# Solutions

## 1. Introduction

1. Let $k = 2l$, $l \in \mathbb{N}$. The number $2^k - 1 = 2^{2l} - 1$ is divisible by 3. To understand this, we calculate $2^{2l} \bmod 3 = (2^2 \bmod 3)^l = 1$. Thus, $2^k - 1$ is divisible by 3.
   From $n = \frac{2^k-1}{3} \in \mathbb{N}$ for even $k \in \mathbb{N}$, it follows that $2^k - 1 = 3n$. Hence, $n$ is odd. Since $3n + 1 = 2^k$, Col($n$) terminates.

2. The invariant of the for loop reads: $a[1], \ldots, a[i]$ is sorted. We show the invariant by induction on $i$. The induction start for $i = 1$ is correct. Let $i \geq 2$. We assume that $a[1..i-1]$ is sorted and show the step from $i - 1$ to $i$. If $x \geq a[j] = a[i-1]$, then $a[i] = x$ (line 6). Thus, $a[1..i]$ is sorted.
   We consider the case $x < a[j]$. After termination of the while loop (i.e., immediately before line 6 is executed) the following is true:

   $$a[1] \leq a[2] \leq \ldots \leq a[j] \leq a[j+2] \leq a[j+3] \leq \ldots \leq a[i] \text{ and}$$
   $$a[j] \leq x < a[j+2].$$

   We show this by induction on the number $k$ of executions of the while loop. The condition is fulfilled if $k = 1$, i.e., $j = i - 2$. So let $k > 1$. We show the step from $k - 1$ to $k$, i.e., from $j$ to $j - 1$.
   By descending induction to $j$, it follows that

   $$a[1] \leq a[2] \leq \ldots \leq a[j], a[j+2] \leq a[j+3] \leq \ldots \leq a[i] \text{ and } x < a[j+2].$$

   After termination of the while loop, the following holds: $x \geq a[j]$. In total, $a[1..i]$ is sorted. For $i = n$, it follows that $a[1..n]$ is sorted, i.e., the algorithm is correct.

3. We have

   $$\begin{aligned} T_1(m_1) &= c_1 m_1 &&= T, \\ T_2(m_2) &= c_2 {m_2}^3 &&= T, \\ T_3(m_3) &= c_3 2^{m_3} &&= T, \end{aligned}$$

   $$\begin{aligned} c_1 k m_1 &= kT, \\ c_2 k {m_2}^3 = c_2 (\sqrt[3]{k} m_2)^3 &&= kT, \\ c_3 k 2^{m_3} = c_3 2^{ld(k)+m_3} &&= kT. \end{aligned}$$

The new maximum sizes of the entries $\tilde{m}_i$, $i = 1, 2, 3$, are $\tilde{m}_1 = km_1$, $\tilde{m}_2 = \sqrt[3]{k} \cdot m_2$ and $\tilde{m}_3 = ld(k) + m_3$.

4. With respect to the asymptotic growth, we have
$f_9 < f_5 < f_{11} < f_4 = f_3 < f_6 < f_{12} < f_2 < f_8 < f_7 < f_1 < f_{10}$

$$
\begin{aligned}
f_9 < f_5: &\ {}^{1}\!/\!_{3^n \log(\log(n))} \longrightarrow 0, n \longrightarrow \infty.\\
f_5 < f_{11}: &\ {}^{\log(\log(n))}\!/\!_{\sqrt{\log(\log(n))\log(n)}}\\
&= \sqrt{{}^{\log(\log(n))^2}\!/\!_{\log(\log(n))\log(n)}}\\
&= \sqrt{{}^{\log(\log(n))}\!/\!_{\log(n)}} \longrightarrow 0, n \longrightarrow \infty.\\
f_{11} < f_4: &\ \sqrt{{}^{\log(\log(n))\log(n)}\!/\!_{\log(\sqrt{n})}} =\\
&\sqrt{{}^{\log(\log(n))\log(n)}\!/\!_{(1/2\log(n))^2}}\\
&\sqrt{{}^{4\log(\log(n)}\!/\!_{\log(n)}} \longrightarrow 0, n \longrightarrow \infty.\\
f_4 = f_3: &\ {}^{\log(\sqrt{n})}\!/\!_{\log(n)} = {}^{1}\!/\!_{2}.\\
f_3 < f_6: &\ {}^{\log(n)}\!/\!_{\log(n)^2} = {}^{1}\!/\!_{\log(n)} \longrightarrow 0, n \longrightarrow \infty.\\
f_6 < f_{12}: &\ {}^{\log(n)^2}\!/\!_{2^{\sqrt{\log(\log(n))\log(n)}}} =\\
&2^{\log(\log(n)^2) - \sqrt{\log(\log(n))\log(n)}} \longrightarrow 0, n \longrightarrow \infty,\\
&\text{since, } \log(\log(n)^2) - \sqrt{\log(\log(n))\log(n)} =\\
&\sqrt{\log(\log(n))\log(n)} \left( 2\sqrt{\frac{\log(\log(n))}{\log(n)}} - 1 \right) \longrightarrow -\infty,\\
&n \longrightarrow \infty.\\
f_{12} < f_2: &\ {}^{2^{\sqrt{\log(\log(n))\log(n)}}}\!/\!_{\sqrt{n}} =\\
&\left( 2^{2\sqrt{\log(\log(n))\log(n)} - \log(n)} \right)^{1/2} \longrightarrow 0, n \longrightarrow \infty,\\
&\text{since, } 2\sqrt{\log(\log(n))\log(n)} - \log(n) =\\
&\log(n) \left( 2\sqrt{\frac{\log(\log(n))}{\log(n)}} - 1 \right) \longrightarrow -\infty, n \longrightarrow \infty.\\
f_2 < f_8: &\ {}^{\sqrt{n}}\!/\!_{\sqrt{n}\log(n)^2} = {}^{1}\!/\!_{\log(n)^2} \longrightarrow 0, n \longrightarrow \infty.\\
f_8 < f_7: &\ {}^{\sqrt{n}\log(n)^2\log(n)}\!/\!_{n} = {}^{\log(n)^3}\!/\!_{\sqrt{n}} \longrightarrow 0, n \longrightarrow \infty.\\
f_7 < f_1: &\ {}^{n}\!/\!_{n\log(n)} = {}^{1}\!/\!_{\log(n)} \longrightarrow 0, n \longrightarrow \infty.\\
f_1 < f_{10}: &\ {}^{n}\!/\!_{(3/2)^n} = {}^{n}\!/\!_{2^{n(\log(3)-1)}} \longrightarrow 0, n \longrightarrow \infty.
\end{aligned}
$$

5. $f_1$ and $f_2$ have the same order,
$f_1, f_2, f_3 = O(f_4)$,

$f_1, f_2, f_4 \neq O(f_3)$,
$f_3 \neq O(f_1), f_3 \neq O(f_2)$.

6. We first investigate the behavior of $n(\sqrt[n]{n}-1)$ for $n \to \infty$.

$$\lim_{n\to\infty} n(\sqrt[n]{n}-1) = \lim_{n\to\infty} \frac{n^{\frac{1}{n}}-1}{\frac{1}{n}} = \lim_{x\to 0} \frac{\left(\frac{1}{x}\right)^x - 1}{x}$$
$$= \lim_{x\to 0} \frac{e^{x\ln\left(\frac{1}{x}\right)}-1}{x}.$$

Because of $\lim_{x\to 0} x\ln\left(\frac{1}{x}\right) = \lim_{y\to\infty} \frac{\ln(y)}{y} = 0$ the limit of this quotient is of the form "$\frac{0}{0}$". We apply the rule of l'Hopital and differentiate numerator and denominator.
From $\frac{d}{dx}\left(x\ln\left(\frac{1}{x}\right)\right) = \ln\left(\frac{1}{x}\right) - 1$, it follows that

$$\frac{d}{dx}\left(e^{x\ln\left(\frac{1}{x}\right)}-1\right) = e^{x\ln\left(\frac{1}{x}\right)}\left(\ln\left(\frac{1}{x}\right)-1\right).$$

Since

$$\lim_{x\to 0} \frac{e^{x\ln\left(\frac{1}{x}\right)}\left(\ln\left(\frac{1}{x}\right)-1\right)}{1} = \infty,$$

we have $\lim_{n\to\infty} n(\sqrt[n]{n}-1) = \infty$ (which is not obvious).

We now show that $n(\sqrt[n]{n}-1) = O(\ln(n))$ holds.

$$\lim_{n\to\infty} \frac{n(\sqrt[n]{n}-1)}{\ln(n)} = \lim_{n\to\infty} \frac{n^{\frac{1}{n}}-1}{\frac{\ln(n)}{n}} = \lim_{x\to 0} \frac{e^{x\ln\left(\frac{1}{x}\right)}-1}{x\ln\left(\frac{1}{x}\right)} \quad \left(= \text{“}\tfrac{0}{0}\text{”}\right).$$

Because of

$$\lim_{x\to 0} \frac{e^{x\ln\left(\frac{1}{x}\right)}\left(\ln\left(\frac{1}{x}\right)-1\right)}{\ln\left(\frac{1}{x}\right)-1} = \lim_{x\to 0} e^{x\ln\left(\frac{1}{x}\right)} = 1$$

we have

$$\lim_{n\to\infty} \frac{n(\sqrt[n]{n}-1)}{\ln(n)} = 1.$$

Thus $n(\sqrt[n]{n}-1) = O(\ln(n))$.

$f_2 = O(n^k)$, since

$$\binom{n}{k} k! \frac{1}{n^k} = \frac{n(n-1)\dots(n-(k-1))}{n\cdot n\dots n} = \left(1-\frac{1}{n}\right)\dots\left(1-\frac{k-1}{n}\right)$$

converges to 1 for $n \longrightarrow \infty$.

7. The difference equation

$$k_n = \left(1 + \frac{p}{100}\right) k_{n-1} + c, k_0 = k,$$

has the solution

$$\begin{aligned} k_n &= \left(1 + \frac{p}{100}\right)^n \left(k + \sum_{i=1}^{n} c \left(1 + \frac{p}{100}\right)^{-i}\right) \\ &= k \left(1 + \frac{p}{100}\right)^n + c \sum_{i=1}^{n} \left(1 + \frac{p}{100}\right)^{n-i} \\ &= k \left(1 + \frac{p}{100}\right)^n + c \sum_{i=0}^{n-1} \left(1 + \frac{p}{100}\right)^{i} \\ &= k \left(1 + \frac{p}{100}\right)^n + \frac{100c\left(\left(1 + \frac{p}{100}\right)^n - 1\right)}{p}. \end{aligned}$$

8. a. Solution of the equation:

$$\pi_n = \prod_{i=2}^{n} 1 = 1, n \geq 1,\ x_n = 1 + \sum_{i=2}^{n} i = \frac{n(n+1)}{2}.$$

b. Solution of the homogeneous equation:

$$\pi_n = \prod_{i=2}^{n} \frac{i+1}{i} = \frac{n+1}{2}, n \geq 1.$$

Solution of the equation:

$$\begin{aligned} x_n &= \frac{n+1}{2} \sum_{i=2}^{n} \frac{4(i-1)}{i(i+1)} \\ &= 2(n+1) \sum_{i=2}^{n} \left(\frac{2}{i+1} - \frac{1}{i}\right) \\ &= 2(n+1) \left(\mathrm{H}_n + \frac{2}{n+1} - 2\right) \\ &= 2(n+1)\mathrm{H}_n - 4n. \end{aligned}$$

9.

$$x_1 = 2,\ x_n = 2n \cdot x_{n-1} + (n+1)!.$$

$$\pi_n = \prod_{i=2}^{n} 2i \ = \ 2^{n-1} n!.$$

$$x_n = 2^{n-1} n! \left(2 + \sum_{i=2}^{n} \frac{(i+1)!}{2^{i-1} i!}\right)$$

$$\begin{aligned}
&\sum_{i=2}^{n} \frac{i}{2^{i-1}} + \sum_{i=2}^{n} \frac{1}{2^{i-1}} \\
&= \frac{(n+1)\,(1/2)^n\,(-1/2) - ((1/2)^n - 1)}{1/4} - 1 + \frac{(1/2)^n - 1}{-1/2} - 1 \\
&= 4(-(n+1)\,(1/2)^{n+1} - (1/2)^{n+1} + 1) - 1 - 2\,((1/2)^n - 1) - 1 \\
&= 4 - (n+3)\,(1/2)^{n-1}\,.
\end{aligned}$$

$$\begin{aligned}
x_n &= 2^{n-1} n!(2 + 4 - (n+3)\,(1/2)^{n-1}) \\
&= n!(3 \cdot 2^n - (n+3))
\end{aligned}$$

10.

$$x_1 = 0,\ x_n = \sum_{i=1}^{n-1} x_i + 2(n-1).$$

$$x_n - x_{n-1} = x_{n-1} + 2(n-1) - 2(n-2).$$

$$x_n = 2x_{n-1} + 2.$$

$$\pi_n = \prod_{i=2}^{n} 2 \;=\; 2^{n-1}.$$

$$x_n = 2^{n-1} \sum_{i=2}^{n} \frac{2}{2^{i-1}} \;=\; 2^n \sum_{i=1}^{n-1} \frac{1}{2^i} \;=\; \sum_{i=1}^{n-1} 2^i \;=\; 2^n - 2.$$

11. Let $n = 2^{2^k}$ and $x_k = T(2^{2^k})$. Then $k = \log_2(\log_2(n))$.

$$\begin{aligned}
T(n) &= T(\sqrt{n}) + \log_2(n)^l,\ l = 0, 1. \\
x_k &= T(2^{2^k}) \;=\; T(2^{2^{k-1}}) + (2^l)^k. \\
x_k &= x_{k-1} + (2^l)^k,\ x_1 = 2^l. \\
x_k &= 2^l + \sum_{i=2}^{k} (2^l)^i \;=\; \sum_{i=1}^{k} (2^l)^i \;=\; \begin{cases} k & \text{for } l = 0, \\ 2^{k+1} - 2 & \text{for } l = 1. \end{cases}
\end{aligned}$$

Applying the transformation $k = \log_2(\log_2(n))$, we get

$$\begin{aligned}
T(n) &= x_{\log_2(\log_2(n))} \\
&= \begin{cases} \log_2(\log_2(n)) & \text{for } l = 0, \\ 2^{\log_2(\log_2(n))+1} - 2 = 2(\log_2(n) - 1) & \text{for } l = 1. \end{cases}
\end{aligned}$$

12. By Proposition 1.15

$$x_k = a^{k-1}(ad + cb^l) + a^{k-1}c\sum_{i=2}^{k}\frac{b^{li}}{a^{i-1}} \;=\; a^k d + ca^k \sum_{i=1}^{k}\left(\frac{b^l}{a}\right)^i$$
$$= \begin{cases} a^k d + cn^l \frac{q^k-1}{q-1}, & \text{falls } q \neq 1, \\ a^k d + ckn^l, & \text{falls } q = 1. \end{cases}$$

Applying the inverse transformation $k = \log_b(n)$ we get

$$T_{(\frac{a}{b})}(n) = x_{\log_b(n)} = \begin{cases} da^{\log_b(n)} + cn^l \frac{q^{\log_b(n)}-1}{q-1} & \text{if } b^l \neq a, \\ da^{\log_b(n)} + cn^l \log_b(n) & \text{if } b^l = a. \end{cases}$$

Note, that $\log_b(n) = \lfloor \log_b(n) \rfloor$ for $n = b^k$.

13. a = 1, l = 0:

$$T(n) = 1 + \lfloor \log_2(n) \rfloor = O(\log_2(n)).$$

a = 1, l = 1:

$$1 + n\left(1 - \left(\frac{1}{2}\right)^{\lfloor \log_2(n) \rfloor}\right) \leq T(n) \leq 1 + 2n\left(1 - \left(\frac{1}{2}\right)^{\lfloor \log_2(n) \rfloor}\right) = O(n)$$

a = 2, l = 0:

$$T(n) = 2^{\lfloor \log_2(n) \rfloor + 1} - 1 = O(n)$$

a = 2, l = 1:

$$2^{\lfloor \log_2(n) \rfloor} + \frac{n}{2}\lfloor \log_2(n) \rfloor \leq T(n) \leq 2^{\lfloor \log_2(n) \rfloor} + n\lfloor \log_2(n) \rfloor = O(n \log_2(n)).$$

14. The number of recursive calls is

$$a_n = a_{n-1} + 1, a_1 = 1.$$

This equation has the solution $a_n = n$, i.e., $n$ stack frames are required.

15. We display the call hierarchy for solving the puzzle for $n$ slices: The call TowersOfHanoi$(n, A, B, C)$ causes the call TowersOfHanoi$(n-1, A, C, B)$, the move of the $n$th disc from $A$ to $B$ and the call TowersOfHanoi$(n-1, C, B, A)$

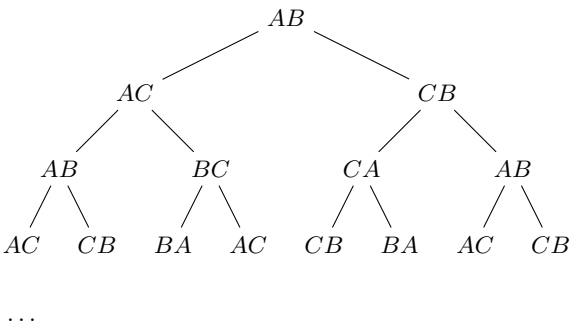



Fig. 2.1: Call hierarchy for TowersOfHanoi.

Note the arrangement in each level:
$AB, BC, CA, \ldots$ or $AC, CB, BA, \ldots$.
We get the necessary steps for the solution, if we traverse this tree inorder.
For $n = 4$, for example, the result is the sequence
$AC, AB, CB, AC, BA, BC, AC, AB, CB, CA, BA, CB, AC, AB, CB$.
The first step corresponds to the node that is on the far left on the lowest level. It is $AB$ if $n$ is odd and $AC$ if $n$ is even. We number the nodes in the order of in-order traversal. Numbers: $1, 3, 5, \ldots$ are leaves, i.e., the nodes of level $n - 1$.
Numbers: $2, 6, 10, \ldots$ are nodes of level $n - 2$. Numbers: $4, 12, 20, \ldots$ are the nodes of the level $n - 3$.
The nodes are distributed among the individual levels as follows:

| level | first node | distance | gen. form | number of |
|---|---|---|---|---|
| $n-1$ | 1 | 2 | $1 + j * 2, j = 0, \ldots, 2^{n-1} - 1$ | $2^{n-1}$ |
| $n-2$ | 2 | $2^2$ | $2 + j * 2^2, j = 0, \ldots, 2^{n-2} - 1$ | $2^{n-2}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n-i$ | $2^{i-1}$ | $2^i$ | $2^{i-1} + j * 2^i, j = 0, \ldots, 2^{n-i} - 1$ | $2^{n-i}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 1 | $2^{n-2}$ | $2^{n-1}$ | $2^{n-2} + j * 2^{n-1}, j = 0, 1 = 2^1 - 1$ | 2 |
| 0 | $2^{n-1}$ | | $2^{n-1}$ | 1 |

A node with the number $k$ is located in the $i$th level if and only if

$$k \bmod 2^{n-i-1} = 0 \text{ und } k \bmod 2^{n-i} \neq 0.$$

This means that in the binary expression for $k$ the first 1 from the left is at the position $n - i$.
For $n = 4$ we get

| $AC$ | $AB$ | $CB$ | $AC$ | $BA$ | $BC$ | $AC$ | $AB$ | $CB$ | $CA$ | $BA$ | $CB$ | $AC$ | $AB$ | $CB$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 3 | 2 | 3 | 1 | 3 | 2 | 3 | 0 | 3 | 2 | 3 | 1 | 3 | 2 | 3 |

The second line indicates the number of the node and the third line indicates the level in which the node is located.

Starting from a leaf $k$ ($k$ odd), the successor $k+1$ and the successor $k+2$ of the successor can be simply determined: If the node $k+1$ is located in the level $l$, which can be calculated according to the formula from above, then the node $k$ is one time left successor and $n-l-2$ times right successor of $k+1$ in the tree. Starting from $k$ you follow in the tree $n-l-2$ times to the left and once to the right to get to the predecessor $k+1$. If a node is a left successor, the first symbol of the labeling is taken from the predecessor node, the second symbol from the predecessor is changed. For a right successor, it is just the opposite. If you turn left on the way to the predecessor $n-l-2$ times, the second symbol is kept. The first symbol is changed when $n-l-2$ is odd. If you turn to the right-hand side once, the first symbol is retained and the second is changed.
The successor $k+2$ of the successor $k+1$ is the leaf to the right of $k$.
Applying these rules, we get the following table, which shows the successor and the successor of the successor

| $k$ | $k+1, n-l-2$ gerade | $k+1, n-l-2$ ungerade | $k+2$ |
|---|---|---|---|
| $AB$ | $AC$ | $CA$ | $BC$ |
| $AC$ | $AB$ | $BA$ | $CB$ |
| $BA$ | $BC$ | $CB$ | $AC$ |
| $BC$ | $BA$ | $AB$ | $CA$ |
| $CA$ | $CB$ | $BC$ | $AB$ |
| $CB$ | $CA$ | $AC$ | $BA$ |

16. When using the generic algorithm (Algorithm 1.34) for matroids, it must be decided at each step whether $O \cup \{s_i\}$ is admissible. To do this, we keep the list $O$ of selected tasks sorted in ascending order by completion time, i.e., we keep the sorting when adding the next task $s_i$. We place $s_i$ at the end of the list and sort the list with InsertionSort (Algorithm 1.57).
Let $s_i$ be the $l_i$–th element in the sorted list. $O \cup \{s_i\}$ is allowed if and only if $l_i \leq$ the completion time of $s_i$. To implement $O$ with an array, $l_i$ is the index of $s_i$. If $O$ is implemented with a chained list, the numbering of the order must be explicit.
17. Let $n_l > n_{l-1} > n_1$ be the values of the coins ($n_1 = 1$) We are looking for $\nu_1, \ldots, \nu_l \geq 0$ with

$$\sum_{i=1}^{l} \nu_i n_i = n \text{ and } \sum_{i=1}^{l} \nu_i \text{ minimal.}$$

Greedy strategy: We set $r_l = n$ and calculate successively for $i = l \ldots 1$

$$\nu_i = \left\lfloor \frac{r_i}{n_i} \right\rfloor, \; r_{i-1} = r_i \bmod n_i \; = \; r_i - \nu_i n_i.$$

Available denominations: 1 cent, 2 cent, 5 cent, 10 cent, 20 cent, 50 cent, 100 cent, 200 cent
For a solution generated by the above algorithm we have

$$\begin{aligned}&\nu_1 \leq 1,\ \nu_2 \leq 2, \nu_3 \leq 1,\\ &\nu_4 \leq 1, \nu_5 \leq 2, \nu_6 \leq 1,\\ &\nu_7 \leq 1.\end{aligned}$$

An optimal solution also meets all these conditions. Because, if one of the conditions is violated, a solution can can be found with fewer coins. These conditions determine a solution uniquely. $n \bmod 5 = \nu_1 + 2\nu_2$. Therefore, $\nu_1$ and $\nu_2$ are uniquely determined ($n \bmod 5 = 0$: $\nu_1 = 0, \nu_2 = 0$; $n \bmod 5 = 1$: $\nu_1 = 1$; $n \bmod 5 = 2$: $\nu_2 = 1$; $n \bmod 5 = 3$: $\nu_1 = 1, \nu_2 = 1$; $n \bmod 5 = 4$: $\nu_2 = 2$). $\nu_3$ is 0, if and only if $(n - \nu_1 - 2\nu_2) \bmod 10 = 0$. Hence, $\nu_3$ is uniquely determined. $\nu_4$ is 0, if and only if $(n - \nu_1 - 2\nu_2 - 5\nu_3) \bmod 20 = 0$. Thus, $\nu_4$ is uniquely determined. $(n - \nu_1 - 2\nu_2 - 5\nu_3 - 10\nu_4) \bmod 50$ uniquely determines $\nu_5$. $(n-\nu_1-2\nu_2-5\nu_3-10\nu_4-20\nu_5) \bmod 100$ uniquely determines $\nu_6$. $(n-\nu_1-2\nu_2-5\nu_3-10\nu_4-20\nu_5-50\nu_6) \bmod 200$ uniquely determines $\nu_7$. Thus, $\nu_8$ is also uniquely determined. The above algorithm provides an optimal solution. (For the denominations $n_3 = 11, n_2 = 5, n_1 = 1$ the greedy strategy is not successful ($n = 15$)).

18. We convert the recursive formula for solving the editing distance into a recursive algorithm P$(i, j)$. Let $T(i, j)$ be the number of calls necessary to calculate with the algorithm P $d(i, j)$. Then

$$\begin{aligned}&T(0,0) = T(i,0) \ = \ T(0,j) \ = 1,\\ &T(i,j) = T(i,j-1) + T(i-1,j) + T(i-1,j-1) + 1.\end{aligned}$$

Hence

$$T(n,m) \geq 3T(n-1,m-1) \geq 3^k,\ k = \min\{n,m\}.$$

Although there are only $(n+1)\cdot(m+1)$ subproblems, the running time is exponential. This is true because the same distances are calculated over and over again. For large $n$ and $m$ the algorithm cannot be used.

19. The recursive formula for $l(n, m)$ is

$$\begin{aligned}&l(0,0) = 0, l(i,0) \ = \ 0, l(0,j) \ = 0,\\ &l(i,j) = l(i-1,j-1) + 1 \text{ if } i,j > 0 \text{ und } a_i = b_j,\\ &l(i,j) = \max\{l(i,j-1), l(i-1,j)\} \text{ if } i,j > 0 \text{ and } a_i \neq b_j.\end{aligned}$$

Apply dynamic programming to compute $l(n, m)$.

20. Let $m_n = \max_{i,j \leq n} f(i, j)$. We define $M_k$, $N_k$ by

$$\begin{aligned}&M_1 = N_1 = a_1,\\ &N_k = \max\{N_{k-1} + a_k, a_k\}, k \geq 2,\\ &M_k = \max\{M_{k-1}, N_k\}, k \geq 2.\end{aligned}$$

By induction on $k$, it follows that

$$N_k = \max_i f(i,k) \text{ und } M_k = m_k = \max_{i,j \le k} f(i,j).$$

The assertion holds for $k = 1$. Step from $k-1$ to $k$:

$$N_k = \max\{N_{k-1} + a_k, a_k\} = \max\{\max_i f(i,k-1) + a_k, a_k\} = \max_i f(i,k).$$

We distinguish the two cases:

a. $a_k$ does not occur in the calculation of $m_k$.
   Then $m_k = m_{k-1} = M_{k-1} = M_k$.
b. $a_k$ appears in the calculation of $m_k$.
   Then $m_k = \max_i f(i,k) = N_k = M_k$.

We calculate the sequences $(N_k)_{k=1,\ldots,n}$ and $(M_k)_{k=1,\ldots,n}$ with dynamic programming.

21. We follow with our solution [MotRag95, page 171]. Without restriction we assume $a, b \in \{0,1\}^*$. We consider $a_i \ldots a_{i+m-1}$ and $b_1 \ldots b_m$ as binary representations of numbers with the most significant digit on the left. The string $b$ is a substring of $a$, if $a_i \ldots a_{i+m-1} = b_1 \ldots b_m$ for an $i$, $1 \le i \le n-m+1$. Solving the problem by a complete search has a running time in the order $O(mn)$. A more efficient solution can be developed by using fingerprints.
We choose a family of hash functions

$$h_p : \{0,1\}^l \longrightarrow \{0, \ldots, p-1\},\ x \longmapsto x \bmod p,$$

as in Section 1.6.2. The collision probability for a randomly chosen prime number $p$ is for $x \neq y$

$$\mathrm{p}(h_p(x) = h_p(y)) \le \frac{1}{t}.$$

(Proposition 1.51). We no longer compare substrings $a_i \ldots a_{i+m-1}$ of $a$ and $b_1 \ldots b_m$, but their hash values

$$h_p(a_i \ldots a_{i+m-1}) = h_p(b_1 \ldots b_m).$$

We have

$$a_{i+1} \ldots a_{i+m} = 2(a_i \ldots a_{i+m-1} - a_i 2^{m-1}) + a_{i+m}.$$

Thus

$$h_p(a_{i+1} \ldots a_{i+m}) = 2(h_p(a_i \ldots a_{i+m-1}) - a_i 2^{m-1}) + a_{i+m} \bmod p.$$

$h_p(a_{i+1} \ldots a_{i+m})$ can be calculated from $h_p(a_i \ldots a_{i+m-1})$ in constant time. We get an algorithm with the running time $O(n+m)$. For $h_p(a_i \ldots a_{i+m-1}) = h_p(b_1 \ldots b_m)$ we can check in the time $O(m)$ whether a collision is present.

## 2. Sorting and Searching

1. a. **Algorithm 2.1.**

```
Sort2(item a[1..n])
 1  index l, r, boolean loop ← true
 2  l ← 1, r ← n, a[0] ← 1, a[n + 1] ← 2
 3  while loop do
 4       while a[l] = 1 do l ← l + 1
 5       while a[r] = 2 do r ← r − 1
 6       if l < r
 7         then exchange a[l] and a[r]
 8              l = l + 1, r = r − 1
 9         else loop ← false
```

First, the records which contain 3 are placed at the end of the array, analogous to Sort2. Then Sort2 is applied to the part of the array containing only 1 and 2.

b. **Algorithm 2.2.**

```
Sort(item a[1..n])
 1  index i = 1, j
 2  while i ≤ n do
 3       j ← a[i].key
 4       if i = j
 5         then i ← i + 1
 6         else exchange a[i] and a[j]
```

2. Let $a_i$ be the number of executions of line $i$ in the worst case and $\tilde{a}_i$ the number of executions of line $i$ on average – each depending on $n$.

a. Sorting by insertion (Algorithm 1.57).
Comparisons in line 4:

$$a_4 = \sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1 = \frac{n^2}{2} + \frac{n}{2} - 1.$$

$\tilde{a}_4$ = number of inversions + $(n-1)$:

$$\tilde{a}_4 = \frac{n(n-1)}{4} + (n-1) = \frac{n^2}{4} + \frac{3n}{4} - 1.$$

$$a_5 = \sum_{i=2}^{n} (i-1) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}.$$

$\tilde{a}_5$ = number of inversions on average

$$\tilde{a}_5 = \frac{n(n-1)}{4} = \frac{n^2}{4} - \frac{n}{4}.$$

We summarize the results:

| Zeile $i$ | $a_i$ | $\tilde{a}_i$ |
|---|---|---|
| $3, 6$ | $n-1$ | $n-1$ |
| $4$ | $n^2/2 + n/2 - 1$ | $n^2/4 + 3n/4 - 1$ |
| $5$ | $n^2/2 - n/2$ | $n^2/4 - n/4$ |

b. BubbleSort (Algorithm 2.32).

$$a_4 = \tilde{a}_4 = \sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}.$$

$$a_5 = a_4 = \frac{n^2}{2} - \frac{n}{2}.$$

$\tilde{a}_5$ = number of inversions on average

$$\tilde{a}_5 = \frac{n(n-1)}{4} = \frac{n^2}{4} - \frac{n}{4}.$$

Summarizing, we get

| Zeile $i$ | $a_i$ | $\tilde{a}_i$ |
|---|---|---|
| $4$ | $n^2/2 - n/2$ | $n^2/2 - n/2$ |
| $5$ | $n^2/2 - n/2$ | $n^2/4 - n/4$ |

The worst case scenario is the reverse sorted array. In this case, line 5 is executed each time.

For the number of comparisons in the worst case ($C_w$), the number of comparisons on average ($C_\emptyset$), the number of assignments in the worst case ($A_w$) and the number of assignments on average ($A_\emptyset$) we get:

| | SelectionSort | InsertionSort | BubbleSort |
|---|---|---|---|
| $C_w$ | $n^2/2 - n/2$ | $n^2/2 + n/2 - 1$ | $n^2/2 - n/2$ |
| $C_\emptyset$ | $n^2/2 - n/2$ | $n^2/4 + 3n/4 - 1$ | $n^2/2 - n/2$ |
| $A_w$ | $n^2/2 - 7n/2 - 4$ | $n^2/2 + 5n/2 - 3$ | $3n^2/2 - 3n/2$ |
| $A_\emptyset$ | $(n+1)\mathrm{H}_n + 2n - 4$ | $n^2/4 + 11n/4 - 3$ | $3n^2/4 - 3n/4$ |

3. InsertionSort and BubbleSort are stable.
   SelectionSort is not stable. $2_1, 2_2, 1$ becomes $1, 2_2, 2_1$. SelectionSort can be implemented stable. For this all elements from the insertion point to the position before the minimum are shifted one position to the right. This reduces the performance considerably.
   Quicksort and heapsort are not stable.
4. The while loop in the Algorithm 2.1 terminates with $l = r$ if and only if $a[r] \leq x$ and $a[l] \geq x$ holds, i.e., $a[l] = a[r] = x$ $(= a[j])$. If all elements

in $a$ are pairwise distinct, the while loop in the Algorithm 2.1 terminates with $l = r + 1$. Thus, $n + 1$ comparisons take place. We get

$$\tilde{C}(n, i) = C(i - 1) + C(n - i) + n + 1.$$

$C(n)$ is the mean value of the $\tilde{C}(n, i)$:

$$\begin{aligned} C(n) &= \frac{1}{n} \sum_{i=1}^{n} \tilde{C}(n, i) \\ &= \frac{1}{n} \sum_{i=1}^{n} (C(i - 1) + C(n - i) + n + 1) \\ &= \frac{2}{n} \sum_{i=0}^{n-1} C(i) + n + 1, n \geq 2, \end{aligned}$$

We set

$$x_n = \sum_{i=0}^{n} C(i).$$

Then

$$x_n - x_{n-1} = \frac{2}{n} x_{n-1} + n + 1.$$

We get the difference equation

$$\begin{aligned} x_1 &= C(0) + C(1) = 0, \\ x_n &= \frac{n+2}{n} x_{n-1} + n + 1, n \geq 2. \end{aligned}$$

That equation has the solution

$$x_n = (n + 1)(n + 2) \left( \mathrm{H}_{n+1} + \frac{1}{n+2} - \frac{11}{6} \right)$$

(page 16, (d)). We get

$$C(n) = \frac{2}{n} x_{n-1} + n + 1 = 2(n + 1)\mathrm{H}_n - \frac{8n + 2}{3}.$$

5. The execution of exchange in line 11 is superfluous if the pivot element is the largest element. This is the case with probability $\frac{1}{n}$. The average number of exchanges (without recursion) averaged over all pivot elements is

$$\frac{1}{n} \sum_{i=1}^{n} \left( \frac{(i - 1)(n - i)}{n - 1} + 1 - \frac{1}{n} \right) = \frac{(n + 4)}{6} - \frac{1}{n}.$$

We set $x_n = \sum_{i=0}^{n} E(i)$ and get

$$x_1 = E(0) + E(1) = 0,$$
$$x_n = \frac{n+2}{n} x_{n-1} + \frac{n+4}{6} - \frac{1}{n}, n \geq 2.$$

The solution of this equation is

$$\begin{aligned} x_n &= \frac{(n+1)(n+2)}{6} \left( \sum_{i=2}^{n} \frac{i+4}{(i+1)(i+2)} - \sum_{i=2}^{n} \frac{6}{i(i+1)(i+2)} \right) \\ &= \frac{(n+1)(n+2)}{6} \left( \mathrm{H}_{n+1} - \frac{5}{n+2} + \frac{3}{n+1} - \frac{4}{3} \right). \end{aligned}$$

Using partial fraction decomposition, we get

$$\frac{6}{i(i+1)(i+2)} = \frac{3}{i} - \frac{6}{i+1} + \frac{3}{i+2}.$$

The average number of exchanges

$$\begin{aligned} E(n) &= \frac{2}{n} x_{n-1} + \frac{n+4}{6} - \frac{1}{n} \\ &= \frac{n+1}{3} \left( \mathrm{H}_n - \frac{5}{n+1} + \frac{3}{n} - \frac{4}{3} \right) + \frac{n+4}{6} - \frac{1}{n} \\ &= \frac{1}{3}(n+1)\mathrm{H}_n - \frac{5n+8}{18}. \end{aligned}$$

6. We have to solve

$$T(n) = cn + \frac{2}{n} \sum_{i=0}^{n-1} T(i), n \geq 2, T(0) = T(1) = b.$$

The aim is to transform this recursion into a difference equation by a suitable substitution. For recursions in which the $n$–th element depends on the sum of all predecessors, this is achieved with the substitution

$$x_n = \sum_{i=0}^{n} T(i).$$

Then

$$x_n - x_{n-1} = cn + \frac{2}{n} x_{n-1}.$$

Hence

$$x_n = \frac{n+2}{n} x_{n-1} + cn, n \geq 2, x_1 = 2b.$$

The solution of that equation is

$$x_n = (n+1)(n+2) \left( c\mathrm{H}_{n+1} + \frac{2c}{n+2} + \frac{1}{6}(2b - 13c) \right).$$

(equation (d), page 16).
From the solution for $x_n$ a solution for $T(n)$ can easily be derived.

$$\begin{aligned} T(n) &= \frac{2}{n}x_{n-1} + cn \\ &= \frac{2}{n}n(n+1)\left(c\mathrm{H}_n + \frac{2c}{n+1} + \frac{1}{6}(2b-13c)\right) + cn \\ &= 2c(n+1)\mathrm{H}_n + \frac{1}{3}(2b-10c)n + \frac{1}{3}(2b-c) \end{aligned}$$

is a closed solution for $T(n)$.

7. With $n$ elements, the pivot element only needs to be compared once with each of the $n-1$ remaining elements. For this purpose the indices are controlled. A comparison is only necessary if $l < r$ holds and a comparison for $l = r$. We get

$$C(1) = 0, C(n) = C\left(\left\lfloor \frac{n-1}{2} \right\rfloor\right) + C\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + n - 1.$$

Then

$$C(n) \leq 2C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n - 1.$$

We replace $\leq$ with $=$ and solve the recursion

$$C(n) = 2C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n - 1.$$

Let $n = n_{k-1} \ldots n_0$, $n_{l-1} = 1$, the binary expression of $n$. Then $k = \lfloor \log_2(n) \rfloor + 1$.

$$\begin{aligned} C(n) &= \sum_{i=0}^{k-2} 2^i \left(\left\lfloor \frac{n}{2^i} \right\rfloor - 1\right) \leq (k-1)n - (2^{k-1} - 1) \\ &= n\lfloor \log_2(n) \rfloor - 2^{\lfloor \log_2(n) \rfloor} + 1 \end{aligned}$$

(Lemma 1.25 with $g^i(n) = \lfloor \frac{n}{2^i} \rfloor$, $r(n) = n - 1$ and $d = 0$).

8. See [Ďurian86].
9. Apply heapsort and stop after $k$-steps.
10. a. The number of comparisons is $n - 1$. This optimization is at the expense of the number of exchanges. The code is remarkably simple.
    b. Invariant of the for loop:

$$a[i], \ldots, a[l-1] \leq x \text{ and } a[l], \ldots, a[k-1] > x.$$

This is true for $k = i$. Because for $k = i$, we have $l = i$ and for empty arrays the statement is trivially correct.
Conclusion from $k-1$ to $k$:

If $a[k] \leq x$, then $a[k]$ and $a[l]$ are swapped. After incrementing $l$, $a[i], \ldots, a[l-1]$ are $\leq x$ and after incrementing $k$, $a[l], \ldots, a[k-1]$ are $> x$.
If $a[k] > x$, $l$ remains unchanged and after incrementing $k$, $a[l], \ldots, a[k-1]$ are $> x$.
The invariant remains valid even after termination of the for loop. After execution of exchange in line 8, the pivot element is at position $l$ and the following holds $a[i], \ldots, a[l-1]$ are $\leq x$ and $a[l+1], \ldots, a[j]$ are $> x$.
QuickSortVariant sorts the array $a$ in ascending order. The subarrays which are parameters of QuickSortVariant have a maximum of $n-1$ elements, if $n$ is the number of elements of $a$. The proof follows immediately by induction.

11. **Notes on the algorithm.**
    a. Pivot returns an index $i \neq 0$ if there are at least two different elements. The pivot element is stored in $a[i]$ and there is an element in the $a$ which is smaller than $a[i]$.
    b. The splitting process is only necessary if the return value of Pivot is $> 0$.
    c. The first exchange (line 5) is not necessary (but gives simple code).
    d. After termination of repeat-until we have $a[i], \ldots, a[l-1] < x$ and $a[l], \ldots, a[j] \geq x$.

We assume, that the elements to be sorted are pairwise distinct.
Set $n := j - i + 1$. Effort for lines 2 - 8 of QuickSort: $cn$, $c$ constant. The pivot element is located at the first or second position. Each of the $n-1$ decompositions has the probability $1/(n-1)$. If you divide into arrays of length $r$ and $n-r$, the running time $T(n)$ is recursively:

$$T(n) = T(r) + T(n-r) + cn, c \text{ constant.}$$

**Average term.** For the average term $T(n)$, this results in (averaged over all decompositions):

$$T(n) = cn + \frac{1}{n-1}\sum_{r=1}^{n-1}(T(r) + T(n-r)) = cn + \frac{2}{n-1}\sum_{r=1}^{n-1} T(r), n \geq 2.$$

We solve:

$$T(1) = b, T(n) = cn + \frac{2}{n-1}\sum_{i=1}^{n-1} T(i), n \geq 2.$$

Set

$$x_n = \sum_{i=1}^{n} T(i).$$

Then

$$x_n - x_{n-1} = cn + \frac{2}{n-1}x_{n-1}.$$

We get the equation

$$x_1 = b,\ x_n = \frac{n+1}{n-1}x_{n-1} + cn, n \geq 2.$$

The solution of that equation is

$$\begin{aligned}
\pi_n &= \prod_{i=2}^{n} \frac{i+1}{i-1} = \frac{n(n+1)}{2}, n \geq 2, \pi_1 = 1. \\
x_n &= \frac{n(n+1)}{2}\left(b + \sum_{i=2}^{n} ci\frac{2}{i(i+1)}\right) \\
&= \frac{n(n+1)}{2}\left(b + 2c\sum_{i=2}^{n}\frac{1}{i+1}\right) \\
&= \frac{n(n+1)}{2}(2c\mathrm{H}_{n+1} - 3c + b).
\end{aligned}$$

We get for $T(n)$:

$$\begin{aligned}
T(n) &= \frac{2}{n-1}x_{n-1} + cn \\
&= \frac{2}{n-1}\frac{(n-1)n}{2}(2c\mathrm{H}_n - 3c + b) + cn \\
&= 2cn(\mathrm{H}_n - 1) + bn
\end{aligned}$$

**Running time in the worst-case.** The worst case occurs when in each recursion step the decomposition process yields a one-element and a $(n-1)$-element set.

$$T(n) = T(n-1) + T(1) + cn, T(1) = b.$$

has the solution

$$T(n) = \left(b + \sum_{i=2}^{n}(c \cdot i + b)\right) = c\left(\frac{n(n+1)}{2} - 1\right) + bn.$$

The worst case occurs with the specified pivot procedure for a sorted array.

12. **Algorithm 2.3.**

```
QuickSort(item a[i..j])
 1  item x, index l, r
 2  while i < j do
 3      if p ← Pivot(a[i..j]) ≠ 0
 4        then x ← a[p], l ← i, r ← j
 5             repeat
 6                 exchange a[l] and a[r]
 7                 while a[l] < x do l ← l + 1
 8                 while a[r] ≥ x do r ← r − 1
 9             until l = r + 1
10             QuickSort(a[i..l − 1])
11             i ← l
12        else i ← j
```

We show that the while loop always terminates. We look at two cases:
1. If $\text{Pivot}(a, i, j) \neq 0$, the decomposition will be performed. After the decomposition, $i = l$ is set. Thus, $i$ is increased by at least 1.
2. If $\text{Pivot}(a, i, j) = 0$, all elements in $a[i..j]$ are equal and thus sorted. $i = j$ is set. The while-loop and QuickSort terminate. This case occurs at the latest for $i = j - 1$.
The consideration shows that the while loop and QuickSort always terminate. This results in a sequence of starting points: $i_1 = i < i_2 < \ldots < i_n$. $i_k$ and $\tilde{i}_k$ denotes the value of $i$ at the $k$th iteration of the while loop at loop entry or at loop exit. By induction on $k$, it follows that $a[i, \tilde{i}_k]$ is sorted at loop exit. Thus, $a[i..j]$ is sorted at termination.
This version of quicksort uses the same decompositions as Algorithm 2.34. Therefore, the same formula for the running time is obtained.
**QuickSort with logarithmically limited recursion depth.** The idea is to perform first the smaller part of the decomposition recursively. After each recursive call, the number of elements reduces by more than half. This results in a logarithmically limited recursion depth.

13. The number of assignments is the same for all variants.
Assignments in BuildHeap: Number of executions of row 2 + number of executions of row 9 + number of executions of row 10 in DownHeap:

$$2 \cdot \left\lfloor \frac{n}{2} \right\rfloor + I_1(n).$$

Assignments in the sorting phase: Number of executions of row 2 + number of executions of row 9 + number of executions of line 10 in DownHeap + 3 times number of executions of line 5 in Heapsort:

$$2(n-1) + I_2(n) + 3(n-1) = 5(n-1) + I_2(n).$$

14. We consider an array with $n = 2^k - 1$ elements which is sorted in reverse order. In this case the upper bound is accepted. For example, consider 15,14,13,12,11,10,9,8,7,6,5,4,3,2,1. Each element ends up in a leaf when percolating down. Thus, the maximum number of iterations is reached:

$$I(n) = \sum_{l=1}^{\lfloor \frac{n}{2} \rfloor} \left\lfloor \log_2 \left( \frac{n}{l} \right) \right\rfloor .$$

15. Using Mergesort you can sort without additional memory. When merging the sorted sub-arrays $a[i, l]$ and $a[l + 1, j]$ the elements between insertion position and extraction position are shifted one position to the right, if an element from the second part is to be inserted,
With $temp\,[1..\lfloor n/2 \rfloor]$ as auxiliary memory this is not necessary. First copy the first part into the auxiliary memory and merge the two parts into $a$.

    In Merge, there are at most $n - 1$ comparisons. Let $C(n)$ be the number of comparisons. Then

$$C(n) \leq C\left( \left\lfloor \frac{n}{2} \right\rfloor \right) + C\left( \left\lceil \frac{n}{2} \right\rceil \right) + n - 1,\ C(1) = 0.$$

    We replace $\leq$ in the recursion with $=$ and get an upper limit for the number of comparisons. Set $n = 2^k$ and $x_k = C(2^k)$. Then $C(2^k) = 2C(2^{k-1}) + 2^k - 1$ and $C(2) = 1$. We get the difference equation

$$x_k = 2x_{k-1} + 2^k - 1,\ x_1 = 1.$$

    This has the solution

$$x_k = 2^{k-1} \left( 1 + \sum_{i=2}^{k} \frac{2^i - 1}{2^{i-1}} \right) = (k - 1)2^k + 1$$

    (Proposition 1.15). We set $k = \log_2(n)$ then

$$C(n) = C(2^{\log_2(n)}) = x_{\log_2(n)} = (\log_2(n) - 1)n + 1 = n \log_2(n) - (n - 1).$$

    Because of $\log_2(n) = \lfloor \log_2(n) \rfloor$ for $n = 2^k$ is

$$C(n) = n\lfloor \log_2(n) \rfloor - (2^{\lfloor \log_2(n) \rfloor} - 1)$$

    an integer solution (Lemma B.23).

16. Determine an element $x$ of order $k$ with Quickselect. Then perform a partitioning step of quicksort with pivot element $x$. Split so that elements $\leq x$ are on the left of $x$ and elements $> x$ are on the right of $x$ (analogous to Algorithm 2.34). The $k$ smallest elements are on the left of $x$.

17. This version finds the first element among equals, Algorithm 2.28 finds any among equals. The following holds true
    a.
    $$l \leq \left\lfloor \frac{l+r}{2} \right\rfloor \leq r.$$
    b. Since $r_{i+1} - l_{i+1} < r_i - l_i$ holds, while terminates with $l = r$.
    c. After a decomposition, the larger part has $\left\lfloor \frac{n-1}{2} + 1 \right\rfloor = \left\lceil \frac{n}{2} \right\rceil$ elements. $I(n)$ fulfills the recursion:
    $$I(n) \leq I\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1, I(1) = 0.$$
    With Proposition 1.28, it follows that $I(n) \leq \lfloor \log_2(n) \rfloor + 1$.

## 3. Hashing

1. A mapping $f$ is given by its value vector $(f(0), \ldots, f(n-1))$. Thus, $\mathcal{F}(M,N) \cong N^m$ and $|\mathcal{F}(M,N)| = |N^m| = n^m$. An injective mapping $f$ is given by its value vector $(f(0), \ldots, f(m-1))$, $f(i) \neq f(j)$ for $i \neq j$. The number of injective mappings is

$$\binom{n}{m} \cdot m! = \frac{n!}{(n-m)!}$$

   Thus, the percentage of injective mappings is

$$\frac{n!}{(n-m)! \cdot n^m} \cdot 100.$$

   For $n = m$ the percentage of injective mappings is $\frac{n!}{n^n} \cdot 100 \approx \frac{\sqrt{2\pi n}}{e^n} \cdot 100$.[11] For $n = 100$ the percentage is $\approx 10^{-40}$. Injective mappings occur rarely.

2. We calculate the values for a 16-bit processor and 11-bit hash values. For this we develop 16 digits of $(0.618)_d$ binary: $(0.1001111000110101)_b$. The hash values are:

| $k$ | binary | decimal |
|---|---|---|
| 0 | 10011110001 | 1265 |
| 1 | 00111100011 | 483 |
| 2 | 01111000110 | 966 |
| 3 | 11110001101 | 1933 |
| 4 | 11100011010 | 1818 |
| 5 | 11000110101 | 1589 |
| 6 | 10001101010 | 1130 |
| 7 | 00011010100 | 212 |
| 8 | 00110101000 | 428 |
| 9 | 01101010000 | 848 |
| 10 | 11010100110 | 1696 |

3. Let $(x_1, y_1), (x_2, y_2) \in \mathbb{Z}_p \times \mathbb{Z}_p, (x_1, y_1) \neq (x_2, y_2)$.
   From $h_a = h_{\tilde{a}}$, it follows that $ax + y = \tilde{a}x + y$ for $x, y \in \mathbb{Z}_p$. Hence, $ax = \tilde{a}x$ $(y = 0)$ and $a = \tilde{a}$ $(y = 0, x = 1)$. Thus, $|\mathcal{H}| = p$.
   From $ax_1 + y_1 = ax_2 + y_2$ follows $a(x_1 - x_2) = y_2 - y_1$ and for $x_1 \neq x_2$ it follows that $a = (y_2 - y_1)(x_1 - x_2)^{-1}$, i.e, $|\{h_a \in \mathcal{H} \mid h_a(x_1, y_1) = h_a(x_2, y_2)\}| = 1$.
   For $x_1 = x_2$, it follows that $y_1 = y_2$, this contradicts the assumption $(x_1, y_1) \neq (x_2, y_2)$, hence $\mathrm{p}(h_a(x_1, y_1) = h_a(x_2, y_2)) = 1/p$.

[11] Here we used Stirling's formula $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$.

4. Let $\mathrm{Lin}(\mathbb{Z}_p^k, \mathbb{Z}_p^l)$ be the set of linear mappings $A : \mathbb{Z}_p^k \longrightarrow \mathbb{Z}_p^l$. The linear mappings are represented by $l \times k$- matrices. Thus, $\mathrm{Lin}(\mathbb{Z}_p^k, \mathbb{Z}_p^l) \cong M(l \times k, \mathbb{Z}_p)$ and $|\mathrm{Lin}(\mathbb{Z}_p^k, \mathbb{Z}_p^l)| = |M(l \times k, \mathbb{Z}_p)| = p^{kl}$.
Let $A \in M(l \times k, \mathbb{Z}_p)$ and let $x_1, x_2 \in \mathbb{Z}_p^k$, $x_1 \neq x_2$, with $A(x_1) = A(x_2)$. Then $A(x_1 - x_2) = 0$. Let $y \in \mathbb{Z}_p^k \setminus \{0\}$. We calculate the number of matrices $A \in M(l \times k, \mathbb{Z}_p)$ with $A(y) = 0$. Since $(y_1, \ldots, y_k) \neq 0$, there is a $y_i \neq 0$. For the columns $A_1, \ldots, A_k$ of $A$, we have $A_i = -\frac{1}{y_i} \sum_{k \neq i} y_k A_k$. It follows

$$|\{A \in M(l \times k, \mathbb{Z}_p) \mid A(x_1) = A(x_2)\}| = p^{(k-1)l}$$

Altogether, it follows that the family $\mathrm{Lin}(\mathbb{Z}_p^k, \mathbb{Z}_p^l)$ is a universal family of hash functions.

5. Statement: The elements from $\{j \cdot c \bmod m \mid 0 \leq j \leq m-1\}$ are pairwise distinct. Suppose $j \cdot c \equiv \tilde{j} \cdot c \bmod m \Longrightarrow m$ divides $(i - j) \cdot c$.
$\gcd(m, c) = 1 \Longrightarrow m$ divides $j - \tilde{j}$, a contradiction, since $|j - \tilde{j}| < m$.

6. $n = 10,000$, $B = {}^{n}/_{m}$.
$1 + {}^{1}/_{2}B = 2 \Longrightarrow B = 2 \Longrightarrow m = 5,000$.
$\mathrm{E}(kol) = n - m(1 - p_0), p_0 = e^{-2} = 0.0006179$.
$\mathrm{E}(kol)/_n = 1 - {}^{1}/_{B}(1 - p_0) = 0.5676$.
56.76 % of the records cause collisions.
Overflow area: 5676 places, primary area: 5,000 places.

7. We use the formula $\frac{1}{B} \ln\left(\frac{1}{1-B}\right)$ for the average length of a probing sequence when searching for an element. For $B = 0.8$ we get two accesses on average. With $B = \frac{n}{m}$ we obtain $l = 1250$. We choose for $m = 1259$, the smallest prime number $> 1250$.

8. Use a hash procedure with hash function $h : \{\{0.1\}^k \longrightarrow t[0..m]$ to speed up the search for matching substrings. In the table $t$ indices are stored, which point into the text buffer $tb[i - w..i - 1]$ relative to the beginning of the preview buffer $pb[i..i + v - 1]$. For example, use the first 3 characters $pb[1..3]$ as input for the hash function ($k = 24$). One step of the compression consists of
   a. Search the longest matching subsequence $l$ with the starting points defined by the entries in $t$, colliding with $h(pb[1], pb[2], pb[3])$.
   b. Store the index for $pb[1..3]$ in $t$.
   c. Code $l$ and move text and preview buffers.
   d. Update all indexes in $t$, which is necessary due to moving text and preview buffers.

   For details see appendix B in [HanHarJoh98], which also contains an implementation in $C$ for compression and decompression.

9. a. Uniqueness: Use a hash procedure and check for collisions. For this purpose $n$ hash values have to be calculated. Let $L = \{l_1, \ldots, l_n\}$.

Use a hash function $h$ and successively calculate $h(l_i)$, $i = 1, \ldots, n$. For $h(l_i) = h(l_j)$ compare $l_i$ and $l_j$.

b. Use a hash function $h$ and calculate $h(z_i)$, $i = 1, \ldots, n$. Then check if $h(s - z_i)$, $i = 1, \ldots, n$, collides with $h(z_j)$, $j = 1, \ldots, n$. In case of collisions check if $s - z_i = z_j$ is valid.

10. The probability

$$p_i := p_{ij} := p(n_j = i) = \frac{\binom{\frac{|S|}{m}}{i}\binom{|S|-\frac{|S|}{m}}{n-i}}{\binom{|S|}{n}}.$$

indicates that $i$ keys are mapped to a value $j$. To see this we compute, number of positive cases:
Select in $h^{-1}(j)$ $i$ elements. This has $\binom{\frac{|S|}{m}}{i}$possibilities.
Select $n - i$ elements from the remaining fibers. There are $\binom{|S|-\frac{\perp}{m}}{n-i}$ possibilities for this. There are $\binom{|S|}{n}$ possible cases.

The distribution $\mathrm{p}(n_j = i)$ is the hypergeometric distribution with the parameters $\left(n, M = \frac{|S|}{m}, N = |S| - \frac{|S|}{m}\right)$. We have $\mathrm{E}(n_j) = n\frac{M}{N} = \frac{n}{m-1}$ (Proposition A.24).

11. a. $bm$ denotes the available memory.
b. $\beta$ is proportional to $B$ ($\beta = bB$).
The number of keys mapped to $j$ is

$$n_j = |\{s \in S \mid H(s) = j\}|, j = 0, \ldots, m - 1.$$

The number of values with $i$ pre-images $w_i = \sum_{j=0}^{m-1} \delta_{n_j,i}, i = 0, \ldots, n$.

$$\mathrm{E}(w_i) = mp_i, p_i = \binom{n}{i}\left(\frac{1}{m}\right)^i\left(1 - \frac{1}{m}\right)^{n-i}.$$

If $i$ keys are mapped to a value, $i - b$ keys lead to collisions. The number of collisions is

$$kol = \sum_{i=b+1}^{n}(i - b)w_i.$$

$n_j, w_i, kol$ are random variables.
**Proposition.** *Let $H$ be a hash table with $m$ blocks, block size $b$ and load factor $\beta$. In $H$ are $n$ elements saved.*
*The average number of collisions is*

$$m\left(\frac{n}{m} - \sum_{i=1}^{b} ip_i - b\left(1 - \sum_{i=0}^{b} p_i\right)\right).$$

*Proof.*

$$\begin{aligned}
\mathrm{E}(Kol) &= E\left(\sum_{i=b+1}^{n}(i-b)w_i\right) = \sum_{i=b+1}^{n}(i-b)\mathrm{E}(w_i) \\
&= \sum_{i=b+1}^{n}(i-b)mp_i. = m\sum_{i=b+1}^{n}(i-b)p_i. \\
&= m\left(\sum_{i=b+1}^{n} ip_i - b\sum_{i=b+1}^{n} p_i\right) \\
&= m\left(\frac{n}{m} - \sum_{i=1}^{b} ip_i - b\left(1 - \sum_{i=0}^{b} p_i\right)\right).
\end{aligned}$$

□

**Corollary.** *Let $H$ be a hash table with $m$ blocks, block size $b$ and load factor $\beta$. In $H$ are $n$ elements stored. The percentage of inserted elements causing collisions is*

$$f(b,\beta) = \frac{100}{\beta}\left(\beta - \sum_{i=1}^{b} ip_i - b\left(1 - \sum_{i=0}^{b} p_i\right)\right).$$

*Proof.*

$$\begin{aligned}
\frac{\mathrm{E}(Kol)}{n} &= \frac{m}{n}\left(\frac{n}{m} - \sum_{i=1}^{b} ip_i - b\left(1 - \sum_{i=0}^{b} p_i\right)\right) \\
&= \frac{1}{\beta}\left(\beta - \sum_{i=1}^{b} ip_i - b\left(1 - \sum_{i=0}^{b} p_i\right)\right).
\end{aligned}$$

□

We now calculate values for $f(b,\beta)$. For this we use $p_i \approx \frac{\beta}{i!}e^{-\beta}$.

| | $\beta=0.1$ | $\beta=0.5$ | $\beta=1$ | $\beta=1.5$ | $\beta=2$ | $\beta=2.5$ | $\beta=3$ |
|---|---|---|---|---|---|---|---|
| $b=1$ | 4.8 | 21.3 | 36.8 | 48.2 | 56.8 | 63.3 | 68.2 |
| $b=2$ | 0.2 | 3.3 | 10.4 | 18.7 | 27.1 | 34.8 | 41.5 |
| $b=3$ | 0.0 | 0.4 | 2.3 | 6.0 | 10.9 | 16.5 | 22.3 |
| $b=4$ | 0.0 | 0.0 | 0.4 | 1.6 | 3.8 | 6.8 | 10.6 |
| $b=5$ | 0.0 | 0.0 | 0.1 | 0.4 | 1.1 | 2.5 | 4.4 |

*Example.* Storage space for 10,000 elements, number of elements 5,000:

| $b$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $m$ | 10,000 | 5,000 | 3,333 | 2,500 | 2,000 |
| $\beta$ | 0.5 | 1 | 1.5 | 2.0 | 2.5 |
| *kol* | 1065 | 520 | 300 | 190 | 125 |
| free blocks (in %) | 60.7 | 36.8 | 22.3 | 13.5 | 8.2 |

12. Let $ST$ and $BT$ be stacks with the operations Push and Pop.
The following operations are executed:

| | |
|---|---|
| at begin block: | PushBT($topST$); |
| at end block: | $topST \leftarrow$ PopBT; |
| to insert: | $entry.next \leftarrow HT[H(entry.name)]$, |
| | $HT[H(entry.name)] \leftarrow topST$, |
| | PushST($entry$), |
| to search: | sequential search in $ST$, |
| checking uniqueness of names: | |
| | sequential search in $ST$, |
| | the topmost element in $BT$ |
| | gives the termination condition. |

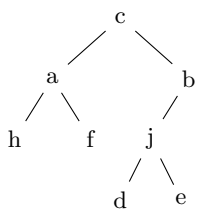
This causes the following problem: The sequential search is not very efficient. However, the efficiency can be improved by using hash methods.

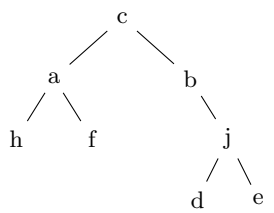
**Stack symbol tables and hash procedures.** As before, let $ST$ and $BT$ be stacks with the operations Push and Pop. To organize concatenated lists of symbol table entries, we extend the symbol table with a column for indices. The hash table $HT$ stores pointers (indices) to entries in the symbol table. Collision resolution for identical hash values is achieved through chaining. This is done using the index column of $ST$.
The following operations are executed:

| | |
|---|---|
| at begin block: | PushBT($topST$); |
| at end block: | delete all names of the block from the |
| | hash organization, $topST \leftarrow$ PopBT; |
| to insert: | $entry.next \leftarrow HT[H(entry.name)]$, |
| | $HT[H(entry.name)] \leftarrow topST$, |
| | PushST($entry$), |
| to search: | search in $ST$ using $H$; |
| checking uniqueness of names: | |
| | search in $ST$ using $H$, the topmost element in $BT$ |
| | gives the termination condition. |

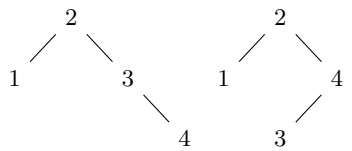
## 4. Trees

1. We denote by $L_1$ the pre-order list and by $L_2$ the post-order list. The root $r$ is the first element in $L_1$ and the last element in $L_2$. Let $x$ be the second element in $L_1$ and $y$ be the second last element in $L_2$.
   For $x \neq y$ then
   (1) $x$ is left successor of the root. $y$ is right successor of the root.
   (2) All elements in $L_1$, which lie between $x$ and $y$ belong to the left subtree.
   (3) All elements in $L_2$, which lie between $x$ and $y$ belong to the right subtree.
   (4) Continue the procedure recursively with the subtrees with root $x$ and $y$.
   For $x = y$ the root $r$ has only one successor $x$. We cannot decide whether it is a left or right successor. Then
   (1) The subtree $L$ with root $x$ contains all elements from $L_1$ which are located to the right of $x$.
   (2) These are also the elements from $L_2$, which lie to the left of $x$.
   (3) Continue the procedure recursively with each of the alternatives.

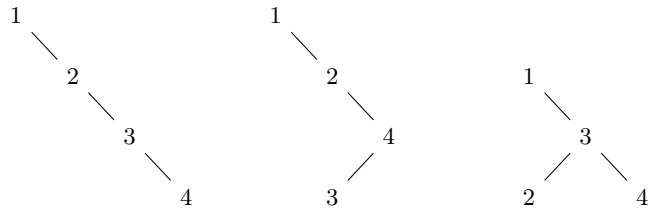


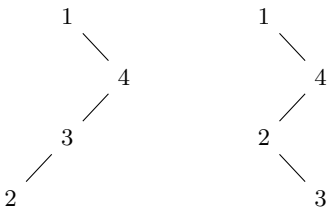




2. With root 2 the following cases occur:




With root 1 the following cases occur:






The cases root 3 and root 4 are symmetrical to root 2 and root 1. In total there are 14 different arrangements for 24 different inputs.

3. The postfix notation is represented by the post-order output, the prefix notation by the pre-order output and the infix notation is generated by the in-order output. Parenthesis must be set into the in-order output. On the recursive descent, an opening parenthesis is output after $*$, $+$ and on the way back a closing parenthesis after $+$,$*$. .
Structure of an arithmetic expression:

$$(a+b)*c*((a+b)*c+(a+b+e)*(e+f)).$$

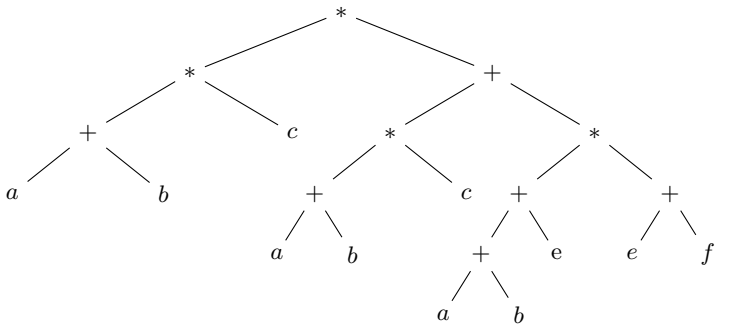



Fig. 2.2: Syntax tree of an arithmetic expression.

4. a. If $v$ has a left successor, then $vv$ is located in the left subtree of $v$ on the far right. This node has no right successor.
   b. In a binary tree and a given node $v$ we denote by $nv$ the node that occurs in the in-order sequence immediately after $v$ (if existing). Show: If $v$ has a right successor $nv$, then $nv$ has no left successor.

5. We define for each subsequence $s_1, s_2, \ldots, s_k$, $k \geq 1$, an interval $I_k$. We set $I_1 = [-\infty, \infty]$ and

$$I_2 = \begin{cases} [s_1, \infty], & \text{if } s_2 \text{ is right successor of } s_1, \\ [-\infty, s_1], & \text{if } s_2 \text{ is left successor of } s_1. \end{cases}$$

Let $k \geq 3$ and $I_k = [b_k, c_k]$ be defined on the basis of $s_1, s_2, \ldots, s_k$. If $s_{k+1} \notin I_k$, then the sequence is not possible. Otherwise, we set

$$I_{k+1} = \begin{cases} I_k, & \text{if } s_k \text{ and } s_{k+1} \text{ are both left or right successors,} \\ [s_k, c_k], & \text{if } s_k \text{ is left and } s_{k+1} \text{ is right successor,} \\ [b_k, s_k], & \text{if } s_k \text{ is right and } s_{k+1} \text{ is left successor.} \end{cases}$$

If $s \in I_{n-1}$, then the sequence is possible.

6. Either $x$ is right or left successor of $y$. Since $x$ is a leaf, $x$ will be placed immediately before $y$ or after $y$ when traversing in-order. Since in-order traversal produces the sorted sequence, the statement follows.

7. We construct a binary search tree for $\{1, \ldots, n\}$ by the following rule: Choose the middle element $x$ as root. Continue the construction recursively for all elements smaller than $x$ and all elements larger than $x$.
The following holds for the height $h(n)$ of the search tree

$$h(1) = 1,\ h(n) \leq h\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + 1 = h\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1.$$

By Proposition 1.28 follows $h(n) \leq \lfloor \log_2(n) \rfloor + 1$.

8. Insert the elements in the "order" of the layers. First the root, then the elements of layer 1 from left to right and so on. The binary search tree property creates the original tree. The AVL condition is fulfilled.

9. The result is a tree whose leaves are contained in at most two levels. After $2^k - 1$ elements are inserted, the lowest layer is fully occupied. For the height $h$ holds $h = \lfloor \log_2(n) \rfloor$.

10. Let $u$ be a node in $\overline{T}$, $u = u_0, u_1, \ldots, u_n$ be a path connecting $u$ with a leaf and $\overline{T}_u$ the subtree of $\overline{T}$ with root $u$. By $hb_u$ we denote the number of black edges in $u_0, u_1, \ldots, u_n$ and by $n_u$ we denote the number of inner nodes of $\overline{T}_u$. Note that $hb_u$ is independent of the choice of the leaf. We show by induction on the height $h_u$ of $u$ that

$$n_u \geq 2^{hb_u} - 1$$

holds. For $h_u = 0$ $u$ is a leaf and $2^{hb_u} - 1 = 2^0 - 1 = 0$ and $n_u = 0$. Let $h_u \geq 1$. The node $u$ has two successors $v$ and $w$. $n_u = n_v + n_w + 1$ and $h_v, h_w < h_u$. We have $hb_v, hb_w \geq hb_u - 1$. From this follows

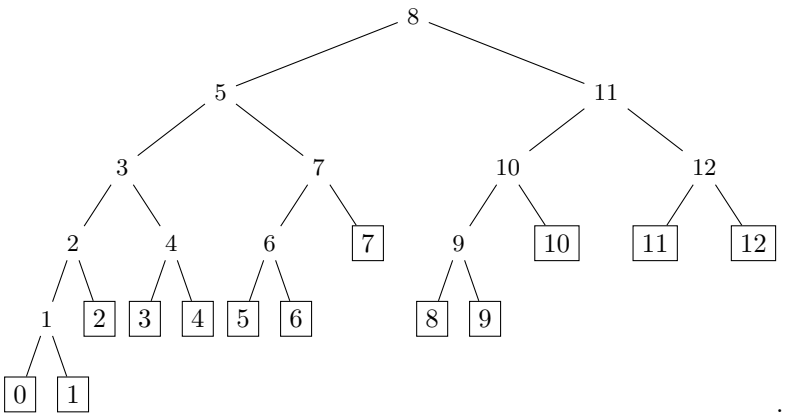
$$n_u = n_v + n_w + 1 \geq 2^{hb_u - 1} - 1 + 2^{hb_u - 1} - 1 + 1 = 2^{hb_u} - 1.$$

This shows the assertion.
Since a red edge is followed by a black edge, for the root $r$ and the height $h$ of $\overline{T}$ holds $hb_r \geq h/2$. It follows that

$$n \geq 2^{hb_r} - 1 \geq 2^{\frac{h}{2}} - 1,$$

which is equivalent to $h \leq 2 \log_2(n+1)$.
Algorithms for insertion and deletion in red-black trees are discussed, for example, in [CorLeiRivSte09, Chapter 13].

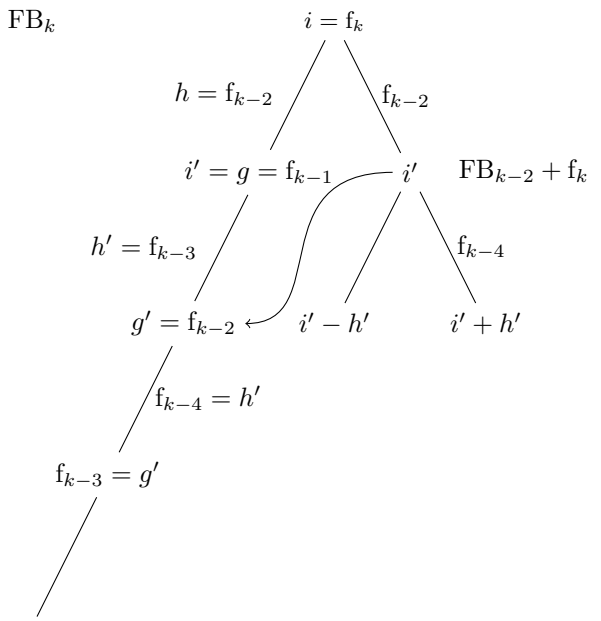
11. Algorithm 4.1 assumes that three consecutive Fibonacci numbers are pre-calculated. This can be done with Algorithm 1.20. Alternatively, the first $\kappa$ Fibonacci numbers can be pre-calculated and stored in the array $fib[0..\kappa]$. From $\mathrm{f}_\kappa \approx \frac{\mathrm{g}^k}{\sqrt{5}} \geq n$, it follows that $k \approx \log_{\mathrm{g}}(n\sqrt{5}) = O(\log_2(n))$. $O(\log_2(n))$ additions are necessary to initialize $fib$ .
The Fibonacci tree $\mathrm{FB}_6$ below shows the search paths for all elements in an array $a[1..n]$, $8 \leq n \leq 12$.




We assume that $n = \mathrm{f}_{k+1} - 1$. Then we use the inner nodes of the Fibonacci tree $\mathrm{FB}_k$. For an inner node $v$, both successors have the same difference with $v$ and this difference is a Fibonacci number. We find the Fibonacci numbers in descending order on the far left of $\mathrm{FB}_k$.

We will derive the navigation using the following sketch.
Initialization: $i = \mathrm{f}_k$, $g = \mathrm{f}_{k-1}$ and $h = \mathrm{f}_{k-2}$.

| | |
|---|---|
| $h > 0$ and $a[i] > x$ : | $g > 1$ and $a[i] < x$ : |
| $i' = i - h$ (left), | $i' = i + h$ (right), |
| $g' = h$, | $g' = g - h$, |
| $h' = g - h$. | $h' = h - g'$. |




$(g, h) = (\mathrm{f}_j, \mathrm{f}_{j-1})$ are two consecutive Fibonacci numbers. If the distance between predecessor $v$ and successor $w$ (in the tree) is equal to $\mathrm{f}_j$, then the next distance is $\mathrm{f}_{j-1}$, if $w$ is a left successor, and $\mathrm{f}_{j-2}$, if $w$ is a right successor.
So the next distance is non-existent for "left" and $h = \mathrm{f}_0 = 0$ and for "right" and $g = \mathrm{f}_2 = 1$ (and $h = 1$ or $h = 0$).
In these cases the element to be searched for is not located in $a$.
If $\mathrm{f}_k < n + 1 < \mathrm{f}_{k+1}$ is true, the case $i' > n$ may occur. Then a step to the left and no access to $a$ will occur.

Let $k+1$ be the index of the smallest Fibonacci number for which $\mathrm{f}_{k+1} \geq n+1$ holds. When calling FibSearch, the array $a[1..n]$, in which the search is to be carried out, the element $x$ to be searched for and the index $k$ must be passed.

**Algorithm 4.1.**

```
FibSearch(item a[1..n], x; index k)
 1  int i ← f_k, g ← f_{k−1}, h ← f_{k−2}
 2  while true do
 3      if i > n
 4        then i ← i − h, t ← g, g ← h, h = t − h
 5        else if a[i] < x
 6                then if g = 1 then return −1
 7                     i ← i + h, g ← g − h, h = h − g
 8        else if a[i] > x
 9                then if h = 0 then return −1
10                     i ← i − h, t ← g, g ← h, h = t − h
11        else if a[i] = x then return i
```

We divide an array of length $\mathrm{f}_k - 1$ into arrays of length $\mathrm{f}_{k-1} - 1$, 1 and $\mathrm{f}_{k-2} - 1$. We set up a recursive equation for comparisons with array elements.

$$C(\mathrm{f}_k - 1) \le C(\mathrm{f}_{k-1} - 1) + 1$$

We set $n = \mathrm{f}_k - 1$ and $x_k = C(\mathrm{f}_k - 1)$ and get $x_k = x_{k-1} + 1 = \sum_{i=2}^{k} 1 = k - 1$. $n + 1 = \mathrm{f}_k \approx \frac{\mathrm{g}^k}{\sqrt{5}}$.

Thus $k \approx \log_g(\sqrt{5}(n+1))$ and

$$C(n) \approx \log_g(\sqrt{5}(n+1)) - 1 = O(\log_2(n)).$$

See also Proposition 4.14, which estimates the height of a balanced tree with $n$ nodes.

12. Since a treap is uniquely defined by the stored elements, the treap does not depend on which insert and delete operations it was created by.

13. Let $c_n$ be the number of binary trees with $n$ nodes. There is exactly one binary tree with 0 nodes. Let $B$ be a binary tree with $n$ nodes $n \ge 0$. The left subtree of the root has $j$ nodes, the right subtree has $n - j - 1$ nodes. Therefore, the following recursion results

$$\begin{aligned} c_0 &= 1, \\ c_n &= \sum_{j=0}^{n-1} c_j c_{n-j-1}. \end{aligned}$$

We consider the generating function

$$\begin{aligned} G(z) &= \sum_{i=0}^{\infty} c_i z^i = \sum_{i=0}^{\infty} \sum_{j=0}^{i-1} c_j c_{i-j-1} z^i \\ &= zG(z)^2 - 1. \end{aligned}$$

The equation $x^2 - \frac{1}{z}x - \frac{1}{z}$ has the solution $x = {}^1\!/_{2z}(1 - \sqrt{1-4z})$. We develop ${}^1\!/_{2z}(1 - \sqrt{1-4z})$ with the binomial series and get

$$\begin{aligned} G(z) &= \frac{1}{2z}(1 - 4\sqrt{1-4z}) = \frac{1}{2z}\left(1 - \sum_{n=0}^{\infty}\binom{\frac{1}{2}}{n}(-4z)^n\right) \\ &= 2\sum_{n=0}^{\infty}\binom{\frac{1}{2}}{n+1}(-4z)^n = \sum_{n=0}^{\infty}\binom{-\frac{1}{2}}{n}\frac{(-4z)^n}{n+1} \\ &= \sum_{n=0}^{\infty}\binom{2n}{n}\frac{z^n}{n+1}\,. \end{aligned}$$

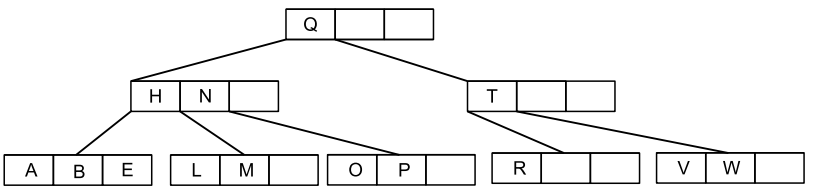
From this follows $c_n = \frac{1}{n+1}\binom{2n}{n}$ .

14. There are pages with 4 and with 2 elements. Therefore, $d \geq 5$ and $\left\lfloor \frac{d-1}{2} \right\rfloor \leq 2$ holds. So $d = 5$ or $d = 6$ follows.

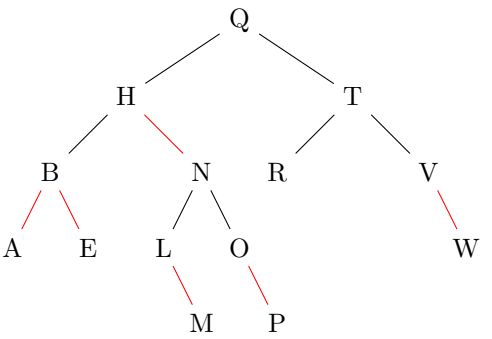
15. a. $3 \leq d - 1$, $\left\lfloor \frac{d-1}{2} \right\rfloor \leq 1$, hence $d = 4$.
    b. Swap h with l, delete h:
    Layer 0: m, layer 1: e, t and x, layer 2: b, f, l, r, u, v, w and y.
    Delete l, delete b: Underflow in layer 2, merge:
    Layer 0: m, layer 1: t and x, layer 2: e, f, r, u, v, w and y.
    Underflow in layer 1, balance:
    Layer 0: t, layer 1: m and x, layer 2: e, f, r, u, v, w and y.

16. It is easy to find a counterexample: Let $d = 5$.
    a. Level 0: 11.50 and level 1: 2.3; 12.17; 52.53.
    b. level 0: 17 and level 1: 2,3,11,12; 50,52,53.

17. We assign to the B-tree




its red-black tree.




Let $B$ be a B-tree and $R_B$ be the assigned red-black tree. An element in a B-tree page is followed by B-tree edges. The edges following a red edge in $R_B$ are black, thus, a red edge in a path is followed by a black edge. Since the leaves in the B-tree are located on one level, all paths leading from a node to a leaf have equal length. Therefore, for each node $v \in R_B$ the number of black edges is the same for all paths that start in $v$ and end in a leaf.

Let $R$ be a red-black tree. We assign to $R$ the B-tree $B_R$. Since the successors of a red edge are black, at most three nodes are connected by (two) red edges. We define a B-tree side by nodes connected by a red edge. The black edges become edges between the B-tree sides. Since in a red-black tree all paths from the root to a leaf have the same number of black edges, all leaves in the assigned B-tree are on the same level. The construction results in a B-tree of order 4.

18. a. $C_1$ is not uniquely decodable, because bbaa‖c‖dea‖bbd = bb‖aacde‖abbd.
    b. $C_2$ is uniquely decodable because the reverse code {c, bb, dbb, aed, ebba, daab, aabb, edcaa} is prefix-free.

19. Let $Y = \{y_1, \ldots, y_n\}$. The code tree for $Y^* = \cup_{i=0}^{\infty} Y^i$ has $n^i$ nodes in the $i$-th level. In the $i$-th level the $n_k$ nodes, which originate from the $k$-th level, $k = 1, \ldots, i-1$, have $n_k n^{i-k}$ successors. For a prefix code $C$ these nodes cannot be included in the code tree of $C$. The nodes of code words of length $i$ are a subset of the remaining $n^i - n_1 n^{i-1} - \ldots - n_{i-1} n$ nodes. Therefore,

$$
\begin{aligned}
n_1 &\leq n \\
n_2 &\leq n^2 - n_1 n \\
n_3 &\leq n^3 - n_1 n^2 - n_2 n \\
&\vdots \\
n_i &\leq n^i - n_1 n^{i-1} - \ldots - n_{i-1} n \\
&\vdots \\
n_l &\leq n^l - n_1 n^{l-1} - \ldots - n_{l-1} n
\end{aligned}
$$

20. a. The given code is compact because it can be generated with Algorithm 4.43.
    b. The result of Algorithm 4.43 is not unique. By selecting two nodes from the set of the nodes with the lowest probability, you can even produce codes with different word lengths.
    C = {00,010,011,100,101,110,111} is also a Huffman code. For both codes the average code word length is $2\frac{5}{7}$.

21. If you encode over an alphabet with $q$ symbols, you can encode $q$ messages with one symbol. For a code word, $q$ extensions can be created by adding a symbol to it. Therefore, in a step the $q$ nodes with the lowest probabilities are summarized. In order for the algorithm to terminate with $q$ messages, it may be necessary to accomplish $X$ with messages with probability 0, so that $|X| = q - k(q-1)$ is valid.

22. We have $H(X) = l(C)$. By the Noiseless Coding Theorem of Shannon (Proposition 4.39), it follows that the code is compact.

23. From $\sum_{i=1}^{k} p_i = 1$, it follows that $p_i = \frac{1}{2^i}$ for $i = 1, \ldots, k-1$ and $p_k = \frac{1}{2^{k-1}}$. Let

$$
c_1 = 0, c_2 = 10, c_3 = 110, \ldots, c_{k-1} = \underbrace{1 \ldots 1}_{k-2} 0, c_k = \underbrace{1 \ldots 1}_{k-1}.
$$

    Then

$$
l(C) = \sum_{i=1}^{k-1} \frac{i}{2^i} + \frac{k-1}{2^{k-1}} = 2 - \frac{1}{2^{k-2}}.
$$

    From $H(X) = l(C)$, it follows by the Noiseless Coding Theorem of Shannon (Proposition 4.39) that the code is compact.

24. Each compact code is a Huffman code. Therefore, we use Algorithm 4.43 for constructing a compact code. The sum of the word lengths $\sum_{i=1}^{n} l_i$ is the greatest if in each construction step a code word which has maximum length is extended by 0 or 1. This results in the code word lengths $1, 2, 3, \ldots, n-1, n-1$. For these code word lengths

$$\sum_{i=1}^{n} l_i = \sum_{i=1}^{n-1} i + n - 1 = \frac{(n-1)(n+2)}{2} = \frac{1}{2}(n^2 + n - 2).$$

(Appendix B, page 330).

25. a. The code for message $acfg$ is 1709.
    b. The number 1688 encodes the message acfaeb.

26. Let $X = \{x_1, \ldots, x_{2^l}\}$, $\mathrm{p}(x_i) = 1/2^l$.

$$I(x_{i_1} \ldots x_{i_n}) = \left[\frac{j}{2^{nl}}, \frac{j+1}{2^{nl}}\right[, j = 0, \ldots, 2^{nl} - 1.$$

Encode $j$ with $nl$ bits: $j_{nl-1} \ldots j_k \ldots j_0$, $j_{nl-1} = \ldots = j_k = 0$, $j_{k-1} = 1$ and

$$x_{i_1} \ldots x_{i_n} \to j_{k-1} \ldots j_0.$$

27. $\mathrm{p}(a) = 1 - 1/2^k$, $\mathrm{p}(b) = 1/2^k$. We have $I(b) = [.\underbrace{1 \ldots 1}_{k}, \overline{1}[$.

By induction, it follows that $I(b^n) = [.\underbrace{1 \ldots 1}_{nk}, \overline{1}[$:

We show the step from $n$ to $n+1$:

$$\begin{aligned} I(b^{n+1}) &= .\underbrace{1 \ldots 1}_{nk} + .\underbrace{0 \ldots 0}_{nk}\overline{1}[.\underbrace{1 \ldots 1}_{k}, \overline{1}[ \\ &= [.\underbrace{1 \ldots 1}_{(n+1)k}, \overline{1}[. \end{aligned}$$

$$\begin{aligned} I(b^n a) &= .\underbrace{1 \ldots 1}_{nk} + .\underbrace{0 \ldots 0}_{nk}\overline{1}[0, .\underbrace{1 \ldots 1}_{k}[ \\ &= [.\underbrace{1 \ldots 1}_{nk}, .\underbrace{1 \ldots 1}_{(n+1)k}[ \end{aligned}$$

Encode

$$b^n a \to .\underbrace{1 \ldots 1}_{nk}.$$

28. For example $r = 5$ and

$$abcdefabcdefabcdefabcdef \ldots \in \{a, b, c, d, e, f\},$$

require that each character must be encoded as a single character. There is maximum expansion.

## 5. Graphs

1. If $G$ has an Euler cycle, $G$ is connected. An Euler cycle, which enters a node, must also exit the node. Therefore, the degree of each node is even. To show the opposite direction we consider a path

   $$P : v_0, \ldots, v_k$$

   of maximum length which does not contain an edge multiple times. Since this is a path of maximum length, all edges that are incidences to $v_k$ are on that path. Since with each occurrence of $v_k$ inside $P$ two incident edges are connected, it follows from $v_0 \neq v_k$ that $\deg(v_k)$ is odd. Therefore, $v_0 = v_k$ holds. We show that all edges of $G$ occur in $P$.
   Assume that there is an edge $e$ that is not on $P$. Then, since $G$ is connected, there is an edge $e$, which is incident to a node of $P$ and is not on $P$. Let $e = \{v_i, u\}$, $0 \leq i \leq k$. Then

   $$u, v_i, v_{i+1}, \ldots, v_{k-1}, v_k = v_0, v_1 \ldots, v_{i-1}, v_i$$

   is a path that does not contain an edge multiple times and has the length $k + 1$. A contradiction to the choice of $P$.

2. We prove the statement by induction on the number of edges $E$. For $|E| = 0$ we have $|V| = 1$ and $|F| = 1$. The formula is correct. Let $|E| \geq 1$. If the graph is a tree, then $|E| = |V| - 1$ and $|F| = 1$. The formula is correct.
   Now let $Z$ be a cycle and $e$ an edge on $Z$. Let $G' = G \setminus \{e\}$ be the graph, which is created from $G$ by removing the edge $e$. Then $|E'| = |E| - 1$. Since there is a different area on each side of $e$, which merge when $e$ is removed, thus we have $|F'| = |F| - 1$. By the induction hypothesis, the following holds for $G'$:

   $$|V'| - |E'| + |F'| = 2,$$

   hence

   $$|V| - |E| + |F| = |V'| - (|E'| + 1) + (|F'| + 1) = 2.$$

3. Let $G = (V, E)$ be a graph, $V = \{1, 2, \ldots, n\}$ and let $col = \{1, 2, \ldots, n\}$ be the color numbers of pairwise distinct colors.

**Algorithm 5.1.**

```
col[1..n]; node adl[1..n]

colourNodes()
 1  node no; set availableCol
 2  col[1] ← 1
 3  for k ← 2 to n do
 4      col[k] ← 0
 5  for k ← 2 to n do
 6      availableCol ← {1, ..., n}
 7      no ← adl[k]
 8      while no ≠ null do
 9          availableCol ← availableCol \ {col[no.v]}
10          no ← no.next
11      col[k] ← min availableCol
```

We color the nodes one after the other. In each step, the colors used for the neighbors of the node $k$ are excluded. $l = \min availableCol$ is greatest if these are the colors with the color numbers $1, \ldots, \deg(k)$. Therefore, $l \leq \deg(k) + 1 \leq \Delta + 1$, $k = 1, \ldots, n$ holds. For a complete graph, the barrier is accepted. The running time is of the order $O(n + m)$.

4. Let $m$ be the number of edges of $G = (V, E)$. Then

$$\sum_{v \in V} \deg(v) = 2m, \text{ thus, } \sum_{v \in V, \deg(v) \text{ is odd}} \deg(v)$$

is even. Since the sum of an odd number of odd summands is odd, the number of odd summands is even.

5. Let $[x] = [\tilde{x}]$ and $[y] = [\tilde{y}]$ be given. 21 divides $x - \tilde{x}$ and $\tilde{y} - y$. So 21 also divides $x - \tilde{x} + \tilde{y} - y = x - y - (\tilde{x} - \tilde{y})$. Thus, 7 also divides $x - y - (\tilde{x} - \tilde{y})$. Therefore it holds: 7 divides $x - y$ if and only if 7 divides $\tilde{x} - \tilde{y}$.Thus, the definition does not depend on the choice of the representative.
The equivalence classes are the strongly connected components.

6. a. We prove by induction on $m := |E|$.
(1) Induction basis: $m = 0 \Longrightarrow n = 1$ (since $G$ is connected). Therefore, the formula is true. (2) "$< m \Longrightarrow m$": We remove an edge in $G$. Either $G$ remains connected. In that case, according to the induction prerequisite $m - 1 \geq n - 1$ holds, so $m \geq n - 1$ also holds. In the other case the graph breaks down into two components $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$. According to induction hypothesis:
$|E_1| \geq |V_1| - 1$ and $|E_2| \geq |V_2| - 1 \Longrightarrow |E| = |E_1| + |E_2| + 1 \geq |V_1| + |V_2| - 1$.
   b. We prove by induction on $n = |V|$.
(1) Induction basis: $n = 1 \Longrightarrow G$ connected. Therefore, the statement is correct.

(2) $n \Longrightarrow n+1$: Let $n \geq 2$ and lets assume $|E| \geq \binom{n}{2} + 1$. We show that $G$ is connected. If the graph is complete, it is also connected. If the graph is not complete, there is a node $v$ with less than $n-1$ incident edges. We remove $v$ and all incident edges of $v$ in $G$ and get $G' = (V', E')$. Then

$$|E'| \geq |E| - (n-1) \geq \binom{n}{2} + 1 - (n-1) \geq \binom{n-1}{2} + 1.$$

According to induction hypothesis, $G'$ is connected. Between the nodes of $V \setminus \{v\}$ there are at most $\binom{n-1}{2}$ edges. Therefore, the remaining edge has one endpoint in $E'$ and the other endpoint is $v$. Thus, $G$ is connected.

7. a $\Longleftrightarrow$b: If $G$ is a tree, $G$ is acyclic and has $|V|-1$ edges (see Proposition 5.7).
   Let $G$ be acyclic with $|V|-1$ edges. Suppose $G$ is not connected. Let $G_1, \ldots, G_r$, $r \geq 2$, the connected components of $G$. By adding $r-1$ edges $G$ becomes connected and stays acyclic. $G$ is a tree with $|V|-1+r-1 > |V|-1$ edges, a contradiction (see Proposition 5.7).
   a $\Longleftrightarrow$c: Let $G = (V, E)$ be a tree (especially $G$ is connected). Let $e = \{v, w\}$ an additional edge. In $G$ there is a path $P$ from $v$ to $w$. Let us extend $P$ by $\{w, v\}$ so we get a cycle. If, for the opposite direction $v, w \in V$ are given. If we get a cycle by adding the edge $\{w, v\}$, then there must have been a path from $v$ to $w$. Thus, $G$ is connected.
   a $\Longleftrightarrow$d: Let $G = (V, E)$ be a graph and $e = \{v, w\} \in E$. For $G$ holds, $G \setminus \{e\} = (V, E \setminus \{e\})$ decomposes into at most two components. $G \setminus \{e\}$ is connected if and only if there are cycles in $G$. $G \setminus \{e\}$ decomposes into at least two components if $G$ is acyclic. So the statement in point d holds for a tree. If after removing $e$ for all edges $e$, a graph decomposes into two components, the graph is acyclic. Thus, $G$ is a tree.
   a $\Longleftrightarrow$e.: $G$ is connected if and only if there is at least one path between every two nodes. $G$ is acyclic if and only if there is at most one path between two nodes.

8. Let $v \in V$. Either there are three nodes adjacent to $v$ or there are three nodes, who are not adjacent to $v$.
   Let us look at the first case. Either there are two nodes among the three, which are mutually adjacent, then these two form a group with $v$ of three nodes and each two nodes from this group are adjacent. Or two nodes of each triplet are not adjacent. Then this is a triplet and two nodes of this group are not adjacent.
   The second case - there are three nodes, which are not adjacent to $v$ - can be treated with an analog argumentation.

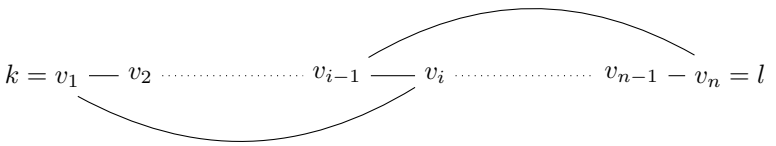
9. Suppose the statement is not correct. We consider a graph $G$ with $n$ nodes with a maximum number of edges for which the statement does not hold, i.e., for non-adjacent nodes $k$ and $l$ holds

$$(*) \qquad \deg(k) + \deg(l) \geq n$$

but $G$ has no Hamilton circuit.
Let $G' = G \cup \{k, l\}$. In $G'$ the condition $(*)$ holds and for $G'$ the statement is correct. Thus, there is a cycle $Z$ in $G'$, which contains each node exactly once. This cycle must contain the edge $\{k, l\}$. Let $P$ be the path that we get, when we remove the edge $\{k, l\}$ in $Z$.

$$P : k = v_1, \ldots, v_n = l.$$

We will immediately show that for each node $v_i$, $i = 2, \ldots, n$, which is adjacent to $k$, $v_{i-1}$ is not adjacent to $l$. From this condition follows $\deg(l) \leq n - 1 - \deg(k)$, i.e. h. $\deg(k) + \deg(l) \leq n - 1$, a contradiction to the condition $\deg(k) + \deg(l) \geq n$. The statement of the exercise is therefore correct.
We now show the above condition. Let us assume that there is an $i$ with $v_i$ being adjacent to $k$ and $v_{i-1}$ being adjacent to $l$.




Then

$$v_1, v_i, \ldots, v_n, v_{i-1}, v_{i-2}, \ldots, v_1$$

is a cycle in $G$, which contains each node of $G$ exactly once, a contradiction.

10. The algorithm sorts the intervals by the starting points and uses an additional list of active nodes (intervals). The list of active nodes is sorted by endpoints.
1. Sort the list $V$ of intervals by the starting points.
2. Process the sorted list $V$. Each step consists of: (1) Add the node $I$ to the list of active nodes. (2) Remove all nodes whose end points are before the starting point of $I$ from the list of active nodes. (3) All nodes remaining in the list of active nodes are adjacent to $I$.

11. The problem can be solved by searching for the shortest path between two nodes in a graph. Thus, we model the situation with a graph. A node is given by a subset of {m,k,w,z} (m = man, k = cabbage, w = wolf, z = goat) consisting of the members which are on this side of the river. First all members of {m,k,w,z} are located on this side of this river. Each departure or arrival of the boat defines a new subset. Permissible subsets are those subsets, that do not lead to a catastrophe (goat eats cabbage head, wolf eats goat). Between permissible subsets $z_1$ and $z_2$ an edge is drawn if a transition from $z_1$ to $z_2$ by departure or arrival of the

boat is possible. A path is searched that connects {m,k,w,z} and {}. A shortest way can be found with breadth-first search. A solution can be simply found (without graphs).

a. The man goes over with the goat (this is the only possibility).
b. He rows back alone.
c. The man crosses over with the wolf.
d. He rows back with the goat.
e. The man crosses over with the head of cabbage.
f. He rows back alone.
g. The man crosses over with the goat.

12. We replace the queue in Algorithm 5.11 with a stack. The visit order does not always completely match the visit order of the recursive version.

**Algorithm 5.2.**

```
DepthFirstSearch()
 1  vertex k, int nr
 2  for k ← 1 to p do
 3       where[k] ← 0; parent[k] ← 0
 4  nr ← 1
 5  for k ← 1 to p do
 6       if where[k] = 0
 7         then Visit(k); nr ← nr + 1

Visit(vertex k)
 1  node mo; no = adl[k]
 2  if no ≠ null
 3    then Push(k, no); where[k] ← −1
 4  repeat
 5       (k, no) = Top()
 6       while where[no.v] ≠ 0 and no ≠ null do
 7            no ← no.next
 8       (k, mo) = Pop()
 9       if no ≠ null
10         then parent[no.v] ← k; Push(k, no)
11              mo = adl[no.v]
12              if mo ≠ null
13                then Push(no.v, mo); where[no.v] ← −1
14         else  where[k] ← nr
15  until QueueEmty
```

13. For a graph, BFS only produces tree edges and cross edges (no backward and no forward edges). A possible backward or forward edge would mean that a node of a lower level is adjacent to a higher level (height difference at least 2). In this case, however, the node of the lower level appears as an immediate successor of the node of the higher level.

In a graph, DFS does not produce cross edges. A possible cross edge inserts one of the two end nodes of the edge as successor of the other. Edges between predecessor and successor in the DFS tree are both forward and backward edges.
In a directed graph, BFS produces both cross and back edges (no forward edges) in addition to tree edges. A possible forward edge would mean that a node of a lower level is adjacent to a higher level (height difference at least 2). In this case, however, the node of the lower level appears as an immediate successor of the node of the higher level.
In a directed graph, DFS produces all kinds of edges.

14. In the arrangements of (1) and (3) G comes before C, therefore they are not topological sorted.
The arrangement (2) can be created by the calls Visit(H), Visit(C), Visit(J) and Visit(A). Thus, it is a topological sorting.
If $v_{i_1}, \ldots, v_{i_n}$ is a topological sorting, then the calls of Visit with the parameters in the order $v_{i_n}, \ldots, v_{i_1}$ result in in the given order, the original sort order.

15. In a topological sorting, a node is at the first position. No edge goes into this node. This is the only node with this property if and only if all other nodes can be reached from it.

16. Suppose that in $G_{red}$ there are mutually accessible nodes $V_i$ and $V_j$. Then there would be $v_1, v_2 \in V_i$ and $w_1, w_2 \in V_j$ and edges $(v_1, w_1)$, $(w_2, v_2)$ in $G$. But then $v_1$ and $w_1$ would also be mutually accessible. A contradiction.

17. a. We prove the statement by induction on $r$:
For $r = 1$ the statement follows from the definition of the adjacency matrix. We now show the step from $r$ to $r+1$. A path of length $r+1$ from $i$ to $j$ consists of a path of length $r$ from $i$ to $k$ and an edge $(k, j)$. According to the induction hypothesis, $A^r[i, k]$ is the number of paths of length $r$ from $i$ to $k$. Therefore, $\sum_{k=1}^{n} A^r[i, k]A[k, j]$ is the number of paths of length $r + 1$ and

$$\sum_{k=1}^{n} A^r[i, k]A[k, j] = A^{r+1}[i, j]$$

according to the definition of the die matrix product.

b. We consider $A$ as adjacency matrix of a directed graph. By depth-first search a directed graph can be tested for cycles (Proposition 5.15). The running time is in the order $O(n + m)$.

18. a. The graph of the base relations:

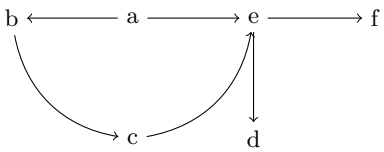
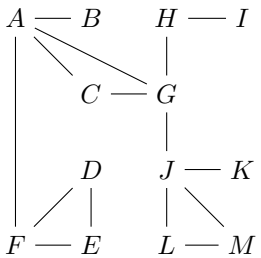



Fig. 2.3: Base relations of an arrangement.

b. Due to transitivity, a cycle $n_1 < n_2 < \ldots < n_1$ implies $n_1 < n_1$, a contradiction. Thus, the graph is acyclic. Each node forms a strongly connected component.
c. We obtain all elements $b \in M$ with $b > a$ by depth-first search in $G$ with starting point $a$ and all elements $b \in M$ with $b < a$ by depth-first search in $\text{rev}(G)$, the reverse graph of $G$, with starting point $a$.

19. a. The articulation points of the following graph




are the nodes A, F, H, G, J.
A graph with one articulation point has at least three nodes.
b. Let $v$ be an articulation point in $G$. Then $G \setminus \{v\}$ breaks down into at least two connected components. If $u$ is selected in one component and $w$ in the other, then all paths from $u$ to $w$ in $G$ pass through $v$. Conversely, if there are nodes $u$ and $w$ in $G$, so that all paths from $u$ to $w$ in $G$ pass through $v$, then the node $u$ is no longer reachable from $w$ after we remove $v$. $G \setminus \{v\}$ has at least two connected components.
c. Since no cross edges occur in the DFS tree of $G$, the root is an articulation point if and only if it has two successors.
d. If $v$ is an articulation point, $G \setminus \{v\}$ is divided into at least two connected components. Thus, there is a son $v'$ of $v$ and for all nodes $u$ in $T$ accessible from $v'$, there is no backward edge $(u, w)$ to a predecessor $w$ of $v$. Vice versa, if there is a son $v'$ of $v$ and for all nodes $u$ in $T$ reachable from $v'$ there is no backward edge $(u, w)$ to a predecessor $w$ of $v$, then $G$ is not connected without $v$. i.e., $v$ is an articulation point.
e. If $v$ is an articulation point and $v'$ is a son of $v$ and for all nodes $u$ in $T$ reachable from $v'$ there is no backward edge $(u, w)$ to a predecessor $w$ of $v$, then it follows that $low(v') \geq t_b(v)$. If there is a son $v'$ of

$v$ with $low(v') \geq t_b(v)$, then there can be no backward edge from a node reachable from $v'$ to a predecessor of $v$. Altogether, the following holds: $v$ is an articulation point if and only if there is a son $v'$ of $v$ with $low(v') \geq t_b(v)$. We formulate the calculation of $low(v)$ recursively:

$$low(v) = min(\{t_b(v)\} \cup \{t_b(w) | (v,w) \in R\} \cup \{low(v') | v' \text{ is son of } v\}).$$

Therefore, the condition $low(v') \geq t_b(v)$ can be checked with depth-first search, analogously to Tarjan's algorithm for calculating the strongly connected components. For this purpose, Algorithm 5.19 can be easily adapted.

20. Let $u, v \in V$. If there is a circle that contains $u$ and $v$, there are two disjoint paths from $u$ to $v$. If we remove $w \notin \{u, v\}$, $u$ and $v$ remain mutually accessible.
We prove the opposite direction by induction on $d(u, v)$.
Induction Statement: For each node $v$ there is a circle with start and end node $u$ on which $v$ lies.
Let $u, v \in V$. We run DFS with the start node $u$. There is a path $P$ from $u$ to $v$ in the DFS tree $T$ of $G$.
Induction start $d(u, v) = 1$: $v$ is not an articulation point. Since there are no cross edges in $T$, there is a backward edge from a successor of $v$ to node $u$. With the edge $u, v$ we obtain a cycle in $G$ on which $v$ is located.
For the induction step, assume $d(u, v) \geq 2$: Let $w$ be the predecessor of $v$ in $T$. Since $d(u, w) < d(u, v)$, there is a cycle $Z$ with start node $u$ containing $w$. Since $w$ is not an articulation point, there is a path $u = v_0, \ldots, v_l = v$ from $u$ to $v$ in $G \setminus \{w\}$. Let $i$ be the largest index with $v_i \in Z$. We get a cycle $Z'$ with start node $u$: We follow $Z$ from $u$ to $v_i$, then $v_{i+1}, \ldots, v_l = v, w$ then $Z$ from $w$ to $u$. $Z'$ is a cycle with start and end node $u$ and contains $v$.

## 6. Weighted Graphs

1. a. Induction basis: $A(1,0) = A(0,1) = 2$.
   Induction hypothesis: $A(1,n) = n+2$.
   Induction statement: $A(1,n+1) = (n+1)+2$.
   Induction step: $A(1,n+1) = A(0,A(1,n)) = A(0,n+2) = (n+1)+2$.
   b. Induction basis: $A(2,0) = A(1,1) = 1+2 = 3$.
   Induction hypothesis: $A(2,n) = 2n+3$.
   Induction statement: $A(2,n+1) = 2(n+1)+3$.
   Induction step:
   $A(2,n+1) = A(1,A(2,n)) = A(1,2n+3) = 2n+3+2 = 2(n+1)+3$.
   c. Induction basis: $A(3,0) = A(2,1) = 2+3 = 5$.
   Induction hypothesis: $A(3,n) = 2^{n+3} - 3$.
   Induction statement: $A(3,n+1) = 2^{(n+1)+3} - 3$.
   Induction step:
   $A(3,n+1) = A(2,A(3,n)) = A(2,2^{n+3}-3) = 2 \cdot (2^{n+3}-3) + 3 = 2^{(n+1)+3} - 3$.
   d. Induction basis:

$$A(4,0) = A(3,1) = 2^4 - 3 = 13.$$

$$\underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}}_{n+3 \text{ times}} - 3 = 2^{2^2} - 3 = 16 - 3 = 13.$$

Induction hypothesis:

$$A(4,n) = \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}}_{n+3 \text{ times}} - 3$$

Induction statement:

$$A(4,n+1) \ \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}}_{n+4 \text{ times}} - 3.$$

Induction step:

$$A(4,n+1) = A(3,A(4,n)) \ = \ A(3, \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}}_{n+3 \text{ times}} - 3)$$

$$= 2 \ \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}}_{n+3 \text{ times}} \ ^{-3+3} - 3 \ = \ \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}}_{n+4 \text{ times}} - 3.$$

2. a. The axiom "positive definite" and the triangle inequality are violated.
   b. Dijkstra's algorithm returns the path 4 - 3 - 1 - 2 of length 4. The path 4 - 5 - 2 has length 3. Dijkstra's algorithm therefore does not give a correct result with negative weights.
   c. Kruskal's algorithm yields a minimum spanning tree of weight 2 with the edges (1,2), (1,3), (2,5) and (3,4).
   In general, this is true , because the proof of correctness is also valid if edges with negative weight are present. The Proof uses only comparison arguments, which are also valid for negative edges.

3. We assume that node 1 is the start node and node $n$ is the end node. No edge enters the start node and no edge exits the end node. Further we assume that $n$ is accessible from 1.
   The following algorithm calculates a longest path from 1 to $n$. We use the arrays $succ[1..n]$ and $len[1..n]$, which are initialized with 0. The array $succ[k]$ stores the successor of a node on a longest path from $k$ to $n$ and $len[k]$ stores the length of a longest path from $k$ to $n$.

   **Algorithm 6.1.**
   *vertex parent*$[1..n]$, *succ*$[1..n]$; *node adl*$[1..n]$; *boolean visited*$[1..n]$

```
Visit(vertex k)
 1  node no
 2  visited[k] ← true
 3  no ← adl[k]
 4  while no ≠ null do
 5      if visited[no.v] = false
 6        then parent[no.v] ← k; Visit(no.v)
 7             if len[no.v] + no.w > len[k]
 8               then len[k] ← len[no.v] + no.w; succ[k] = no.v
 9      no ← no.next
```

   After termination of the call Visit(1), $len[1]$ contains the length of a critical path and it can be displayed with the array *succ*.

4. The problem of calculating the distances from a node $s$ to all nodes in $G$ can be solved by depth-first search in time $O(n + m)$ for an acyclic graph.
   We start the depth-first search in $s$ and create the DFS tree. In each node $k$ we record the distance between $s$ and $k$ in the DFS tree. If $k$ is the end node of a cross or forward edge, we note the new distance of $s$ for $k$, which results from this. Since $G$ is acyclic, no backward edges occur.
   We again apply depth-first search in the DFS tree and calculate the distances including the distances which originate from cross and forward edges. In the DFS tree, depth-first search is performed with the running time $O(n)$. Altogether, we get the running time $O(n + m)$.

5. If all edges have different weights, then the minimum spanning tree and the result of Kruskal's algorithm are unique.
   If edges have the same weight, then the degree of freedom of Kruskal's algorithm results from the choice of the order of edges of equal weight. We sort the edge list $E$ from $G$ ascending by weight. Then we put the edges of the given minimum spanning tree $T$ in each group of edges of equal weight at the first position of the group. If the edges are processed in this order, the edges from $T$ will not lead to any cycles. They are included in the minimum spanning tree constructed by Kruskal's algorithm.
6. The data structure priority queue generalizes queue and stack. When assigning ascending priorities the priority queue simulates a queue and when assigning descending priorities a stack.
7. a. By adding an edge $e$ to $T$ a cycle $Z$ is created. We determine the next common ancestor of the end nodes of $e$ and by doing this $Z$ (in the time $O(|V_T|)$ (Proposition 6.15)). Alternatively, $Z$ can also be determined by depth-first search (also in the time $O(|V_T|)$.
   We remove the edge of maximum weight $e'$ from $T \cup \{e\}$. The tree $(T \cup e) \setminus \{e'\}$ is a minimum spanning tree of $G \cup \{e\}$.
   b. First add to $G$ and to the MST $T$ the node with the edge of the least weight. We denote the result with $G'$ and $T'$. $T'$ is a minimal spanning tree for $G'$. We add the remaining edges using the method from point a.
8. Let $e$ be the edge of $G$, whose weight is changed by $x$. We distinguish:
   a. The edge $e$ is a tree edge.
      i. $x < 0$. The change has no effect on $T$. We position in Kruskal's algorithm in the sorted sequence of edges, the edge $e$ as the first edge among those with the same weight. Kruskal's algorithm applied to the new sorting order returns $T$.
      ii. $x > 0$. If there is an edge $e' \notin T$ with $w(e) \leq w(e') < w(e) + x$ so that by adding $\{e'\}$ to $T \setminus \{e\}$ is acyclic, $T$ is no longer the MST after the weight change.
   b. The edge $e$ is not a tree edge. $T$ is a minimum spanning tree of $G' = G \setminus \{e\}$. With Exercise 7. a we can now determine the minimum spanning tree of $G = G' \cup \{e\}$.
9. The probability that the edge $(u, v)$ will not fail is $q_{(u,v)} = 1 - \mathrm{p}(u, v)$. The probability that a connection $v_0, \ldots, v_n$ from $v_0$ to $v_n$ will not fail is $\prod_{i=1}^{n} q_{(v_{i-1}, v_i)}$.

$$\log(\prod_{i=1}^{n} q_{(v_{i-1}, v_i)}) = \sum_{i=1}^{n} \log(q_{(v_{i-1}, v_i)}) =: p.$$

   $p$ is maximum if and only if $-p$ is minimum. We weight an edge $(u, v)$ with $-\log(q_{(u,v)})(> 0)$. Then a path that connects two nodes has maximum probability that a connection along the path will not fail if the path

has minimum length in a weighted graph, i.e., is equal to the distance between two nodes in a weighted graph. We calculate the distance using Dijkstra's algorithm.

10. We model the problem with a state graph. The nodes are defined by the people who still have to cross the river and by the position of the torch. For example, $\{P_1, P_2, P_3, P_4, F\}$ is the start state and $\{\overline{F}\}$ is the end state.
For example, if $P_1$ and $P_2$ cross the bridge, an edge from $\{P_1, P_2, P_3, P_4, F\}$ to $\{P_3, P_4, \overline{F}\}$ is to be drawn with weight 10. If $P_1$ runs back, a transition to $\{P_1, P_3, P_4, F\}$ is added.
The problem can be solved with Dijkstra's algorithm. The shortest path: $\{P_1, P_2, P_3, P_4, F\} \to \{P_3, P_4, \overline{F}\} \to \{P_2, P_3, P_4, F\} \to \{P_4, \overline{F}\} \to \{P_1, P_4, F\} \to \{\overline{F}\}$.

11. a. $r(G) = \min_{i=1,\ldots,n} e[i] \leq \max_{i=1,\ldots,n} e[i] = d(G)$. Let $i$ and $j$ be nodes with $d(i,j) = d(G)$ and $k$ a center. Then $d(i,j) \leq d(i,k) + d(k,j) \leq \max_{i=1,\ldots,n} d(i,k) + \max_{j=1,\ldots,n} d(k,j) = e(k) + e(k) = 2r(G)$.
    b. i. Calculate the distance matrix $A[1..n, 1..n]$ for $G$ with the algorithm by Floyd (Algorithm 6.57).
    ii. Calculate the arrays $e[1..n]$, $r(G)$ and $d(G)$:

$$e[i] = \max_{j=1,\ldots,n} d(i,j), r(G) = \min_{i=1,\ldots,n} e[i] \text{ and } d(G) = \max_{i=1,\ldots,n} e[i]$$

    and all $i$ with $e[i] = r(G)$.
    c. Calculate in each step the eccentricity for a node with the algorithm of Dijkstra (Section 6.2). Calculate the minimum $m$ of eccentricities already calculated. In the next node Dijkstra's algorithm can be aborted as soon as a distance $> m$ is found.

12. a. Start with a complete weighted graph $G$ with the cities as nodes and the distances as weights.
    b. Calculate a minimal spanning tree $T$ for $G$.
    c. Calculate the distance matrix $A$ for $T$ and the array $d[1..n]$, defined by $d[i] = \sum_j A[j,i]$.
    d. Determine the minima in $d$ (centers). These fulfill the condition for point c.

13. Let $d(v)$ be the distance between $v$ and the root $r$ in $T$. If for all $u \in V$

$$d(u) + g(u,v) \geq d(v) \text{ for } v \in U(u)$$

holds, every path in $T$ is also a shortest path in $G$.
Assuming there is a node $w$ and a path $s = v_0, \ldots, v_l = w$ in $G$ with $\sum_{i=1}^{l} g(v_i - 1, v_i) < d(w)$. Let $j$ be the smallest index with $\sum_{i=1}^{j} g(v_{i-1}, v_i) < d(v_j)$. The edge $(v_{j-1}, v_j)$ is not a tree edge. For $u = v_{j-1}$ and $v = v_j$, we have $d(u) + g(u,v) < d(v)$, a contradiction.

The condition can be tested by traversing $T$. In each node $u$, $\deg(u)$ comparisons are necessary. Altogether there are $\sum_{u \in V} \deg(u) = 2m$ comparisons.

14. Let $G_T = (V_T, E_T)$ be the transitive closure of $G$. The following conditions are equivalent:
    a. $b$ is satisfiable.
    b. For all nodes $v$, it holds that $(v, \overline{v}) \notin E_T$ or $(\overline{v}, v) \notin E_T$.

    a. $\Longrightarrow$b.: Assuming there is a node $v$ with $(v, \overline{v}) \in E_T$ and $(\overline{v}, v) \in E_T$. Then follows $v \Longrightarrow \overline{v}$ and $\overline{v} \Longrightarrow v$. For a satisfying assignment for $b$, assume $v = 0$ (without restriction). Then $\overline{v} = 1$ and $1 \Longrightarrow 0$ is a contradiction.
    In general, the following applies : $(v, w) \in E_T$ if and only if $(\overline{w}, \overline{v}) \in E_T$.
    b. $\Longrightarrow$a.: We iteratively define a satisfying assignment of the variables $x_1, \ldots, x_n$. Let $v$ be a node for which no assignment is defined and let $(v, \overline{v}) \notin E_T$ (without restriction of the generality, otherwise consider $(\overline{v}, v)$), i.e., in $G$ there is no path from $v$ to $\overline{v}$. Let $Z_v$ be the set of nodes in $G$ that can be reached from $v$. For $w \in Z_v$, we have $\overline{w} \notin Z_v$. Assuming that $w \in Z_v$ and $\overline{w} \in Z_v$, then $(\overline{w}, \overline{v}) \in E_T$ implies $(v, \overline{v}) \in E_T$, a contradiction. For $w \in Z_v$, we set $w = 1$ and $\overline{w} = 0$. We repeat the construction with a node for which no assignment is defined until all nodes have an assignment. This assignment is a satisfying assignment for $b$, because we set the assignment to 1 in each step for all nodes that are accessible from $w$ (the implication $1 \Longrightarrow 0$ cannot occur). To decide whether $b$ is satisfying, we have to check whether $v$ and $\overline{v}$ are not mutually reachable for all $v$. $v$ and $\overline{v}$ are mutually reachable if and only if they are in the same strongly connected component.
    The strongly connected components can be calculated for a directed graph with $n$ nodes and $m$ edges using depth search in the time $O(n+m)$ (see Section 5.6).
    An alternative solution can be obtained by calculating the transitive closure $G_T$ of $G$ and checking whether for all nodes $v$ either $(v, \overline{v}) \notin E_T$ or $(\overline{v}, v) \notin E_T$. If this is the case, $b$ can be fulfilled.

15. The calculated augmenting path is S - H - J - F - B - F. Its augmentation is 5. After the flow increase, only D and H can be reached in the residual graph from S. The maximum total flow is 70 and an section of minimum capacity is {S,D,H}.

16. Let $p_1, \ldots, p_n$ be the processes that are fixed with $P$ and $q_1, \ldots, q_m$, the processes that are fixed with $Q$. The processes still to be assigned are designated by $pq_1, \ldots, pq_l$. We use a weighted graph for our solution. The nodes are $P$, $Q$, $p_1, \ldots, p_n$, $q_1, \ldots, q_m$ and $pq_1, \ldots, pq_l$. We define edges $(P, p_1), \ldots, (P, p_n)$, $(Q, q_1), \ldots, (Q, q_m)$ with the capacity $\infty$. The capacity of the edges between the processes is given by the communication effort between the processes.

The distribution of the processes can now be done by calculating a cut of minimum capacity. This in turn can be traced back to the calculation of a maximum flow and subsequent breadth-first search in the residual graph.

17. We reduce the problem to the flow problem with one source $S$ and one sink $T$.
We extend the graph by a source S, a sink T and edges (S,$S_i$,$c_i$), $i = 1, \ldots, n$, and ($T_i$,T,$d_i$), $i = 1, \ldots, m$. We set $c_i$ equal to the sum of the capacities of the edges, which have $S_i$ as start node and $d_i$ equal to the sum of the capacities of the edges, which end in $T_i$.
Then we apply the algorithm of Ford-Fulkerson in the variant of Edmonds-Karp.

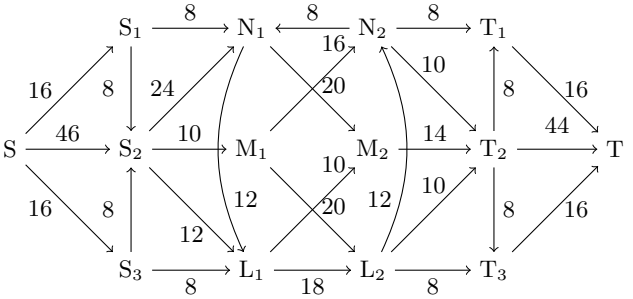



Fig. 2.4: Variante.

18. We assign to the bipartite graph $G = (V \cup W, E)$ a flow network $N = (V \cup W, E, s, t)$ with capacity $c : E \longrightarrow \mathbb{Z}$ and flow $f : E \longrightarrow \mathbb{Z}$. We add two additional nodes $s$ and $t$ to the nodes of $G$. The edges of $E$ become directed edges with the direction from $V$ to $W$. Further we add for each node $v$ from $V$ an edge $(s, v)$ and for each node $w$ from $W$ an edge $(w, t)$. All these edges get the capacity 1.
Let $Z$ be a matching in $G$. For each edge $e = (v, w)$ from $Z$, we define $f(e) = 1$, $f(s, v) = 1$ and $f(w, t) = 1$. Capacity limit and flow conservation are maintained. The number of edges in $Z$ is equal to the total flow $F$ assigned to $f$.
Let $f$ be a flow for $N$ calculated with the Ford-Fulkerson algorithm using augmenting paths. The evaluation of an augmenting path generates a flow $f$ with $f(e) = 0$ or $f(e) = 1$. Since the edges from $s$ to $V$ and from $W$ to $t$ have capacity 1, all edges with flow 1 have different endpoints. They define a matching $Z$. The number of edges in $Z$ is equal to the total flow $F$ assigned to $f$.
It follows that the number of edges of a maximum matching in $G$ is equal to the maximum total flow $F$ in the network assigned to $G$.

The calculation of a maximum matching in $G$ is reduced to the calculation of a maximum total flow in $N$.

19. a. An augmenting path starts in $s$ and ends in $t$. The number $n-1$ of edges of a path starting from a node of $V_1$ and ending in a node of $V_2$ is odd. Altogether, the augmenting path has $n+1$ edges.
    b. Since the edges are all directed from $V_1$ to $V_2$ and the flow along an edge can only have the value 0 or 1, $e_1 \in Z$, $e_2 \notin Z$, $e_3 \in Z$, ....
    c. Due to point b, the increase of the flow along an augmenting path is 1. Therefore, the number of edges of the matching assigned to the flow increases by 1 in each step.
    d. The number $k$ of edges in a matching is limited by $\min\{|V_1|,|V_2|\} = O(|V|)$. Therefore, the number of iterations of Ford-Fulkerson is of order $O(m)$, where $m$ is equal to the number of edges of $G$.