3er-Blöcken von Binärzahlen dargestellt werden. Da das Wort `mississippi` insgesamt 11 Buchstaben hat, würde diese Art der Kodierung einen Speicherplatz von $3 \cdot 11 = 33$ Bits benötigen. Bei der Kodierung mit dem Huffman-Algorithmus kamen wir auf eine Länge von $3 \cdot 3 + 2 \cdot 8 = 25$ Bits. Es wurde also knapp 24% Speicherplatz eingespart.

Aufgabe 7.2

a) Zu Beginn liegen dem Huffman-Algorithmus nur „ungruppierte“ Buchstaben mit ihren Gewichten vor. Diese werden zunächst sortiert und dann die zwei kleinsten zu einem Zwei zusammengefasst, der nun als Gewicht die Summe der beiden Buchstaben erhält. Die zwei Buchstaben werden nun nicht mehr als einzelne betrachtet, sondern der neu entstandene Zweig wieder in die Liste der anderen Elemente einsortiert.

Beim Einsortieren gibt es genau zwei Möglichkeiten: Entweder der Zweig wird an erster oder zweiter Stelle einsortiert, oder er landet weiter hinten. Im ersten Fall ist der Ausgang der nächsten Zweig-Erstellung logisch: Der alte Zweig wird mit einem weiteren Element kombiniert, wodurch ein neuer Zweig mit höherer Gesamtsumme entsteht. Landet er jedoch (im zweiten Fall) weiter hinten, so könnte man vermuten, dass die nun zwei kleinsten Elemente in Summe kleiner sein können, als der Zweig. Würde man diese zwei kleinsten Elemente nun kombinieren, entstünde ein neuer Zweig der ein kleineres Gewicht hat als unser erster Zweig.

Wir betrachten n Symbole a_i, die ihrer Häufigkeit nach sortiert seien: $a_1 < ... < a_n$. Im ersten Algorithmus-Durchlauf werden die Elemente a_1 und a_2 kombiniert, wodurch der Zweig $b_1 = a_1 + a_2$ entsteht. Dieser wird nun wieder in die Liste einsortiert. Dabei können genau zwei Fälle auftreten:

$$\text{I: } b_1 < a_3 ... < a_n \text{ oder } a_3 < b_1 < a_4 < ... < a_n$$
$$\text{II: } a_3 < a_4 < ... < b_1 < ... a_n$$

Im Fall I landet der Zweig b_1 ganz am Anfang oder an zweiter Stelle und wird folglich im nächsten Durchlauf sofort mit dem Symbol a_3 kombiniert. Für den resultierenden Zweig b_2 gilt, wie gefordert, $b_2 = a_3 + b_1 > b_1$. Im Fall 2 muss gewährleistet sein, dass $b_2 = a_3 + a_4 > b_1$ gilt. Zum Beweis helfen dabei diese Abschätzungen: $b_1 < 2a_2$ (da $a_1 < a_2$) und $b_2 > 2a_3$ (da $a_4 > a_3$). In die Bedingung eingesetzt ergibt sich:

$$b_2 \overset{!}{>} b_1$$
$$b_2 > 2a_3 \overset{!}{>} 2a_2 > b_1$$
$$a_3 > a_2 \quad \checkmark$$

Ein neu erstellter Zweig ist also stets größer, als ein zuvor erstellter Zweig.

b) Zur Umsetzung betrachten wir zwei Queues, A und B die jeweils mit Knoten-Objekten gefüllt werden können. Diese Knoten-Objekte besitzen als Attribute ihr Gewicht, sowie Pointer zu ihren Kind-Knoten. A sei zu Beginn gefüllt mit den bereits ihrer Reihe nach sortierten Knoten-Objekten der Symbole. Die Kind-Knoten all dieser Objekte sind Null-Pointer.

Nun betrachte man in jedem Schleifendurchlauf jeweils bis zu zwei erste (kleinste) Objekte der Queues A und B und suche aus diesen bis zu vier Knoten-Objekten die zwei mit dem geringsten Gewicht. Diese zwei entferne man aus den Queues, füge sie zu einem gemeinsamen neuen Knoten-Objekt als Kinder hinzu und reihe diesen neuen Knoten am Ende von B ein. Die Ordnung von B wird dabei nicht gestört, da jeder neue Knoten größer als jeder der zuvor erstellten ist (wie in a) bewiesen).

Dieses Vorgehen wird so lange fortgeführt, bis irgendwann in Queue A keine Knoten mehr enthalten sind und in Queue B nur noch ein Knoten enthalten ist. Dieser ist dann der Wurzelknoten.

Aufgabe 7.3

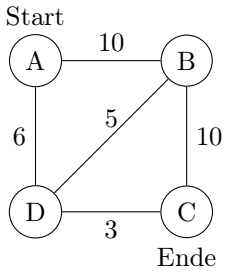
i	A	B	C	D	E	F
0	**0**	∞	∞	∞	∞	∞
1	0	**2**	∞	7	∞	∞
2	0	2	7	5	**3**	∞
3	0	2	7	**5**	3	12
4	0	2	**6**	5	3	12
5	0	2	6	5	3	**12**

Tabelle 1: Ausführung des Dijkstra-Algorithmus. Jede Zeile entspricht einem „Sprung“ zu einem weiteren Knoten. Der Algorithmus entscheidet sich in jeder Zeile für den fett markierten Knoten.

Kürzeste Wegstrecken: $\overline{\text{AB}} = 2, \overline{\text{ABEDC}} = 6, \overline{\text{ABD}} = 5, \overline{\text{ABE}} = 3, \overline{\text{ABEF}} = 12$.

Aufgabe 7.4

Der nebenstehende Graph zeigt ein Gegenbeispiel für den vorgeschlagenen Algorithmus. Der längste Weg von A nach C wäre $\overline{\text{ADBC}} = 21$, jedoch findet der Algorithmus den Weg $\overline{\text{ABC}} = 20$.



i	A	B	C	D
0	**0**	$-\infty$	$-\infty$	$-\infty$
1	0	**10**	$-\infty$	6
2	0	10	**20**	15
3	0	10	20	**23**

Aufgabe 7.5

Abgabe in DOMjudge. Teamname: „test“