Informatik 5. Klasse

Examensvorbereitung

Teil 1

Lösungen

Python: Aufgabe 1

```
3.5
3
49
1
2
2
2.0
True
```

Python: Aufgabe 2

```
18
14
6 4 7 1
```

Python: Aufgabe 3

```
True
True
False
12
6
11
```

Python: Aufgabe 4

```
0
3
6
9
2
5
7
2
3
8
5
7
2
3
8
5
```

Python: Aufgabe 5

```
7
1
[7, 4]
[5, 8, 7]
9
5
0
[4, 7, 3, 1, 5]
[2, 6, 3]
```

Python: Aufgabe 6

(a)
```
None
8
9
22
5
[2, 7]
```

(b) In den Zeilen 4–6 sind `x` und `y` formale Parameter während in der Zeile 22 die Zahlen `3` und `4` die aktuellen Parameter der Funktion `g` sind.

Python: Aufgabe 7

```
abcabcxyz
12
21.0
65
0
format
ZB
aaaaa
2024.06.07
3
3 + 4 + 4 = 11
```

Python: Aufgabe 8

```
6
2
1 -> 2
1 -> 3
2 -> 1
2 -> 3
3 -> 1
3 -> 2
```

Python: Aufgabe 9

```
1  L = [3, 5, 4  7, 9, 6]
2
3  for i in range(1, 7)
4      if L[i] % 2 = 0:
5          anzahl =+ 1
6
7   print(anzahl)
```

Syntaxfehler:

- Zeile 1: Komma fehlt zwischen 4 und 7
- Zeile 3: Doppelpunkt fehlt am Zeilenende
- Zeile 4: Vergleich mit `==`
- Zeile 5: Erweiterte Zuweisung ist falsche; `+=` wäre richtig
- Zeile 6: Einrückung falsch

Laufzeitfehler: Die Variable `anzahl` wird nicht initialisiert

Logischer Fehler: Das Programm bestimmt die Anzahl der *geraden* Zahlen statt die Anzahl der ungeraden Zahlen.

Python: Aufgabe 10

```
1  def steuern(E):
2      if E  < 100000:
3          S = E * 0.1
4      elif E < 200000:
5          S = E * 0.2
6      else:
7          S = E * 0.3
8      return S
9
10 print(steuern(10000))
11 print(steuern(150000))
12 print(steuern(200000))
```

Python: Aufgabe 11

```
1  def mittelwert(L):
2      s = sum(L)
3      return s/len(L)
```

Python: Aufgabe 12

```python
def positive(L):
    M = []              # neue leere Liste
    for x in L:         # durchlaufe L
        if x > 0:       # falls x positiv ...
            M.append(x) # ... setze x ans Ende von M
    return M

print(positive([1, -4, 3, 7, -1]))
print(positive([-5, 0, -2]))
print(positive([8, 2, 1, 5]))
```

Python: Aufgabe 13

```python
def find_item(L, item):
    for i in range(0, len(L)): # iteriere über alle Indizes
        if L[i] == item:       # ist L[i] das gesuchte Objekt?
            return i           # falls ja, gib seinen Index zurück
    return False               # offenbar wurde 'item' nicht gefunden
```

Algorithmen: Aufgabe 1

Ein Algorithmus ist eine Handlungsvorschrift zur Lösung einer Klasse von Problemen, die

- endlich (sonst können wir lange auf das Ergebnis warten)
- korrekt (muss das richtige Ergebnis liefern)
- eindeutig (die Abfolge der Schritt ist genau festgelegt)
- allgemein (sollte nicht nur ein sondern eine *Klasse* von Problemen lösen)
- ausführbar (die „Maschine“ muss die Befehle ausführenn können)

ist. Auf ein Kochrezept übertragen:

- endlich (ja)
- korrekt (schwer zu beurteilen; „Wurden diese Crêpes richtig zubereiteit?“)
- eindeutig (nein, „mit Salz und Pfeffer würzen“)
- allgemein (eher nein; meist ist ein Rezept nur für ein Gericht)
- ausführbar (ja, wenn alle Geräte und Zutaten vorhanden sind)

Algorithmen: Aufgabe 2

(a) $T(n) = 2n^2 + 3n - 1 \in O(n^2)$

(b) $T(n) = (n^3 + 4n^2 + n - 2) + (n^2 + 9n + 3) \in O(n^3)$

(c) $T(n) = (n^3 + 4n^2 + n - 2) \cdot (n^2 + 9n + 3) \in O(n^5)$

(d) $T(n) = 7.5 \in O(1)$

(e) $T(n) = 3^{n+2} = 3^n \cdot 3^2 \in O(3^n)$

(f) $T(n) = \log(5n^3) = \log(5) + 3\log(n) \in O(n)$

Algorithmen: Aufgabe 3

(a) Das Programm importiert das `time`-Modul, definiert dann die Funktion `my_function` und misst schliesslich in einer `for`-Schleife die Ausführungszeit der Funktion, für $n = 1 \cdot 10^6$, $n = 2 \cdot 10^6$, …, $n = 9 \cdot 10^6$ und gibt die Problemgrösse (`n`) und die Laufzeiten in Sekunden auf der Shell aus.

(b) *praktisch:* Aus dem Output können wir schliessen, dass der Zeitaufwand ungefähr proportional zur Problemgrösse, d. h. linear wächst. Im Code links können wir erkennen, dass `my_function` die Zahlen von 0 bis $n - 1$ addiert, was die linearen Abhängigkeit der Laufzeit von der Problemgrösse erklärt.

(c) Moderne Computer führen heute mehrere Programme scheinbar gleichzeitig auf einem Prozessor aus. Dabei kann es dazu kommen, dass ein Programm, das plötzlich im Hintergrund gestartet wird, einem anderen Programm Rechenzeit „stiehlt“.

Algorithmen: Aufgabe 4

(a)

Klasse	Beispiel
$O(1)$	einfache arithmetische Operationen, Listenzugriffe
$O(\log(n))$	binäre Suche in sortierten Listen
$O(n\log(n))$	effiziente Sortieralgorithmen (Quicksort)
$O(n^2)$	naive Sortieralgorithmen (Insertion- oder Selectionsort)
$O(2^n)$	Erzeugen aller n-stelligen Binärzahlen
$O(n!)$	Brute Force-Lösung des Travelling Salesman Problems

(b) Die Grenze der praktischen Berechenbarkeit liegt zwischen den polynomiellen Laufzeiten wie $O(n^2)$ oder $O(n^3)$ und den exponentiellen Laufzeiten wie $O(2^n)$ oder $O(3^n)$.

Algorithmen: Aufgabe 5

$T(n) = C \cdot n^2$

$T(200) = C \cdot 200^2 \overset{(1)}{=} 0.1\,\mathrm{s}$

$T(1000) = C \cdot 1000^2 = C \cdot (5 \cdot 200)^2 = 25 \cdot C \cdot 200^2 \overset{(1)}{=} 25 \cdot 0.1 = 2.5\,\mathrm{s}$

Algorithmen: Aufgabe 6

$T(n) = C \cdot \log(n)$

$T(1000) \overset{*}{=} C \cdot \log_{10}(1000) = 3C = 3\,\mathrm{s} \quad \Rightarrow \quad C \overset{(1)}{=} 1\,\mathrm{s}$

$T(10\,000) = C \cdot \log_{10}(10\,000) = 4C \overset{(1)}{=} 4 \cdot 1\,\mathrm{s} = 4\,\mathrm{s}$

Algorithmen: Aufgabe 7

$T(n) = C \cdot 3^n$

$T(5) = C \cdot 3^5 \overset{(1)}{=} 24\,\mathrm{Min.}$

$T(4) = C \cdot 3^4 = C \cdot 3^{5-1} = C \cdot 3^5 : 3^1 \overset{(1)}{=} 24 : 3 = 8\,\mathrm{Min.}$

Algorithmen: Aufgabe 8

$T(n) = C \cdot n!$

$T(8) = C \cdot 8! \overset{(1)}{=} 1\,\mathrm{s}$

$T(10) = C \cdot 10! = C \cdot 8! \cdot 9 \cdot 10 \overset{(1)}{=} 1\,\mathrm{s} \cdot 90 = 1.5\,\mathrm{Min.}$

Algorithmen: Aufgabe 9

$T(n) = C \cdot 1$

$T(2000) = C \cdot 1 \overset{(1)}{=} 20\,\text{ms}$

$T(3000) = C \cdot 1 \overset{(1)}{=} 20\,\text{ms}$

Algorithmen: Aufgabe 10

(a) Die Funktion `funktion_a(n)` liegt in $O(n)$.

(b) Die Funktion `funktion_b(n)` liegt in $O(n^2)$.

(c) Die Funktion `funktion_c(n)` liegt in $O(1)$.

(d) Die Funktion `funktion_d(n)` liegt in $O(n)$.

Achtung: die Schleifen werden *nacheinander* und *nicht verschachtelt* ausgeführt. Und auch wenn die zweite Schleife $2n$-Mal ausgeführt wird, ist der Aufwand immer noch linear.

(e) Die Funktion `funktion_e(n)` liegt in $O(\log(n))$.

- Für $n = 2$ wird der Schleifenkörper von `while` für $i = 1$ ausgeführt.
- Für $n = 4$ wird der Schleifenkörper von `while` für $i = 1, 2$ ausgeführt.
- Für $n = 8$ wird der Schleifenkörper von `while` für $i = 1, 2, 3$ ausgeführt.
- ...

Der Zusammenhang zwischen n und der Anzahl der Schleifendurchläufe ist logarithmisch.

Algorithmen: Aufgabe 11

(a) $\text{ggT}(36, 20) = \text{ggT}(20, 16) = \text{ggT}(16, 4) = \text{ggT}(4, 12) \overset{(s)}{=} \text{ggT}(12, 4)$
$= \text{ggT}(4, 8) \overset{(s)}{=} \text{ggT}(8, 4) = \text{ggT}(4, 4) = \text{ggT}(4, 0) = 4$

(s) = Swap

(b) $\text{ggT}(20, 36) = \text{ggT}(36, 20) = \text{ggT}(20, 16) = \text{ggT}(16, 4) = \text{ggT}(4, 0) = 4$

(c) Mit dem Divisionsrest werden die vielen Subtraktionen vermieden, die bei der Berechnung des ggTs von zwei Zahlen unterschiedlicher Grössenordnungen (z. B. $a = 100$ und $b = 3$) nötig sind.

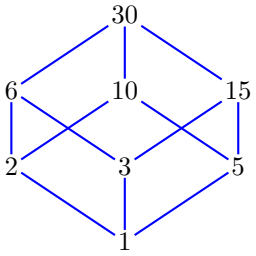
(d) Für zwei benachbarte Zahlen der Fibonacci-Folge $(1, 1, 2, 3, 5, 8, 13, \dots)$

Algorithmen: Aufgabe 12

```
1 def gcd(a, b):
2 
3     while b != 0:
4         a, b = b, a % b
5 
6     return a
7 
8 print(gcd(72,40))
```

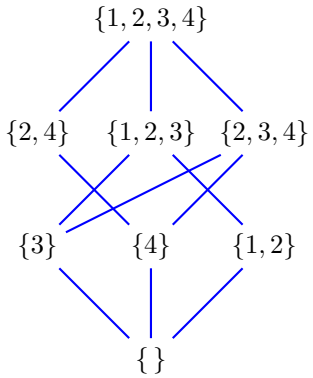
Halbordnungen: Aufgabe 2

$T_{30} = \{1, 2, 3, 5, 6, 10, 15, 30\}$



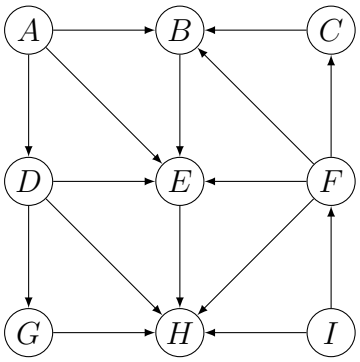
Solange alle direkten Nachfolger oberhalb ihrer Vorgänger liegen, braucht es keine Pfeile.

Halbordnungen: Aufgabe 3



Solange alle direkten Nachfolger oberhalb ihrer Vorgänger liegen, braucht es keine Pfeile.

Halbordnungen: Aufgabe 4



Bemerkung: Aus Platzgründen werden in dieser Lösung zuerst alle Knoten ohne eingehenden Kanten bestimmt und erst dann deren Kanten entfernt. Dies kann manchmal zu einer anderen Reihenfolge führen, als wenn die Kanten unmittelbar nach dem Auffinden eines Knotens ohne eingehende Kanten gelöscht werden und so gewisse Knoten schon früher von eingehenden Kanten „befreit“ werden. Die topologische Sortierung ist aber trotzdem noch korrekt. Wenn der Graph nicht zu gross ist, kann man die eigene Lösung prüfen, indem man die gerichteten Kanten des Graphen auf die eigene Sortierung überträgt. Geht kein Pfeil rückwärts (also von rechts nach links), stimmt die Lösung.

A, I haben keine eingehenden Kanten. $\Rightarrow$ $S = [A, I]$

Streiche A, I und ihre ausgehenden Kanten.

D, F haben keine eingehenden Kanten $\Rightarrow$ $S = [A, I, D, F]$

Streiche D und F und alle ihre ausgehenden Kanten.

C, G haben keine eingehenden Kanten $\Rightarrow$ $S = [A, I, D, F, C, G]$

Streiche C und G und alle ihre ausgehenden Kanten.

B hat keine eingehenden Kanten $\Rightarrow$ $S = [A, I, D, F, C, G, B]$

Streiche B und alle seine ausgehenden Kanten.

E hat keine eingehenden Kanten $\Rightarrow$ $S = [A, I, D, F, C, G, B, E]$

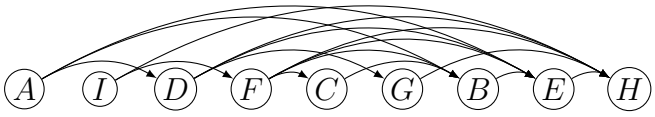
Streiche E und alle seine ausgehenden Kanten.

H hat keine eingehenden Kanten $\Rightarrow$ $S = [A, I, D, F, C, G, B, E, H]$

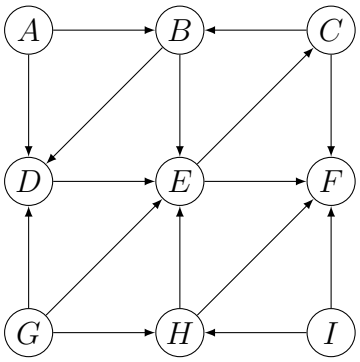
Streiche H und alle seine ausgehenden Kanten.

Alle Kanten wurden entfernt $\Rightarrow$ Ende

Kontrolle:



Halbordnungen: Aufgabe 5



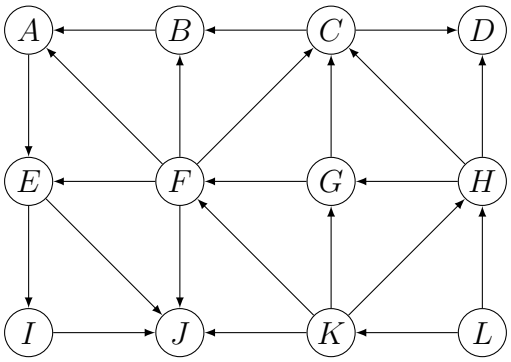
A, G, I haben keine eingehende Kanten $\Rightarrow S = [A, G, I]$

Streiche alle ausgehenden Kanten von A, G, I

H hat keine eingehende Kanten $\Rightarrow S = [A, G, I, H]$

Der Graph hat keine Knoten ohne eingehende Kanten und ist nicht leer. Somit ist der Graph zyklisch ($BDECB$, $BECB$) und kann nicht topologisch sortiert werden.

Halbordnungen: Aufgabe 6



L hat keine eingehende Kanten $\Rightarrow S = [L]$

Streiche alle ausgehenden Kanten von L.

K hat keine eingehende Kanten $\Rightarrow S = [L, K]$

Streiche alle ausgehenden Kanten von K.

H hat keine eingehende Kanten $\Rightarrow S = [L, K, H]$

Streiche alle ausgehenden Kanten von H.

G hat keine eingehende Kanten $\Rightarrow S = [L, K, H, G]$

Streiche alle ausgehenden Kanten von G.

F hat keine eingehende Kanten $\Rightarrow S = [L, K, H, G, F]$

Streiche alle ausgehenden Kanten von F.

C hat keine eingehende Kanten $\Rightarrow S = [L, K, H, G, F, C]$

Streiche alle ausgehenden Kanten von C.

B, D haben keine eingehende Kanten $\Rightarrow S = [L, K, H, G, F, C, B, D]$

Streiche alle ausgehenden Kanten von B, D.

A hat keine eingehende Kanten $\Rightarrow S = [L, K, H, G, F, C, B, D, A]$

Streiche alle ausgehenden Kanten von A.

E hat keine eingehende Kanten $\Rightarrow S = [L, K, H, G, F, C, B, D, A, E]$

Streiche alle ausgehenden Kanten von E.

I hat keine eingehende Kanten $\Rightarrow S = [L, K, H, G, F, C, B, D, A, E, I]$

Streiche alle ausgehenden Kanten von I.

J hat keine eingehende Kanten $\Rightarrow S = [L, K, H, G, F, C, B, D, A, E, I, J]$

Streiche alle ausgehenden Kanten von J.

Alle Kanten wurden entfernt $\Rightarrow$ Ende

Kontrolle: