Lösungsskizzen zur Prüfung

(1)

(a) Siehe Skriptum Seite 8.

(b) $O(1) = O(4) \in O(ln\ n) = O(log_5\ n) \in O(4n) \in O(n^{ln\ 7}) = O(7^{ln\ n}) \in \in O(n^4) \in O(4^n) \in O(n!) \in O(n^n)$

Beweise der Äquivalenzen:

(1) $\underline{1} = \underline{c_1 \cdot 4}$

(2) $\underline{ln\ n} = ln(5^{log_5 n}) = log_5 n \cdot ln\ 5 = \underline{c_1 \cdot log_5 n}$

(3) $\underline{n^{ln\ 7}} = e^{ln(n^{ln\ 7})} = e^{ln\ 7 \cdot ln\ n} = e^{ln\ n \cdot ln\ 7} = e^{ln(7^{ln\ n})} = \underline{7^{ln\ n}}$

(2) Siehe Skriptum Seiten 35-37.

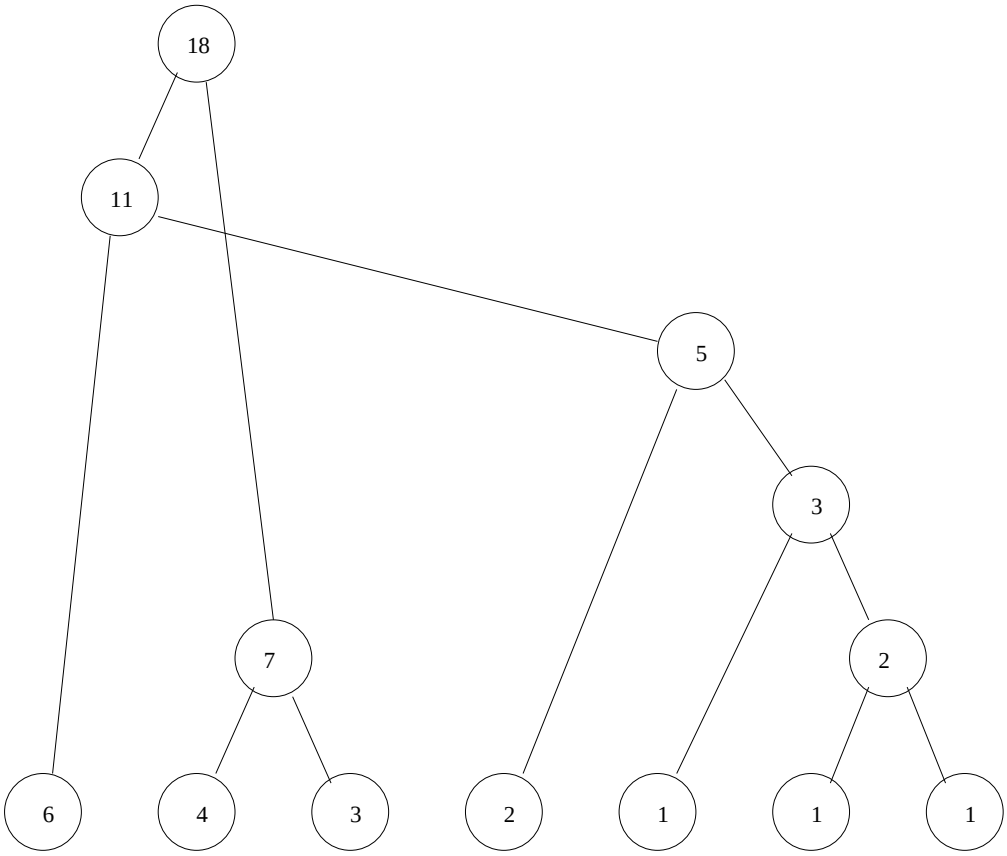
(3) Es kann z.B. ein Heap mit Grösse k als Datenstruktur verwendet werden. Dabei ist von Vorteil, dass das Verhalden eines Elementes in $O(log\ k)$ Zeit funktioniert und dass das grösste Element immer an erster Stelle steht. Das bedeutet, man kann in $O(1)$ Zeit das grösste der gespeicherten Elemente finden. Der Algorithmus kann damit einfach wie folgt aussehen:

* Initialisiere den Heap mit den ersten k Zeichen die online eingelesen werden. Der Aufbau dieses Heap geschieht in $O(k)$ Zeit.
* Für jedes weiter Element prüfe, ob es grösser als die Wurzel ist ($O(1)$). Wenn nein: Ersetze Wurzel durch neues Zeichen und Verhalde dieses ($O(log\ k)$ Zeit). Wenn ja: verwerfe das neue Zeichen (es ist grösser als die bisherigen k kleinsten).

Die benötigte Gesamtzeit dieses Algorithmus ist $O(n\ log\ k)$ (es wird maximal n mal ein Zeichen in $O(log\ k)$ Zeit verhaldet). Der Speicherbedarf ist $O(k)$, denn es ist maximal konstanter grosser Zusatzspeicher (etwa zum Zwischenspeichern eines Zeichens) in Verwendung.

(4) Für den ersten Teil, siehe Skriptum Seiten 96-101.

Da die optimale Codierung anhand von gegebenen Häufigkeiten nicht eindeutig sein muß, sieht ein möglicher Codebaum wie folgt aus:



Dieser Codebaum kann in eine 'schöne', kreuzungsfrei gezeichnete Form gebracht werden, ist aber zu zwecken der Nachvollziehbarkeit der Entstehung so belassen worden.

Aus diesem Codebaum kann nun z.B. abgelesen werden, daß die Anzahl der zur Codierung verwendeten Symbole '18' war (bei Verwendung von absoluten Häufigkeiten steht dies in der Wurzel des Codebaumes), daß mit Huffman-Codierung zur Übertragung des zugrundeliegenden Textes 46 Bit notwendig sind (Summe der inneren Knoten: $2 + 3 + 5 + 7 + 11 + 18 = 46$, entspricht Berechnung mit Huffman-Formel). Zu diesem Ergebnis kommt man auch, wenn man die Summe der absoluten Häufigkeiten mal der Länge des Weges von der Wurzel bis zum Symbol berechnet: $2 \cdot 6 + 2 \cdot 4 + 2 \cdot 3 + 3 \cdot 2 + 4 \cdot 1 + 5 \cdot 1 + 5 \cdot 1 = 46$.

Belegt man linke und rechte Äste jeweils mit Codierungssymbolen, kann man für jedes zu codierende Zeichen einen möglichen Code ablesen. Man sieht auch, daß der Baum alle Eigenschaften eines optimalen Codebaumes besitzt. Der enstehende Code ist mit Sicherheit präfixfrei und hat eine variable Länge (ist kein Blockcode).