

Algorithmen & Datenstrukturen

Übungsstunde 6

Miniquiz 5

- Passwort: **datastructures2**
- 13 Fragen
- 16 Minuten
- 4 Punkte insgesamt

Programm

- Miniquiz
- Rückmeldung Serie 4
- Theory Recap
- Prüfungsaufgaben
- Besprechung Aufgaben

Besprechung Miniquiz

Rückmeldung Serie 4

- Vorsichtig mit indexing
- Elemente und Positionen unterscheiden
- Bei 'exact number of calls' braucht es auf-/abrunden von $\log, \sqrt{}, /2$ u.ä.
- Rot ist meine Farbe

Theory Recap

Inhalt

- Abstrakte Datentypen 2
- Dynamische Programmierung 1

Binärbaum

- Für Knoten x sind Schlüssel in linkem Teilbaum $< x$ und Knoten im rechten Teilbaum $> x$
- search, insert und delete in $O(h)$
 - Problem je nach Struktur (Baum unique für gegebene Reihenfolge $\rightarrow h = n-1$ möglich)
 - Lösung: Flexibilität durch 2-3 Bäume

2-3 Baum

- Jeder Parent Knoten hat 2-3 Kinder
- Bei 3 Kindern brauchen wir 2 Werte beim Parent Node
 - Linkes Kind - kleiner gleich linker Wert
 - Mittleres Kind - grösser linkem und kleiner gleich rechtem Wert
 - Rechtes Kind - grösser als rechter Wert

2-3 Baum Cont'd

- `find(x)`: Gleich wie in Binärbaum
- `insert(x)`: Füge Blatt und Separator an richtiger Stelle ein, falls 3 Separatoren und 4 Kinder dadurch, immer Mittleren Separator hochziehen
- `delete(x)`: Lösche Knoten und Separator bei Elternknoten, falls nur noch 1 Kind danach, verschmelze mit Geschwisterknoten

Fibonacci

- Top-down schlecht
 - Laufzeit $\geq \Theta(1.5^n)$
 - Selbes Resultat wird mehrfach berechnet
- Memoization/Bottom-up
 - Laufzeit $\leq O(n)$
 - speichern bereits berechnete Resultate in Array

Maximum Subarray Sum

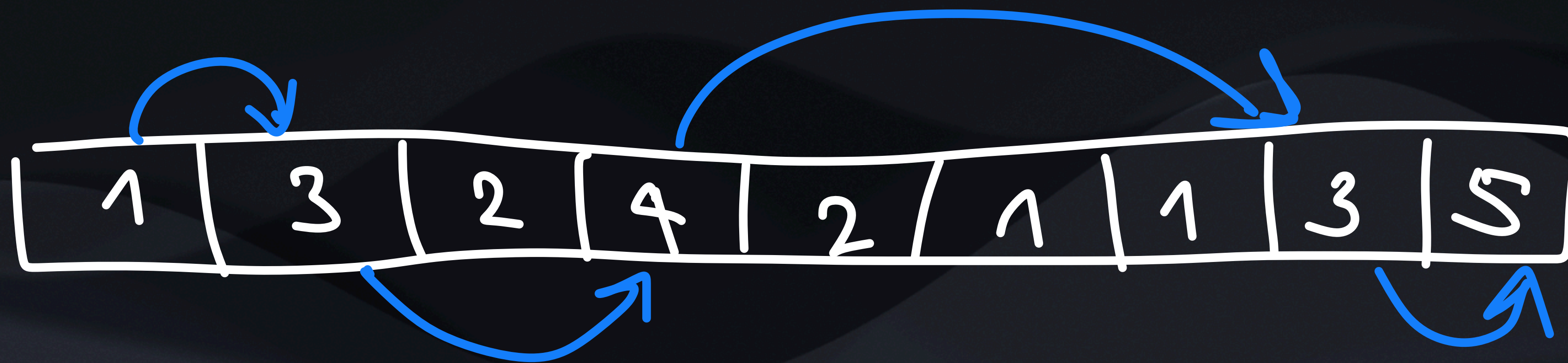
Beste Lösung

j	0	1	2	3	4	5	6	7	8	9
a_j	5	8	-4	1	-12	4	13	-2	-4	-1
R_j	5	13	9	10	-2	4	17	15	11	10
R_M	5	13	13	13	13	13	17	17	17	17

$$R_j = \begin{cases} R_{j-1} + a_j & \text{wenn } R_{j-1} \geq 0 \\ a_j & \text{sonst} \end{cases}$$

Jump Game

- Input: Array der Länge n mit positiven ganzen Zahlen
- Restriktion: von Feld i dürfen wir höchstens $A[i]$ nach vorne
- Gesucht: minimale Zahl an Sprüngen um n zu erreichen



minimum = 4 sprünge

Jump Game

- Teilproblem: $dp[k] :=$ maximaler index, der mit k Spüngen erreichbar ist
- Rekursion: $dp[k] = \max\{i + A[i] \mid dp[k - 2] < i \leq dp[k - 1]\}$
- Laufzeit: $O(n)$

Longest Common Subsequence

- Input: 2 Arrays mit Wörtern
- Restriktion: Teilfolge muss Reihenfolge beibehalten kann aber Stellen skippen
- Gesucht: Längste gemeinsame Teilfolge der Arrays

Longest Common Subsequence

- Teilproblem: $dp[i,j]$:= Länge LGT von $A[1..i]$, $B[1..j]$

- Rekursion: $dp[i][j] = \begin{cases} 0 & , \text{ if } i=0 \text{ or } j=0 \\ 1 + dp[i-1, j-1] & \leftarrow \text{benutze } A[i] \& B[j] \\ \max\{dp[i-1][j], dp[i][j-1]\} & , \text{ if } A[i] = B[j] \\ \end{cases}$, sonst

- D Y N A M I C

	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	1	1	1	1
	0	0	0	0	1	2	2	2
P								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								
O								
R								
S								

Edit Distance

- input: 2 Arrays mit Wörtern
- Restriktion: A in B umwandeln mit einfügen, löschen, ändern
- Gesucht: Editierdistanz

Edit Distance

- Teilproblem: $dp[i,j] :=$ Editierdistanz von $A[1..i]$ zu $B[1..j]$
- Rekursion: Was passiert mit $A[i]$ in optimaler Lösung?

$$dp[i][j] = \begin{cases} \max\{i,j\} & \text{Ändere } A[i] \text{ zu } B[j] \\ \min \left\{ \begin{array}{l} dp[i-1][j-1] + \begin{cases} 0, & A[i]=B[j] \\ 1, & \text{sonst} \end{cases} \\ dp[i-1][j] + 1, \quad \text{füge ein} \\ dp[i][j-1] + 1, \quad \text{lösche} \end{array} \right\} & \text{sonst} \end{cases}$$

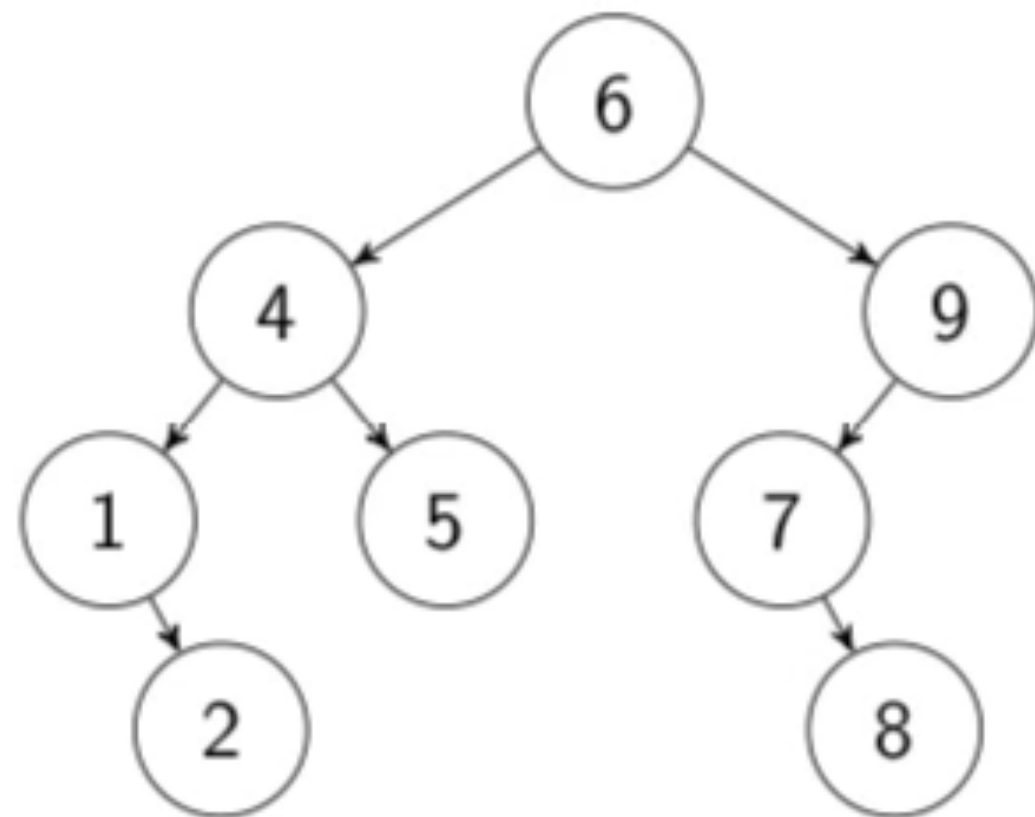
, i oder j = 0

, sonst

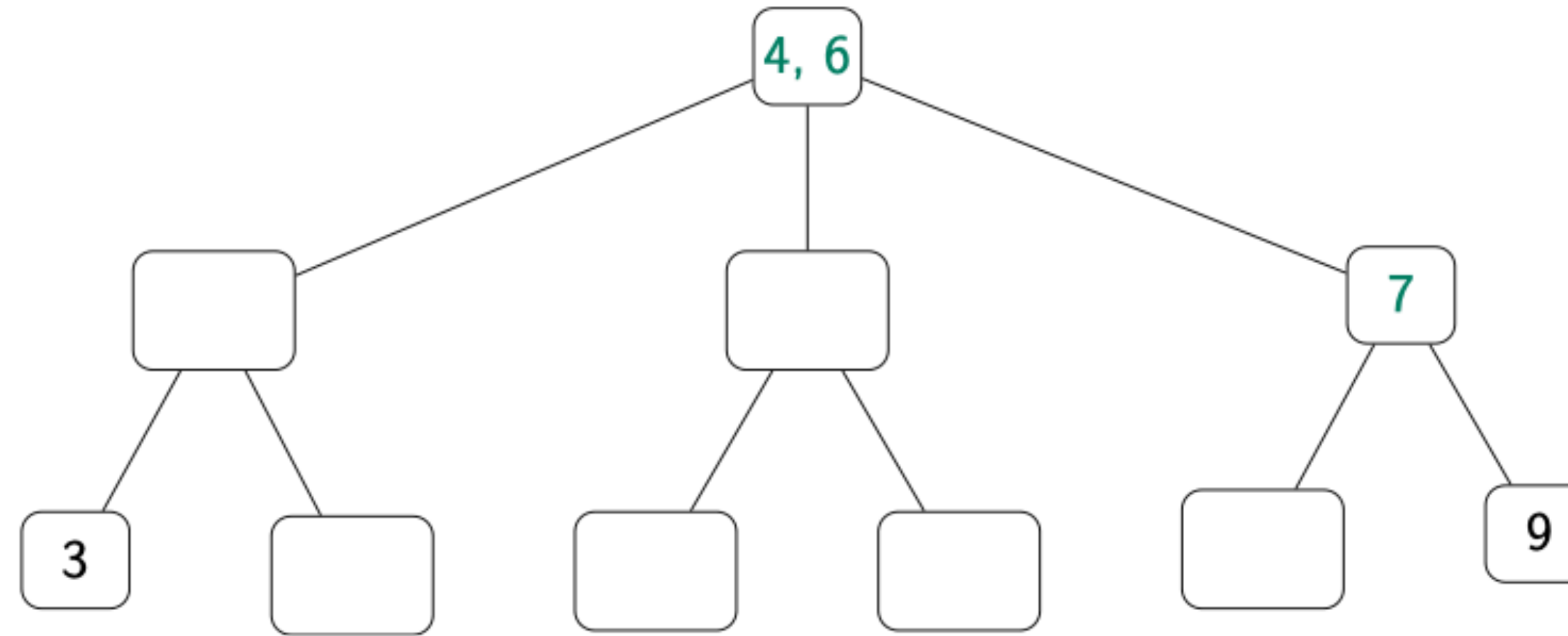
	-	T	I	R	E	D
-	0	1	2	3	4	5
E	1	1	2	3	3	4
T	2	1	2	3	4	4
H	3	2	2	3	4	5

g) *Binary search trees*: Draw the binary search tree that is obtained when inserting into an empty tree the keys 7, 2, 1, 5, 4, 8, 9 in this order.

h) *Binary search trees*: Draw the resulting binary search tree obtained by deleting the keys 6 and 9 in this order from the following binary search tree.

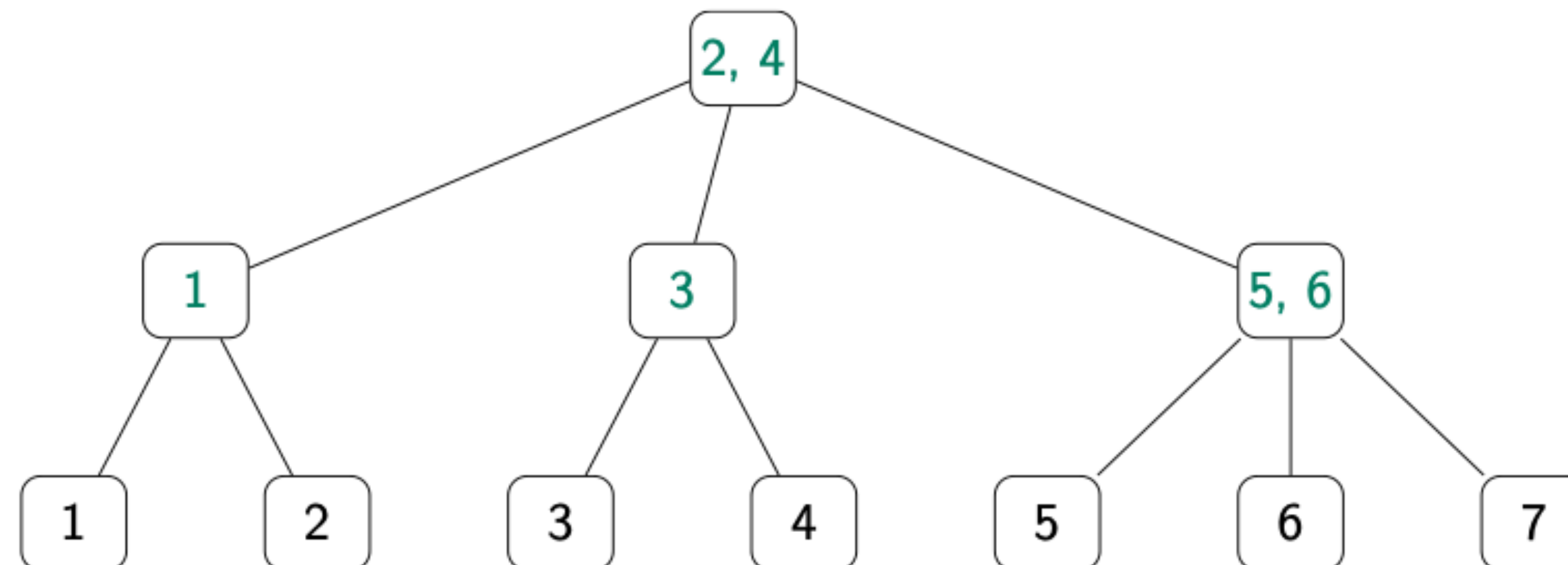


a) Betrachten Sie den folgenden 2-3 Baum:



Füllen Sie in die Knoten des Baumes ganzzahlige Werte ein, sodass ein gültiger 2-3 Baum entsteht.

b) Zeichnen Sie den 2-3 Baum, welcher aus dem folgenden Baum entsteht, nachdem 2 gelöscht wird. Es genügt den finalen 2-3 Baum zu zeichnen.



Exercise Recommendations

Aufgabenblatt 5

- 6.1 Machen, dauert nicht lang und bringt viel
- 6.2-6.3 Bonus
- 6.4 Gute DP Übung, dauert aber wahrscheinlich eher lang, kann auch auf Lernphase geschoben werden
- 6.5 Sehr gute Aufgabe zur Prüfungsvorbereitung (genau solche Aufgaben kommen häufig in Theorie und Programmierprüfung) falls jetzt keine Zeit, in Lernphase lösen!

Leetcode of the Week

- 55. Jump Game
- <https://leetcode.com/problems/jump-game>
- 45. Jump Game II
- <https://leetcode.com/problems/jump-game-ii>

Peergrading

- Aufgabe 5.4