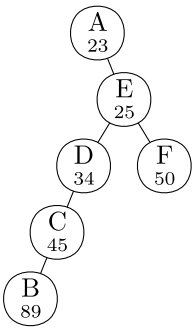
RWTH Aachen
Lehrgebiet Theoretische Informatik
Rossmanith–Hensel–Kuinke–Sánchez

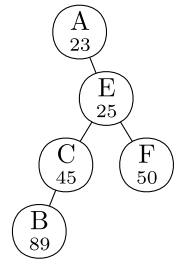
SS 2016
1. Klausur (Gruppe B)
9. August 2016

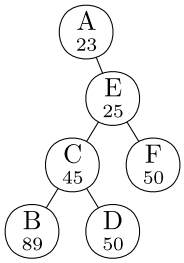
Übung zur Vorlesung Datenstrukturen und Algorithmen

Lösung Aufgabe 1

Diese Treaps sind alle eindeutig (es gibt also nur einen einzigen Suchbaum der ein Heap in Bezug auf die Prioritäten ist).

a)


b)


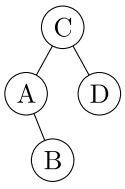
c)


Lösung Aufgabe 2

Tabellen:

```
W[][]
 0.3 0.4 0.6 1.0
 0.0 0.1 0.3 0.7
 0.0 0.0 0.2 0.6
 0.0 0.0 0.0 0.4
E[][]
 0.3 0.5 1.0 1.9
 0.0 0.1 0.4 1.1
 0.0 0.0 0.2 0.8
 0.0 0.0 0.0 0.4
```

Resultierender Baum:



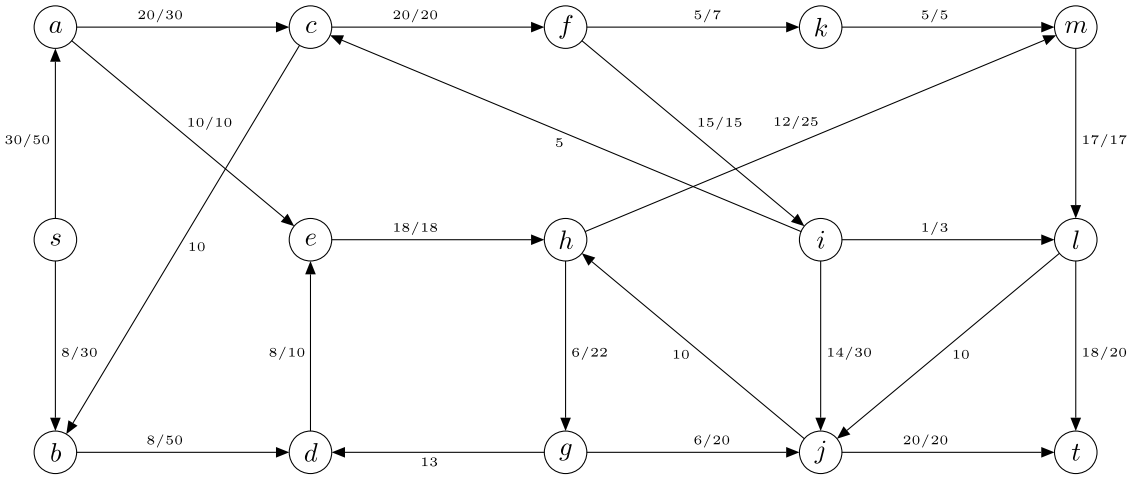
Eine erfolgreiche Suche hat 1.9 Vergleiche (siehe hierzu e_{AD}).

Eine Suche nach dem Schlüssel F braucht zwei Vergleiche (mit B und D). Ein Vergleich mit "`null`" ist nicht ein Vergleich zwischen zwei Schlüssel, zählt somit nicht.

Lösung Aufgabe 3

a) 67

b)



c) 38

d) $(\{s, a, b, c, d, e\}, \{f, g, h, i, j, k, l, m, t\})$

e) 38

f) d, e, h, g, j, m, l

Lösung Aufgabe 4

a) Führe eine binäre Suche auf jeder Zeile oder Spalte aus.

b) Eine binäre Suche braucht $O(\log n)$ auf einer Spalte oder Zeile der Länge n. Die Suche wird auf höchstens n Zeilen/Spalten ausgeführt. Somit ist die Gesammtlaufzeit $O(n \log n)$.

c) Wir entscheiden uns für binäre Suche auf den Spalten.

$$\begin{pmatrix} \mathbf{6} & \mathbf{9} & \mathbf{17} & 18 & 22 & 29 & 37 & 38 \\ \mathbf{13} & \mathbf{21} & \mathbf{29} & \mathbf{33} & 35 & 40 & 41 & 48 \\ \mathbf{19} & \mathbf{23} & \mathbf{37} & \mathbf{42} & \mathbf{43} & 44 & 50 & 57 \\ \textcircled{26} & \mathbf{30} & \mathbf{38} & \mathbf{45} & \mathbf{49} & \mathbf{57} & 60 & 65 \\ 34 & 40 & \mathbf{45} & \mathbf{48} & \mathbf{53} & \mathbf{64} & \mathbf{65} & 67 \\ 39 & 42 & 52 & \mathbf{57} & \mathbf{59} & \mathbf{66} & \mathbf{68} & \mathbf{74} \\ 47 & 53 & 55 & 63 & \mathbf{68} & \mathbf{69} & \mathbf{74} & \mathbf{80} \\ 49 & 59 & 66 & 67 & 74 & \mathbf{79} & \mathbf{86} & \mathbf{87} \end{pmatrix}$$

d) Unser Algorithmus hat zwei Phasen:

Die erste Phase ist rekursiv. Wir springen in die mittlere Bandzeile z. Wenn der gesuchte Wert x kleiner als das erste Element z_1 dieser Bandzeile ist, rufen wir diesen Algorithmus auf der Untermatrix, die nur alle Zeilen über z enthält, auf. Wenn x größer als das letzte Element z_b in z ist, dann rufen wir den Algorithmus auf der Untermatrix, die nur alle Zeilen unter z enthält, auf. Falls das gesuchte Element größer als z_1 und kleiner als z_b ist, wechseln wir in die zweite Phase. Wenn wir noch einen Aufruf machen wollen, aber unsere Untermatrix nur eine Zeile hat, dann geben wir `false` zurück.

In der zweiten Phase geben wir `true` zurück, wenn $x = z_1$ oder $x = z_b$. Wenn das nicht der Fall ist, wissen wir, dass keine Spalte links von der Position von z_1 oder rechts von der Position von z_b die gesuchte Zahl x enthalten kann, da alle diese Spalten nur Elemente, die entweder echt kleiner oder echt größer als x sind, enthalten können. Daher kommen nur die restlichen Spalten in Frage, die mit z ein Element gemeinsam haben. Somit gibt es nur noch b Spalten die in Frage kommen. Wir machen jetzt binäre Suche auf jeder dieser Spalten und geben `true` zurück wenn wir die Zahl finden, sonst `false`.

Die erste Phase funktioniert Analog zu binärer Suche. Da es n Zeilen gibt, wird diese Phase höchsten Zeit $O(\log n)$ brauchen. Die zweite Phase wird nur einmal ausgeführt. Da wir nur auf b Spalten der Länge b arbeiten, können wir das Analog zu Aufgabenteil a) in Zeit $O(b \log b)$ durchführen. Somit ist die Gesamtlaufzeit $O(\log n + b \log b)$.

Anmerkungen zu 4: Folgende Lösungen zu 4a) wurden mit 0 Punkten bewertet:

- Lineare Suche mit *early out* oder vergleichbaren Optimierungen sind trotzdem lineare Suche, und haben eine *worst-case* Laufzeit von $\Omega(n^2)$. Außerdem gilt $n^2 - c = \Theta(n^2)$ für alle Konstanten c.
- Rekursive Loesungen, die trotzdem $\Omega(n^2)$ Laufzeit haben.
- Lösungen die, die Matrix nach einer Suche auf der Diagonale in nicht quadratische Matrizen aufteilen und sich dann rekursiv auf diese Teilmatrizen aufrufen. Da diese nicht quadratisch sind, ist jetzt die Diagonale nicht definiert.
- Lösungen die stark falsche Annahmen treffen, die dazu führen, dass falsche Antworten zurückgegeben werden.
- Riesiger unkommentierter Code, wenn unverständlich.
- "Treppenalgorithmen" die oben links mit der Treppe anfangen. so ein Treppenalgorithmus funktioniert nur wenn man oben rechts oder unten links anfängt, oder man erst zu eine dieser Ecken läuft.
- Lösungen die nicht offensichtlich besser als $\Omega(n^2)$ im *worst-case* sind und deren Laufzeit nicht bewiesen wurde.