

ARBEITSGRUPPE KRYPTOGRAPHIE UND KOMPLEXITÄTSTHEORIE  
Prof. Dr. Marc Fischlin  
Dr. Christian Janson  
Patrick Harasser  
Felix Rohrbach

Sommersemester 2019  
Veröffentlicht am: 31.05.2019  
Letzte Änderung am: 11.06.2019  
Abgabe am: 14.06.2019, 12:00 Uhr

## P1 (Gruppendiskussion)

Nehmen Sie sich etwas Zeit, um die folgenden Fachbegriffe in einer Kleingruppe zu besprechen, sodass sie anschließend in der Lage sind, die Begriffe dem Rest der Übungsgruppe zu erklären:

- (a) Splay-Bäume
- (b) Heaps

## P2 (Splay-Bäume)

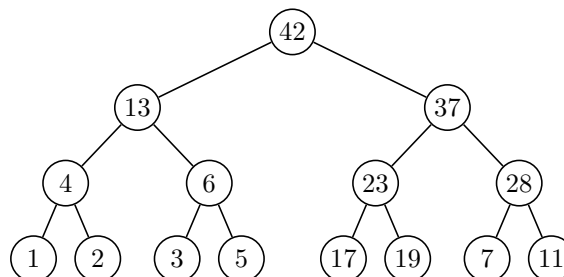
- (a) Fügen Sie der Reihe nach Knoten mit den Schlüsseln 28, 9, 10, 18, 123, 37, 20, 13 in einen leeren Splay-Baum ein. Skizzieren Sie Ihr Zwischenergebnis nach jeder Einfügeoperation.
- (b) Folgende Operationen werden sequentiell hintereinander auf dem Baum  $T$  aus a) ausgeführt:
  - i)  $\text{SEARCH}(T, 37)$
  - ii)  $\text{DELETE}(T, 10)$Zeichnen Sie den entstehenden Baum nach jeder Operation.

## P3 (Heaps)

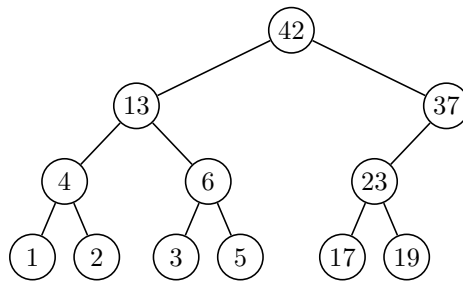
- (a) Geben Sie an, welche der folgenden Bäume und Arrays die Heap-Eigenschaft erfüllen. Begründen Sie bei den Übrigen, was der Heap-Eigenschaft widerspricht.
  - i) 

11	4	23	2	6	17	37	1	3	5	7	13	19	28	42
----	---	----	---	---	----	----	---	---	---	---	----	----	----	----
  - ii) 

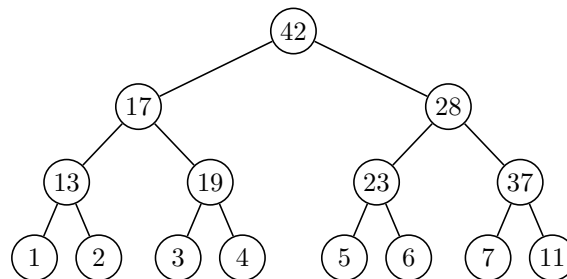
42	37	28	13	19	23	17	11	1	2	3	4	5	6	7
----	----	----	----	----	----	----	----	---	---	---	---	---	---	---
  - iii)



iv)



v)



- (b) Anstelle von  $\text{INSERT}(H, k)$  könnte auch zuerst eine Methode  $\text{APPEND}(H, k)$  und danach  $\text{BUILDHEAP}(H.A)$  aufgerufen werden, wobei  $\text{APPEND}(H, k)$  die Größe des Arrays inkrementiert und den neuen Eintrag an die letzte Position des Arrays anhängt. Welche Nachteile hätte diese Implementierung?
- (c) Erklären Sie, weshalb  $\text{BUILDHEAP}(H.A)$  die Methode  $\text{HEAPIFY}(H.A, i)$  nur für die erste Hälfte (abgerundet) des Arrays durchführt und weshalb dies nicht in aufsteigender Reihenfolge geschehen darf.

---

## P4 (Huffman Codes)

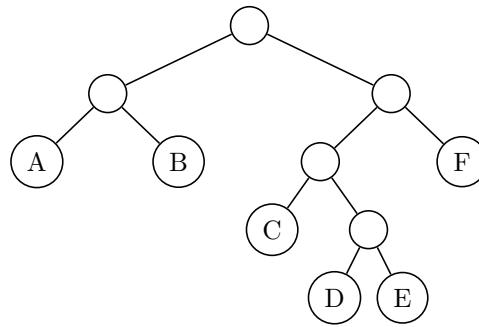
---

Huffman-Codes haben Sie bereits in der letzten Hausübung kennengelernt. Sie berechnen mithilfe eines Greedy Algorithmus ein optimales Encoding basierend auf der Häufigkeit einzelner Buchstaben. Der Algorithmus ist dabei wie folgt:

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n - 1 do
    z = new Node()
    x = EXTRACTMIN(Q)
    y = EXTRACTMIN(Q)
    z.left = x
    z.right = y
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACTMIN(Q)
```

Dabei ist  $C$  das Alphabet und  $Q$  eine Priority-Queue, die immer den Knoten mit der kleinsten Frequenz ausgibt. Über den Baum, den HUFFMAN zurückgibt, können Sie das Codewort für jeden Buchstaben im Alphabet über den Pfad zu diesem Buchstaben im Baum angeben: Für jeden Schritt nach links fügen Sie ein 0-bit dem Codewort hinzu, für jeden Schritt nach rechts ein 1-bit.

Es ist der folgende Huffman-Baum gegeben:



- (a) Geben Sie den Binärcode für folgende Zeichenfolgen an: CBF, DEADBEEF, AABBA.
- (b) Geben Sie die Zeichen an, die durch die folgenden Binärcodes kodiert sind: 10000111011, 0110011101000, 00000000.
- (c) Zeichnen Sie den Baum für den optimalen Huffman-Code für das folgende Alphabet, dessen Frequenzen auf den ersten 8 Fibonacci-Zahlen basiert: A:1 B:1 C:2 D:3 E:5 F:8 G:13 H:21 Wie sieht der Baum für die ersten  $n$  Fibonacci-Zahlen aus?
- (d) Sei das Alphabet nun alle möglichen Bytes (von -128 bis 127). Sie erhalten nun Daten, bei denen die Frequenz des am häufigsten vorkommenden Buchstaben kleiner ist als das doppelte der Frequenz des am seltensten vorkommenden Buchstaben. Zeigen Sie, dass bei diesen Daten die Darstellung durch Huffman-Codes nicht besser ist als die "klassische" Darstellung durch die Bytes.

---

## Hausübungen

---

In diesem Bereich finden Sie die theoretischen Hausübungen von Blatt 7. Bitte beachten Sie die allgemeinen Hinweise zu den Hausübungen und deren Abgabe im Moodle-Kurs! **Denken Sie bitte daran, dass Ihre Lösungen nachvollziehbar und entsprechend ausführlich dargestellt und begründet werden sollen.**

Bitte reichen Sie Ihre Abgabe bis spätestens **Freitag, 14.06.2019, um 12:00 Uhr** ein. Verspätete Abgaben können **nicht** berücksichtigt werden.

---

### H1 (Splay-Bäume)

(3+2 Punkte)

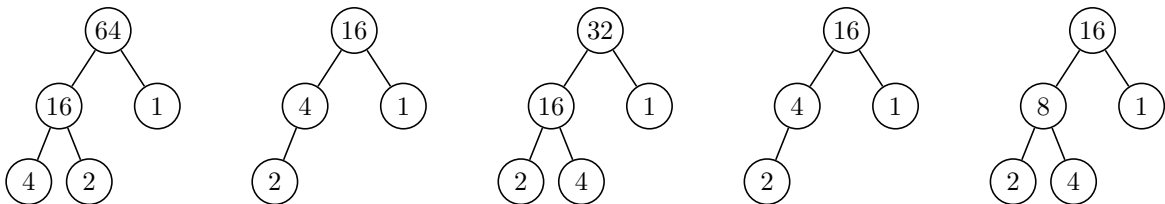
- (a) Fügen Sie der Reihe nach Knoten mit den Schlüsseln 3, 5, 10, 11, 2, 4, 8, 7, 1, 6, 9 in einen leeren Splay-Baum ein. Verwenden Sie dabei die Algorithmen aus der Vorlesung. Skizzieren Sie Ihr Zwischenergebnis nach jeder Einfügeoperation.
- (b) Sei  $T$  der Splay-Baum aus a). Verwenden Sie die Algorithmen aus der Vorlesung, um der Reihe nach folgende Operationen auf  $T$  auszuführen:
- $\text{SEARCH}(T, 5)$ ;
  - $\text{DELETE}(T, 7)$ .
- Skizzieren Sie das Ergebnis jeder Operation.

---

### H2 (Heaps)

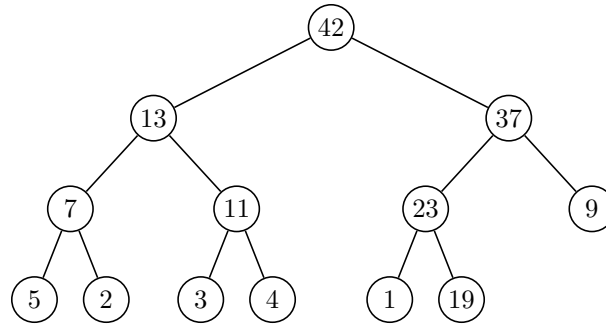
(1.5+1.5+2 Punkte)

- (a) Betrachten Sie das Array  $A = [2, 7, 19, 13, 11]$ . Führen Sie der Reihe nach die folgenden Befehle aus, und stellen Sie den Ergebnisbaum nach jedem Befehl dar. Verwenden Sie dabei die Algorithmen aus der Vorlesung:
- $\text{BUILDHEAP}(H.A)$
  - $\text{EXTRACTMAX}(H)$
  - $\text{INSERT}(H, 37)$
  - $\text{INSERT}(H, 28)$
  - $\text{INSERT}(H, 42)$
  - $\text{HEAPSORT}(H.A)$
- (b) Geben Sie die fünf ausgeführten Befehle zu den folgenden Darstellungen eines Ergebnisbaumes an. Hier ist jede Darstellung das Ergebnis nach dem jeweiligen Befehl (so soll beispielsweise der dritte Befehl vom zweiten zum dritten Baum führen).



- (c) Schreiben Sie eine Methode  $\text{DELETE}(H, i)$ , die den Eintrag an der Stelle  $i$  löscht und die Heap-Eigenschaft aufrecht erhält. Sie dürfen hierfür folgende aus den Vorlesungsfolien bekannte Methoden verwenden (benötigen wahrscheinlich nicht alle hiervon):  $\text{INSERT}$ ,  $\text{EXTRACTMAX}$ ,  $\text{HEAPIFY}$ ,  $\text{SWAP}$ ,  $H.\text{size}$ ,  $i.\text{parent}$ ,  $i.\text{left}$ ,  $i.\text{right}$ ,  $\text{ISEMPTY}$ .

Versuchen Sie, nicht zu viele Elemente zu sortieren, die nicht sortiert werden müssen. Zeigen Sie die von  $\text{DELETE}(H, 4)$  und danach  $\text{DELETE}(H, 1)$  durchgeführten Schritte einmal an folgendem Baum:



---

### H3 (Palindrome)

(3+2 Punkte)

Ein *Palindrom* ist ein Wort  $w$ , welches rückwärts gelesen genau denselben Text ergibt. Beispiele für Palindrome sind ‚regallager‘, ‚reittier‘, ‚neben‘ und ‚rentner‘.

Offensichtlich kann jedes Wort als Sequenz von Palindromen zerlegt werden (indem man einfach die einzelnen Buchstaben als Palindrome auffasst), und diese Zerlegung muss nicht eindeutig sein: Beispielsweise kann ‚ebebbe‘ als ‚e|b|ebbe‘, ‚ebe|bb|e‘, ‚e|beb|b|e‘, oder ‚e|b|e|b|b|e‘ in Palindrome zerlegt werden. Wir bezeichnen die minimale Anzahl von Palindromen, in die ein Wort  $w$  zerlegt werden kann, mit  $p(w)$ .

- Verwenden Sie die Methode der dynamischen Programmierung um einen Algorithmus anzugeben, der für ein Wort  $w$  die Zahl  $p(w)$  in  $O(n^2)$  Schritten berechnet (hier ist  $n$  die Länge von  $w$ ). Die Zerlegung selbst muss nicht ausgegeben werden. Begründen Sie Ihre Angabe.
- Beweisen Sie die Korrektheit und analysieren Sie die Laufzeit Ihres Algorithmus.