Exercise sheet 5

Exercise 5.1

3	6	5	1	2	4	8	7
3	6	5	1	2	4	8	7
3	5	6	1	2	4	8	7

Insertion sort

3	6	5	1	2	4	8	7
3	5	1	2	4	6	7	8
3	1	2	4	5	6	7	8

Bubblesort

3	6	5	1	2	4	8	7
3	6	1	5	2	4	7	8
1	3	5	6	2	4	7	8

Merge sort

3	6	5	1	2	4	8	7
3	6	5	1	2	4	7	8
3	6	5	1	2	4	7	8

Selection sort

Exercise 5.2

a)

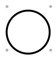
Alice: $a, b \in \{1, \dots, 200\}$

Bob $a', b' \in \{0, \dots, 201\}$

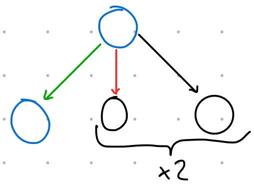
$k_0 = 1$

$k_n = 2\,k_{n-1} + 2$

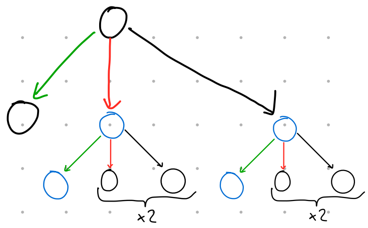
$k=0$



$k=1$



$k=2$



$k_n = 3 \cdot 2^n - 2$

Alice $\frac{200 \cdot 199}{2} = 19.900$

$K_{12} = 12.286 < 19.900$

b)

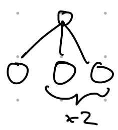
$l_1 = 1$

$l_{n+1} = 2 l_n + 1$

$k=0$

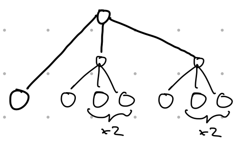
1

$k=1$



3

$k=2$

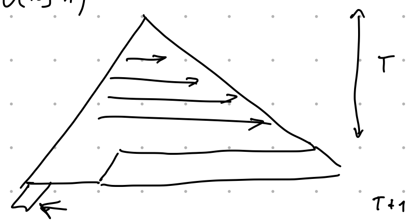


7

$l_n = 2^n - 1$ $l_{14} = 16.383 < 19.900$

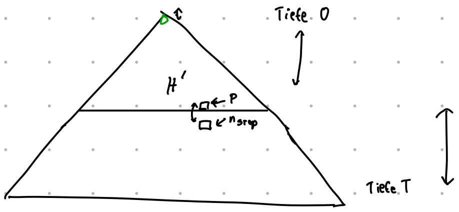
Exercise 5.3

(a) $O(\log n)$

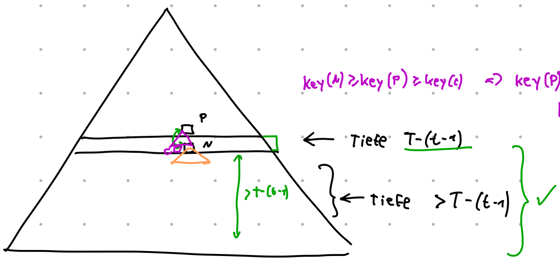


$T+1$ - Iteration $\lesssim O(\log n)$

(b)



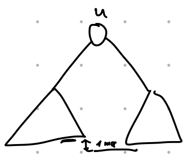
(c)



$\text{key}(N) \geq \text{key}(P) \geq \text{key}(c) \Rightarrow \text{key}(P) \geq \text{key}(N)$

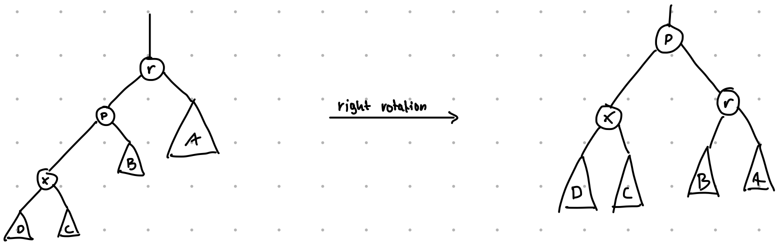
$\text{key}(D) \geq \text{key}(C)$

Theorie: AVL-Bäume

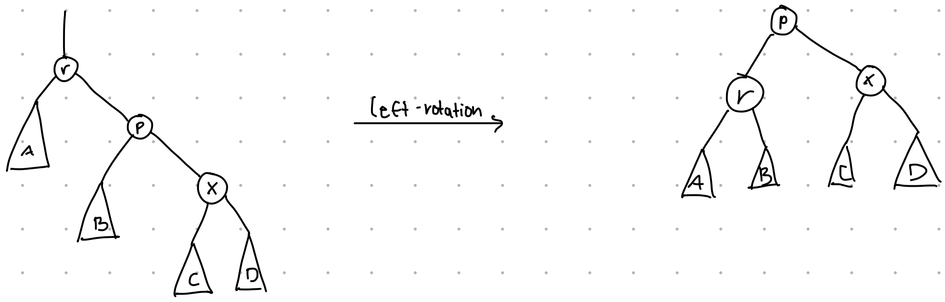


AVL(u): $|h_l(u) - h_r(u)| \leq 1$

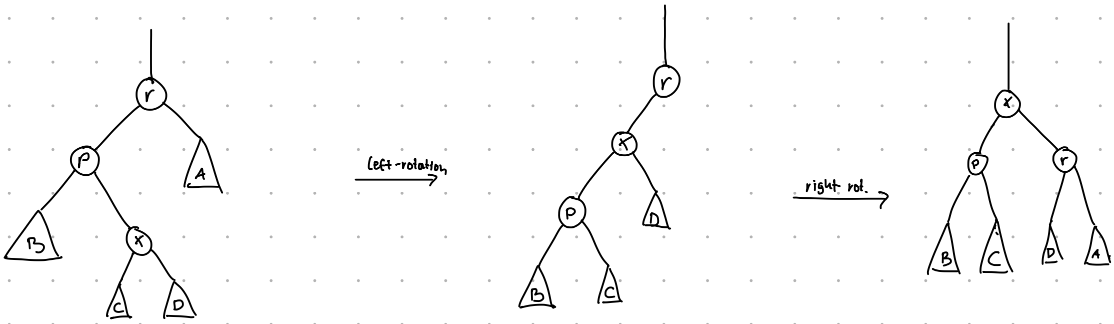
1. Case left-left



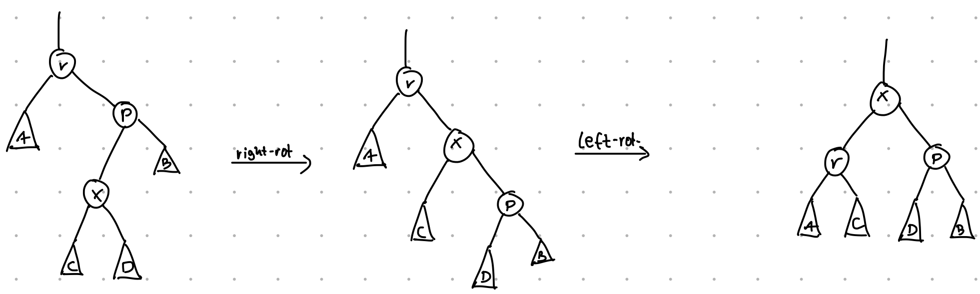
2. right-right



3. left-right

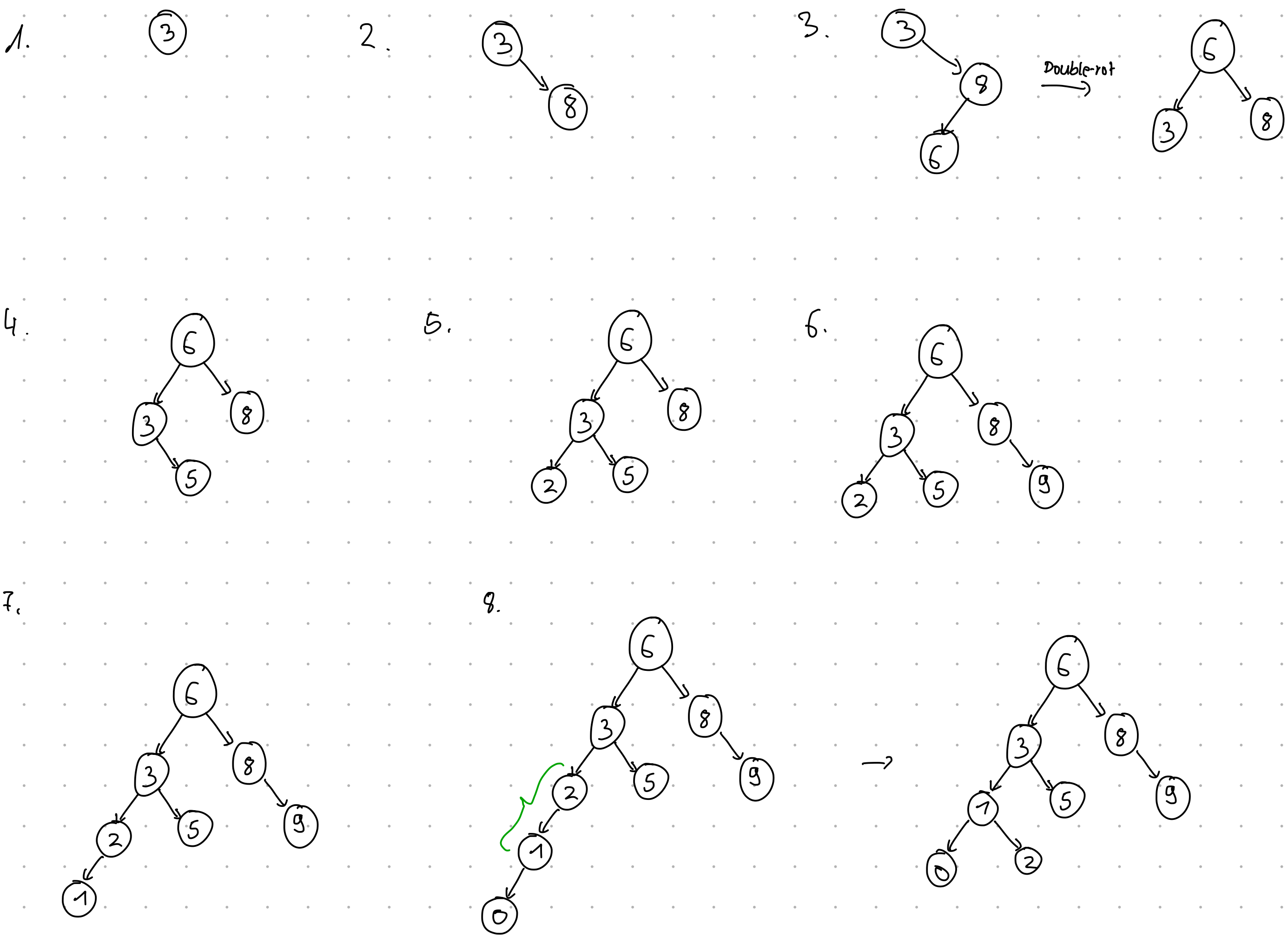


4. Fall: right-left

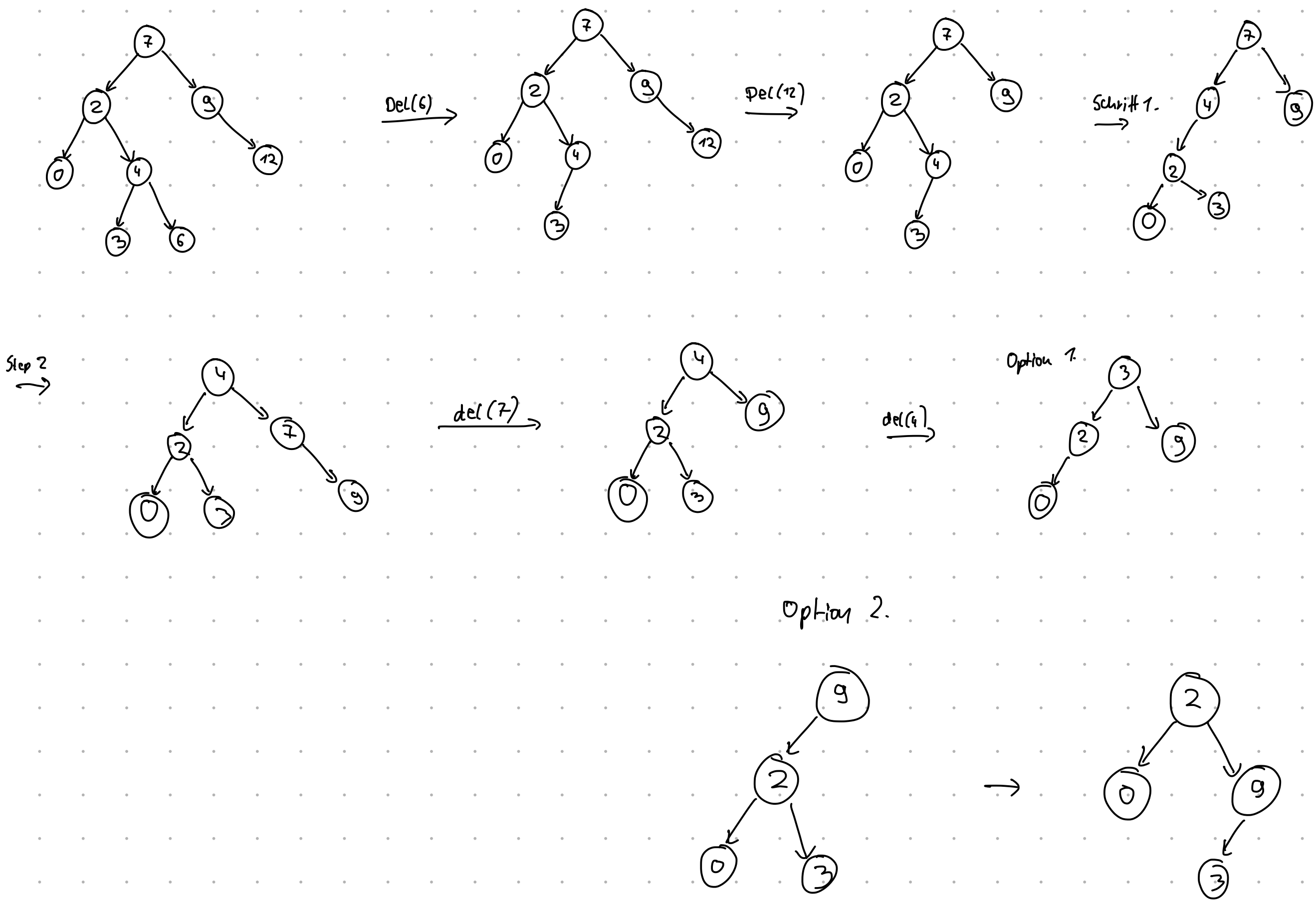


Exercise 5.5

(a)



(b)



Dynamic Programming

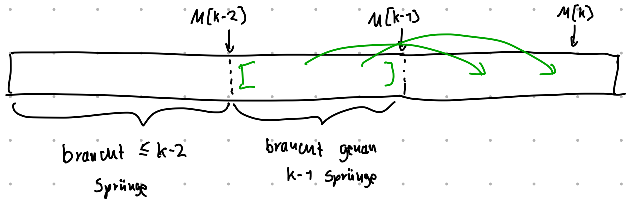
Jump Game

Array $A[1, \dots, n]$

Von pos i max. $A[i]$ Sprünge nach vorne

Teilproblem $M[k] :=$ maximaler Index, den wir in k Sprüngen erreichen können

$$M[k] = \max \{ i + A[i] \mid M[k-2] < i \leq M[k-1] \}$$



Längste Gemeinsame Teilfolge

m, n $DP[m+1][n+1]$

$L(i,j) :=$ Länge LGT von $A(1,\dots,i)$ und $B(1,\dots,j)$

	-	T	I	G	E	R
-	0	0	0	0	0	0
Z	0	0	0	0	0	0
I	0	0	$\boxed{1}$			
E	0					
G	0					
E	0					

1. Benutze $A[i]$ und $A[j]$ falls gleich

2./3. $A[i]$ / $A[j]$ nicht benutzen

Falls $A[i] = B[j]$, dann $L(i,j) = \max\{1 + L(i-1,j-1), L(i-1,j), L(i,j-1)\}$

Sonst $L(i,j) = \max\{L(i-1,j), L(i,j-1)\}$

"Code":

(Zuerst Base Cases)

```
for (i=1,...,n)
    for (j=1,...,m)
        if (A(i)==B(j))
            DP[i][j] = max{...}
        else
            DP[i][j] = ...
```

Editierdistanz

	/	T	I	G	E	R
/	0	1	2	3	4	5
Z	1	1	2			
I	2	2	1			
E	3					
G	4					
E	5					

$ED(i,j)$:= Editierdistanz $A(1,\dots,i)$ zu $B[1,\dots,j)$

Was passiert mit $A(i)$ in optimaler Lösung?

Fall 1: $A(i)$ wird gelöscht

$$ED(i,j) = 1 + ED(i-1,j)$$

Fall 2: $A(i)$ wird am Ende auf etwas in $B[1,\dots,j-1]$ gematched

=> Dann $ED(i,j) = 1 + ED(i,j-1)$

Fall 3: $A(i)$ auf $B(j)$ gematcht

$$ED(i,j) = \begin{cases} 0 + ED(i-1,j-1) & \text{falls } A(i) = B(j) \\ 1 + ED(i-1,j-1) & \text{falls } A(i) \neq B(j) \end{cases}$$

$A(i)$ wird in $B(j)$ geändert

für $i,j > 0$ $ED(i,j) = \min \{ ED(i-1,j)+1,$ — $A(i)$ löschen

$ED(i,j-1)+1,$ — $B(j)$ einfügen

$ED(i-1,j-1) + \begin{cases} 0 & \text{falls } A(i) = B(j) \\ 1 & \text{falls } A(i) \neq B(j) \end{cases}$

$\}$

Longest Snake

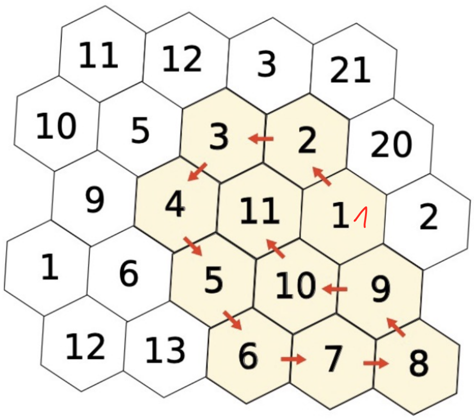


Figure 1: Example of a longest snake.

Input:
- Menge an Hexagon-Feldern $F = \{1, \dots, n\}$
- Jedes Feld $f \in F$ hat einen Wert $v_f \in \mathbb{N}$
- $\mathcal{N}(f) = \{g \mid g \text{ ist an } f \text{ benachbart}\}$ in $O(1)$.

Def. Schlange der Länge k: Sequenz von k Feldern $s = (f_1, f_2, \dots, f_k)$, sodass $\forall j \in \{1, \dots, k-1\}: \; f_{j+1} \in \mathcal{N}(f_j)$, also dass f_{j+1} Nachbar von f_j ist

UND $v_{j+1} = v_j + 1$.

Gesucht: Länge der längsten Schlange in F.

Rekursiv:

$$\text{maxSnake}(\overset{\substack{\text{Menge an Feldern}\\ \downarrow}}{F}, \underset{\substack{\uparrow \\ \text{Schlange endet in}\\ \text{Feld } i}}{i}) := \text{Länge der längsten Schlange, die bei Feld } i \text{ endet.}$$

$$\text{maxSnake}(F, i) = \max\left\{\text{maxSnake}(F \setminus \{i\}, j) + 1 \;\middle|\; j \in \mathcal{N}(i) \wedge v_i = v_j + 1\right\}$$

$$\text{return } \max_{1 \le i \le n} \text{maxSnake}(F, i)$$

$\sim$ exponential runtime

Dynamic-Programming

Dimensions of table: $1 \times n$

Meaning of an entry: $dp[i]$:= length of longest snake ending at field i.

Calculation of an entry:

$$dp[i] = \max\{dp[j] \mid j \in \mathcal{N}(i) \wedge v_i = v_j + 1\} + 1$$

Calculation order:

$\forall j \in \mathcal{N}(i) \wedge v_i = v_j + 1$ muss $dp[j]$ schon berechnet sein!

Lösung: Sortiere F nach v_f aufsteigend.

Runtime: $O(n \log n)$ wegen Sortieren.

Extracting the solution: $\max_{1 \le i \le n} dp[i]$

DP-Marathon

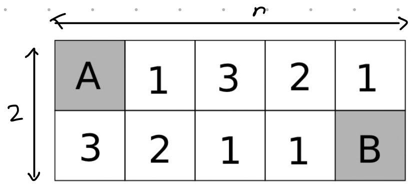


Figure 1: Runner problem for a cost array of size 2×5.

Imagine, a runner wants to run from A to B in Fig. 1. There are two lanes available. One is represented by the first row and the other by the second row. On some sections, the first lane is faster than the second lane, and vice versa. The runner can change lanes at any time, but this costs 1 minute every time. In this exercise, you are supposed to provide a dynamic programming algorithm that computes the optimal track.

Formally, the problem is defined in terms of a cost array $c \in \mathbb{N}^{2\times n}$. In Fig. 1 $n = 5$. Now, the runner starts at position $(1, 1)$ and wants to run to $(2, n)$. Running along a lane from field (i, j) to the field $(i, j + 1)$ requires $c_{i,j+1}$ minutes. Changing lanes from field $(1, j)$ to $(2, j)$ requires $1 + c_{2,j}$ minutes, and from field $(2, j)$ to $(1, j)$ requires $1 + c_{1,j}$ minutes.

Provide an algorithm using dynamic programming that computes the optimal track from A to B. Your algorithm should compute the ~~optimal sequence $(1, 1), (i_1, j_1), \ldots, (i_k, j_k), (2, n)$ and its~~ cost (= the time required by the runner to run the sequence).

Dimensions of DP-Table: $2 \times n$

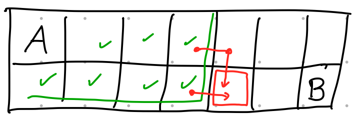
Meaning of an entry: $DP[i][j]$:= kürzeste Zeit benötigt um Feld (i,j) zu erreichen

Calculation of an entry:

$c_{i,j} \hat{=}$ Kosten des Feldes (i,j)

$$DP[1,j] = c_{1,j} + \min\{1 + DP[2,j-1] + c_{2,j},\ DP[1,j-1]\}$$

$$DP[2,j] = c_{2,j} + \min\{\underbrace{1 + DP[1,j-1] + c_{1,j}}_{DP[1,j]},\ DP[2,j-1]\}$$



Calculation order:

Base Case: $DP[1,1] = 0$

$DP[2,1] = c_{2,1} + 1$

von links nach rechts, unten nach oben.

Extracting the solution:

return $DP[2][n]$

Runtime: $O(n)$

P.G. Ex. 5.2