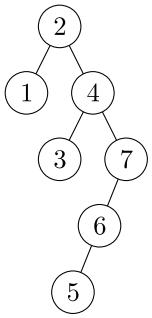
MUSTERLÖSUNG

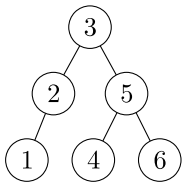
Aufgabe K1 (8 + 6 + 8 + 8 = 30 Punkte)

a) Gegeben sei folgender Splaybaum:



Geben Sie jeweils die resultierenden Splaybäume graphisch an die entstehen, wenn Sie nach dem Verfahren aus der Vorlesung aus diesem Splaybaum zuerst den Schlüssel 2 und aus dem daraus resultierenden Baum den Schlüssel 7 löschen.

b) Wir betrachten nun folgenden Splaybaum:



Geben Sie eine Einfügereihenfolge (ohne Duplikate) der Knoten an, durch die dieser Splaybaum entsteht.

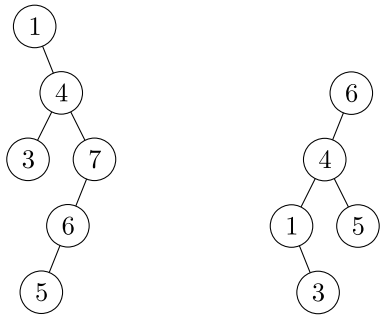
c) Betrachten Sie folgenden Algorithmus.

```
public static int[] sort(int[] input) {
  SplayTree<Integer> tree = new SplayTree<>();
  for(int i = 0; i < input.length; i++) {
    tree.add(input[i]);
  }
  int[] output = new int[input.length];
  for(int i = 0; i < input.length; i++) {
    int min = tree.min();
    tree.remove(min);
    output[i] = min;
  }
  return output;
}
```

Wir benutzen hier einen Splay-Baum, um ein Array zu sortieren. Gehen Sie davon aus, dass die Operation `min` den minimalen Schlüssel ausgibt und amortisiert genau so schnell wie das Suchen in einem Splay-Baum ist. Geben Sie für diesen Algorithmus eine möglichst gute, asymptotische, worst-case Laufzeit an und begründen Sie ihre Antwort. (*Keine Punkte ohne Begründung!*)

d) Geben Sie graphisch einen gerichteten Graphen mit höchstens 5 Knoten an, bei welchem eine Tiefensuche im Tiefensuchbaum eine Querkante erzeugt. Geben Sie den Tiefensuchbaum an und markieren Sie deutlich die Querkante.

Lösungsvorschlag

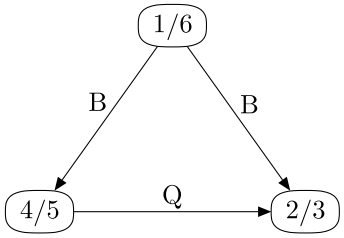


a)

b) Eine mögliche Reihenfolge lautet: 2,1,4,6,5,3. Es gibt noch eine Menge anderer gültiger Einfügereihenfolgen.

c) Sei $n = input.length$. Wir wissen, dass die amortisierte Laufzeit von Splaybäumen im Suchen, Einfügen, Löschen und damit auch Minimum finden $O(\log(n))$ ist. Amortisierte Laufzeit bedeutet aber, dass wenn wir diese Operationen n mal ausführen, wir die echte Laufzeit erhalten. Daher ist die Laufzeit für sowohl die erste Schleife, als auch die zweite Schleife in $O(n \log(n))$. Zusammen ergibt dies eine Gesamtlaufzeit von $O(n \log(n))$.

d)



Aufgabe K2 (23 + 4 + 3 = 30 Punkte)

Gegeben sind folgende Schlüssel mit dazugehörigen Zugriffswahrscheinlichkeiten: A(0.1), E(0.2), I(0.2), O(0.2), U(0.3). Die Schlüssel sind in alphabetischer Reihenfolge aufsteigend geordnet: $A < E < I < O < U$. Konstruieren Sie einen optimalen Suchbaum wie folgt:

a) Füllen Sie untenstehende Tabellen für $w_{i,j}$ und $e_{i,j}$ nach dem Verfahren aus der Vorlesung aus. Geben Sie in $e_{i,j}$ ebenfalls alle möglichen Wurzeln des optimalen Suchbaums für $\{i, \ldots, j\}$ an. Sie dürfen dazu die Notation aus der Übung verwenden.

$w_{i,j}$	A	E	I	O	U
A					
E	—				
I	—	—			
O	—	—	—		
U	—	—	—	—	

$e_{i,j}$	A	E	I	O	U
A	( )	( )	( )	( )	( )
E	—	( )	( )	( )	( )
I	—	—	( )	( )	( )
O	—	—	—	( )	( )
U	—	—	—	—	( )

b) Geben Sie einen optimalen Suchbaum für die Schlüssel A, E, I, O, U mit den gegebenen Zugriffswahrscheinlichkeiten und der gegebenen Reihenfolge der Schlüssel graphisch an.

c) Ist der optimale Suchbaum für die Schlüssel A, E, I, O, U mit den gegebenen Zugriffswahrscheinlichkeiten und der gegebenen Reihenfolge der Schlüssel eindeutig? Geben Sie dazu eine kurze Begründung an.

Sie finden im Anschluss an die Aufgabe weitere Kopien der obigen Tabellen für ihre Notizen.

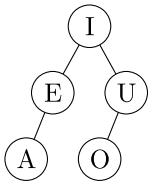
Lösungsvorschlag

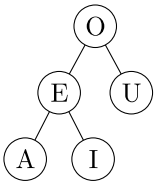
a)

$w_{i,j}$	A	E	I	O	U
A	0.1	0.3	0.5	0.7	1.0
E	0.0	0.2	0.4	0.6	0.9
I	0.0	0.0	0.2	0.4	0.7
O	0.0	0.0	0.0	0.2	0.5
U	0.0	0.0	0.0	0.0	0.3

$e_{i,j}$	A	E	I	O	U
A	0.1 (A)	0.4 (E)	0.8 (E)	1.3 (I)	2.1 (I,O)
E	0.0	0.2 (E)	0.6 (E,I)	1.0 (I)	1.8 (I,O)
I	0.0	0.0	0.2 (I)	0.6 (I,O)	1.2 (O)
O	0.0	0.0	0.0	0.2 (O)	0.7 (U)
U	0.0	0.0	0.0	0.0	0.3 (U)

b) Die beiden folgenden Lösungen sind korrekte optimale Suchbäume.





c) Der optimale Suchbaum ist nicht eindeutig, da bereits die Wurzel des optimalen Suchbaums für {A, . . . , U} nicht eindeutig ist.

Aufgabe K3 (15 + 15 = 30 Punkte)

a) Sei (Q, S) mit $S \subseteq 2^Q$, wobei $Q \subseteq \mathbf{Q}$ eine endliche Menge rationaler Zahlen ist. Sei weiterhin $T \in S$ genau dann, wenn $\sum_{t \in T} t \leq 1$. Beweisen oder widerlegen Sie: Für alle Q gilt: (Q, S) ist ein Matroid.

Kreuzen Sie bitte an: Die Aussage gilt. Die Aussage gilt nicht.

b) Sei $G = (V, E)$ ein ungerichteter Graph. Die unabhängigen Mengen $S \subseteq 2^E$ sind genau die Kantenmengen $T \subseteq E$, deren induzierter Graph maximal einen Kreis enthält. Beweisen oder widerlegen Sie: (E, S) ist ein Matroid.

Kreuzen Sie bitte an: Die Aussage gilt. Die Aussage gilt nicht.

Lösungsvorschlag

a) Nein.

Sei $Q = \{0.4, 0.6, 0.8\}$. Dann ist $\{0.4, 0.6\} \in S$, sowie $\{0.8\} \in S$. Dennoch kann kein Element aus der ersten unabhängigen Menge zur zweiten hinzugefügt werden, sodass die Menge unabhängig bleibt.

b) Ja.

Die Vererbbarkeitseigenschaft ist offensichtlich erfüllt, da beim Entfernen von Kanten keine neuen Kreise entstehen können.

Die Austauscheigenschaft ist ebenfalls erfüllt.

Seien E_1, E_2 unabhängige Mengen mit $|E_1| < |E_2|$. Hat E_1 keinen Kreis, so kann eine beliebige Kante $\{u, v\}$ von $E_2 \setminus E_1$ zu E_1 hinzugefügt werden: Falls $\{u, v\}$ einen Kreis schließt, so schließt es maximal einen Kreis: Der Weg von u nach v in E_1 ist eindeutig (da E_1 kreisfrei ist), dieser Weg und der Kante $\{u, v\}$ wird also der einzige Kreis in $E_1 \cup \{\{u, v\}\}$ sein.

Hat E_1 bereits einen Kreis, so hat (V, E_2) mindestens eine Zusammenhangskomponente weniger als (V, E_1) - selbst wenn (V, E_1) ebenfalls einen (anderen) Kreis hat, muss es mindestens eine Zusammenhangskomponente weniger geben, damit $|E_1| < |E_2|$. Damit gibt es aber auch zwei Knoten u, v, die in (V, E_2) in einer Zusammenhangskomponente sind; in (V, E_1) jedoch nicht. Es gibt nun also mindestens eine Kante auf dem Weg von u nach v in E_2, die zwei Zusammenhangskomponenten in (V, E_1) verbindet - diese Kante kann nun zu E_1 hinzugefügt werden, ohne einen Kreis zu schließen.

(Eine ähnliche Aufgabe wurde bereits in der Globalübung vorgerechnet)

Aufgabe K4 (9 + 21 = 30 Punkte)

a) Gegeben sei folgende Hashfunktion: $x \mapsto x \bmod 100$. Fügen sie folgende Schlüssel der Reihe nach in einen anfangs leeren Treap ein und verwenden Sie die Hashfunktion, um die Priorität jedes Schlüssels zu bestimmen, indem Sie jeweils den Schlüssel selbst hashen:

$142, 194, 430, 1900, -24, 13$

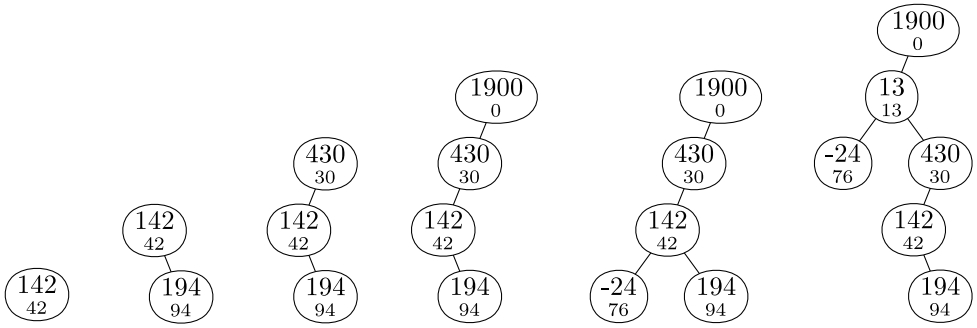
Geben Sie den Treap nach jedem Einfügen graphisch an.

b) Geben Sie einen Algorithmus (in Textform oder ausführlich kommentiertem Pseudocode) an, der eine *split(x)*-Funktionalität implementiert und in $O(h)$, wobei h die Höhe des Treaps ist, läuft. *split(x)* sollte den Treap in zwei Treaps aufteilen, einer der nur Elemente größer x und der andere der nur Elemente kleiner gleich x enthält. Begründen Sie die Korrektheit und die Laufzeit ihres Verfahrens.

Tipp: Wenden Sie ihr Verfahren beispielsweise mit dem Aufruf *split(125)* auf den Treap aus Aufgabenteil a) an, um zu testen, ob ihr Algorithmus so funktioniert, wie Sie es erwarten. (Hinweis: dieser Tipp ist zur eigenen Kontrolle sinnvoll, wird allerdings nicht mit Punkten bedacht)

Lösungsvorschlag

a)



Typischer Fehler: Die Modulooperation von einer negativen Zahl falsch berechnet.

b) Eine mögliche Lösung ist folgende:

1. Zunächst wird das Element x im Treap mittels der Suchbaumeigenschaft gesucht.
2. Falls x vorhanden ist, wird x in die Wurzel rotiert. Ist x nicht vorhanden, so wird der letzte besuchte Knoten der Suche in die Wurzel rotiert. (Die ursprüngliche Priorität von x wird dabei gespeichert.)
3. Der Baum kann nun an der Wurzel geteilt werden: Ist die (neue) Wurzel größer (bzw. kleiner gleich) x, so wird der linke (der rechte) Teilbaum vom Treap gelöst, das linke (bzw. rechte) Kind der Wurzel ist also die Wurzel des zweiten Baums. Ist x die neue Wurzel, so ist egal ob sie dem rechten oder linken Teilbaum zugeordnet wird.
4. Das Element, was im zweiten Schritt in die Wurzel rotiert wurde, wird im entsprechendem Teilbaum wieder an die (bezüglich der Priorität passende Position) herunterrotiert.

Die Laufzeit läuft in $O(h)$, da die ersten beiden Schritte eine Laufzeit von $O(h)$ haben, der dritte eine Laufzeit von $O(1)$, und der letzte wieder eine Laufzeit von $O(h)$.

Typischer Fehler: x muss nicht zwingend Element des Baumes sein. Dies müsste durch den Test auffallen. Ebenfalls ist es möglich, dass x ein Element des Baumes ist.

Ein weiterer Fehler ist das Verwenden der Splay-Operation, da beispielsweise das zigzig die Heap-Eigenschaft kaputt machen kann und entsprechend korrigiert werden müsste.

Ein Ansatz, der grundsätzlich nicht funktioniert, ist das einfache Suchen von x und das einmalige splitten an der Stelle in größer und kleiner: Es ist völlig unklar, was mit allen Knoten passiert die nicht in Teilbäumen an x dranhängen: Diese können sowohl größer, als auch kleiner als x sein, dementsprechend kann es nicht ignoriert werden und komplett als größer bzw. kleiner als x angesehen werden.

Eine alternative korrekte Lösung sucht das Element x im Treap. Findet er auf dem Weg einen Knoten, der größer als x ist, so wird dieser mitsamt dem gesamten rechten Teilbaum in einen zweiten Treap hinzugefügt und im ursprünglichen Treap gelöscht. Kommt man bei x an, so wird der rechte Teilbaum von x noch dem Größer-Treap zugewiesen und im Ursprünglichen Treap gelöscht.

Bei diesem Ansatz ist es die Begründung der Laufzeit nicht trivial: Im Allgemeinen benötigt das Einfügen eines Treaps in einen anderen mehr als $O(1)$ Zeit. Im konkreten Fall kann man sich jedoch die Struktur der abgeschnittenen Teilbäume zunutze machen: Man kommt allein durch das Umsetzen von einem Pointer zu einer Laufzeit von $O(1)$ und behält gleichzeitig die Treap-Struktur.

Analog ist dies natürlich auch mit einem neuen Treap für kleine Elemente (anstatt von größeren), oder mit zwei neuen Treaps möglich.

Die folgenden zwei Treaps sollten nach dem Aufruf von *split(125)* erscheinen.