Karlsruher Institut für Technologie

Institut für Visualisierung und Datenanalyse
Lehrstuhl für Computergrafik
Prof. Dr.-Ing. Carsten Dachsbacher

Hauptklausur Algorithmen I
SS 2021

02. September 2021

Name:	
Matrikelnummer:	

Beachten Sie:

- Schreiben Sie Ihren vollständigen **Namen** und **Matrikelnummer** in Druckschrift in das Feld auf dem Deckblatt!
- Schreiben Sie Ihre Matrikelnummer oben auf jedes bearbeitete Aufgabenblatt.
- Schreiben Sie Ihre Lösungen auf die Aufgabenblätter. Bei Bedarf können Sie weiteres Papier anfordern.
- Halten Sie sich bei Pseudocode-Aufgaben an die Richtlinien auf der nächsten Seite.
- Wir akzeptieren auch englische Antworten.
- Sie dürfen **ein handbeschriebenes Din A4-Blatt** mitbringen, sonst sind **keine weiteren Hilfsmittel** zugelassen.
- Sie haben **120 Minuten** Bearbeitungszeit.
- Die Klausur umfasst 28 Seiten (14 Blätter) mit 10 Aufgaben.

Aufgabe	1	2	3	4	5	6	7	8	9	10	**Gesamt**
Erreichte Punkte											
Erreichbare Punkte	10	31	16	24	23	45	30	14	16	31	**240**

Note

Pseudocode-Richtlinien

Verwenden Sie in der Klausur folgende Konventionen für alle Aufgaben, bei denen Sie Pseudocode verwenden! In den Fällen, die hiervon nicht abgedeckt sind, halten Sie sich an die Notation aus der Vorlesung / Übung.

`for`-Schleifen: Indizes iterieren

Verwenden Sie eine der folgenden Schreibweisen für `for`-Schleifenköpfe! In allen Beispielen nimmt `i` alle ganzzahligen Werte von 0 bis inklusive 99 an, aber nicht 100:

```
for i ← 0; i < 100; i++
for i in [0, 99]                for i ∈ [0, 99]
for i in [0,...,99]             for i ∈ [0,...,99]
for i from 0 to 99              for i ← 0 to 99
```

`foreach`-Schleifen: Datenstrukturen iterieren

Über Elemente einer Datenstruktur können Sie in undefinierter Reihenfolge wie folgt iterieren:

```
foreach e in E
foreach (u,v) in E
foreach e = (u,v) in E
```

Array-Indizierung

Arrays und Felder werden mit 0 indiziert. Das heißt, für ein Array `a: [int; 5]` mit 5 Einträgen wird mit `a[0]` auf das erste Element von `a` zugegriffen und mit `a[4]` auf das Letzte.

2D-Arrays

Ein $M \times N$-Array `a` (z.B. zur Repräsentation einer $M \times N$-Matrix $A = (a_{ij})$) mit M Zeilen und N Spalten und Einträgen vom Typ `typ` wird folgendermaßen deklariert:

```
a: [[typ; N]; M]
```

Das heißt, jede Zeile ist ein Feld `[typ; N]`.
Die Zeilen werden nacheinander in `a` abgelegt.

Auf den Eintrag in der i-ten Zeile und j-ten Spalte (bzw. den Matrixeintrag a_{ij}) wird zugegriffen mit

```
a[i][j]
```

Matrikelnummer: _____

Aufgabe 1: Laufzeit (10 Punkte)

a) Zeigen Sie, dass $\mathcal{O}(n^2) \subseteq \mathcal{O}(n^3)$! **(6 Punkte)**

b) Zeigen Sie, dass $\sum_{i=1}^{n} i \in \mathcal{O}(n^2)$! **(4 Punkte)**

Aufgabe 2: Algorithmenanalyse (31 Punkte)

In dieser Aufgabe sollen Sie die Laufzeit eines Algorithmus analysieren. Der Algorithmus überprüft, ob eine gegebene Texteingabe `text` $\in [a, \ldots, z]^*$ zu der gegebenen Schablone `pattern` $\in [a, \ldots, z, *]^*$ passt. In der Schablone stellt der Wildcard-Buchstabe '$*$' beliebig viele oder keinen Textbuchstaben in `text` dar. Die Tabelle zeigt einige Beispiele:

Pattern / Text	ab	aab	axb	aaxazbbb	ac
ab	✓	·	·	·	·
a∗b	✓	✓	✓	✓	·
aa∗z∗b	·	·	·	✓	·

```
Function match(text: string, textpos: int, pattern: string,
patternpos: int) : bool
{
  endOfPattern : bool ← patternpos == length(pattern)
  endOfText : bool ← textpos == length(text)

  if endOfPattern then
|    return endOfText

   else if pattern[patternpos] == '*' then
|     if not endOfText and
|     match(text, textpos+1, pattern, patternpos) then
|  |    return True
|     return match(text, textpos, pattern, patternpos+1)
   else if not endOfText and
        text[textpos] == pattern[patternpos]) then
|    return match(text, textpos+1, pattern, patternpos+1)

   return False
}
```

Beispielaufruf: `match("aab", 0, "a*b", 0)`

a) Stellen Sie eine Rekurrenzformel für die asymptotische Laufzeit von `match` auf! Die Eingabegröße für die Rekurrenz sei $n = N + M$, wobei noch N Buchstaben im Eingabetext und M Buchstaben im Eingabe-Pattern verbleiben:
$N =$ `length(text)-textpos`,
$M =$ `length(pattern)-patternpos`.
Es genügt, wenn Sie sich bei Zweigen in Abhängigkeit der Eingabebuchstaben auf die maximal mögliche Laufzeit beschränken. **(6 Punkte)**

Hinweis: Beachten Sie, dass der Algorithmus bzgl. n in jeder Rekursionsebene Fortschritte macht.

Matrikelnummer: _____

b) Ist das Master-Theorem anwendbar? Begründen Sie! **(4 Punkte)**

c) Bestimmen Sie eine möglichst enge obere Schranke in $\mathcal{O}$-Notation für die Rekurrenz aus a) und begründen Sie Ihre Antwort mit dem Master-Theorem oder vollständiger Induktion, je nach Anwendbarkeit! **(7 Punkte)**

d) Nehmen Sie an, `pattern` enthält mindestens ein '$*$' und `match` verarbeitet zum ersten Mal den letzten '$*$'-Buchstaben in `pattern`.

Dann setzt sich die Rekurrenz T_R für die Restlaufzeit aus zwei Komponenten T_T und T_P zusammen:

$T_R(N + M) = T_T(N, M) + T_P(N, M)$.

- $T_T(N, M)$ ist die Restlaufzeit zur Verarbeitung des nächsten Text-Buchstabens
- $T_P(N, M)$ ist die Restlaufzeit zur Verarbeitung des nächsten Pattern-Buchstabens
- N ist die Anzahl verbleibender Text-Buchstaben
- M ist die Anzahl verbleibender Pattern-Buchstaben

☐ Geben Sie die Rekurrenzen T_T und T_P an! **(10 Punkte)**

☐ e) Geben Sie für $T_T(N, M)$ und $T_P(N, M)$ möglichst enge obere Schranken in $\mathcal{O}$-Notation abhängig von $n = N + M$ an! **(4 Punkte)**

Matrikelnummer: _____

Aufgabe 3: Hashing (16 Punkte)

a) Wann ist $\mathcal{H} \subseteq \{0 \dots m-1\}^{key}$ eine Familie universeller Hashfunktionen? **(4 Punkte)**

b) Sie erhalten einen Datensatz `D` bestehend aus n Paaren (i, d), die jeweils eine durchgeführte Corona-Impfung dokumentieren. Dabei ist $i \in \mathbb{N}$ eine eindeutige ID der Person, welche eine Impfung bekommen hat, und $d \in \{1, \dots, 366\}$ der Tag der Impfung. Beachten Sie, dass pro Person bis zu zwei Impfungen an unterschiedlichen Tagen durchgeführt werden. Für jede Impfung gibt es genau ein Paar in `D`.

- Geben Sie einen Algorithmus in Pseudocode an, der in erwarteter Zeit $\mathcal{O}(n)$ und mit Speicherverbrauch $\mathcal{O}(n)$ die IDs aller Personen mit `print(id)` ausgibt, die bereits zwei Mal geimpft wurden!
- Begründen Sie kurz, warum Ihr Algorithmus das gewünschte Laufzeitverhalten aufweist!

(12 Punkte)

```
Function printVaccinated(D: [(int, int); n]) {

}
```

Zusätzlicher Platz:

Matrikelnummer: _____

Aufgabe 4: Sortieren (24 Punkte)

a) Geben Sie für die gegebenen Sortieralgorithmen in $\mathcal{O}$-Notation abhängig von der Anzahl n zu sortierender Elemente an:

i) die asymptotische Worst Case-Laufzeit! **(3 Punkte)**

2-Wege-Quicksort	Mergesort	Insertionsort

ii) den zusätzlichen Speicherbedarf (inklusive Rekursionsstack) im Worst Case! **(3 Punkte)**

2-Wege-Quicksort	Mergesort	Insertionsort

b) Gegeben sei der folgende Algorithmus *stableSort*, der Arrays von Elementen vom Typ `T` stabil sortieren soll. Intern verwendet *stableSort* dabei einen nicht stabilen Sortieralgorithmus *unstableSort*. In dieser Aufgabe sollen Sie die Eingabe für *unstableSort* in einen neuen Datentyp mit neuer Ordnungsrelation transformieren, sodass das Ergebnis von *unstableSort* in eine stabile Sortierung der ursprünglichen Eingabe zurücktransformiert werden kann.

```
Function stableSort(array: [T; n]) {
    transformed: [U; n] ← transform(array)
    unstableSort(transformed)
    array ← reverseTransform(transformed)
}
```

i) Definieren Sie den Typ `U` der Elemente des transformierten Arrays! **(4 Punkte)**

`U` :=

ii) Definieren Sie eine passende Ordnungsrelation $<_{\mathtt{U}}$ auf `U`, welche für die anschließende Sortierung verwendet wird! Sie können dabei die auf `T` definierte Ordnungsrelation $<_{\mathtt{T}}$ verwenden. **(6 Punkte)**

_____ $<_{\mathtt{U}}$ _____ :$\Leftrightarrow$

iii) Führen Sie die Transformation der Eingabe in der Funktion *transform* so in Pseudocode durch, dass *stableSort* dann eine stabile Sortierung der Eingabe durchführt! Die Transformation muss dabei reversibel und die Laufzeit in $\mathcal{O}(n)$ sein. **(8 Punkte)**

```
Function transform(array: [T; n]) : [U; n] {

}
```

Matrikelnummer: _____

Aufgabe 5: Prioritätslisten (23 Punkte)

a) Sei v ein Knoten in einem binären Min-Heap mit dem linken Kind l und dem rechten Kind r. Was gilt aufgrund der Heap-Eigenschaft für diese Knoten? **(2 Punkte)**

b) Führen Sie die Binär-Heap-Konstruktion `buildHeapBackwards` aus der Vorlesung auf dem gegebenen 1-indizierten Array aus! Geben Sie dazu nach jedem Aufruf von `siftDown(i)` aus `buildHeapBackwards`, welcher das Array ändert, den neuen Zustand des Arrays sowie den Wert des Parameters `i` an! **(9 Punkte)**

Index	**Array**								
i	50	30	20	70	90	10	80	60	40

c) Die Operation `delete` soll das Element `h[i]` so aus einem binären Min-Heap `heap` mit n Elementen vom Typ `float` löschen, dass `heap` nach Ausführung wieder ein gültiger Min-Heap ist! Implementieren Sie die Funktion `delete` in Pseudocode mit Laufzeit $\mathcal{O}(\log n)$! Sie dürfen die angegebenen Funktionen nutzen. **(12 Punkte)**

```
Struct Heap {
  h: [float; N_MAX],
  n: int,
}

Function parent(i: int) : int
Function leftChild(i: int) : int
Function rightChild(i: int) : int

Function insert(heap: Heap, e: float)
Function deleteMin(heap: Heap) : float
Function siftDown(heap: Heap, i: int)
Function siftUp(heap: Heap, i: int)


Function delete(heap: Heap, i: int) {
   assert 0 ≤ i <heap.n
```

```
}
```

Matrikelnummer: _____

Aufgabe 6: Graphen (45 Punkte)

a) Wie können Sie anhand einer Adjazenzmatrix erkennen, ob der zugehörige Graph schleifenfrei ist? **(2 Punkte)**

b) Bei der Breitensuche oder Tiefensuche in einem Graphen ergibt sich ein Suchbaum. Die Kanten des Graphen können bezüglich eines solchen Suchbaums in 4 Klassen klassifiziert werden: Baumkanten, Cross-Kanten, und zwei Weitere. Nennen und beschreiben Sie die beiden weiteren Klassen! **(6 Punkte)**

1.

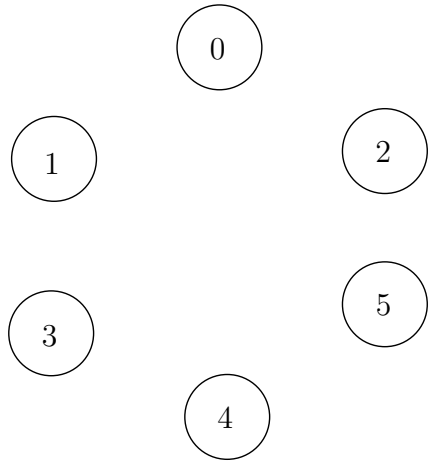
2.

c) Wie können Sie mithilfe der Tiefensuche bestimmen, ob ein gerichteter Graph kreisfrei ist? **(2 Punkte)**

d) Die folgende Adjazenzmatrix M beschreibt einen gerichteten Graphen $G = (V, E)$ mit $V = \{0, 1, 2, 3, 4, 5\}$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

i) Zeichnen Sie den Graphen! **(4 Punkte)**



ii) Stellen Sie den Graphen durch die Adjazenzfeld-Repräsentation dar, indem Sie die angegebene Tabelle ausfüllen! **(8 Punkte)**

V:												
E:												

Zusätzlicher Platz zur Fehlerkorrektur. Kennzeichnen Sie eindeutig, welche Lösung gewertet werden soll!

V:												
E:												

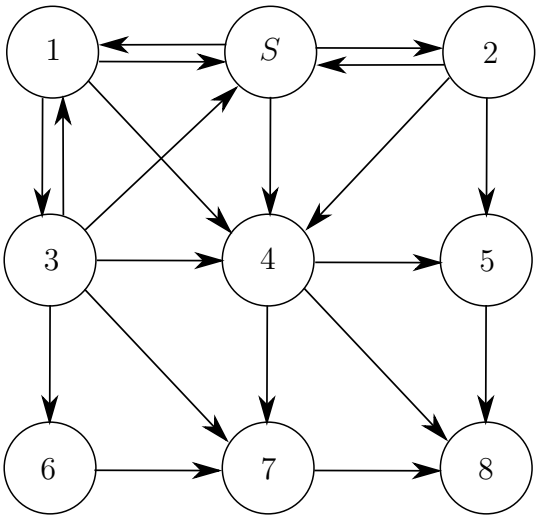
Matrikelnummer: _____

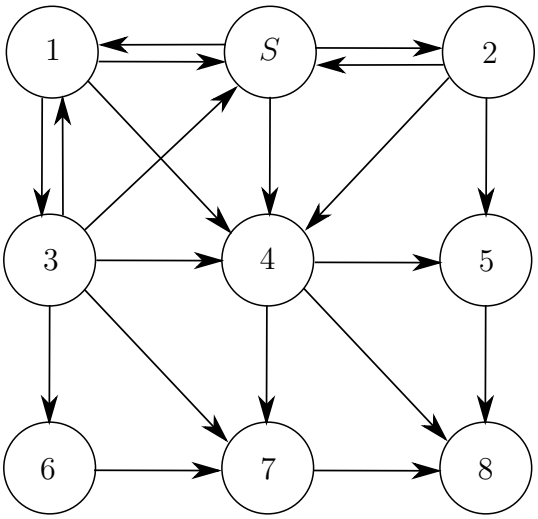
e) Führen Sie auf dem folgenden Graphen eine Tiefensuche vom Startknoten S aus durch! Falls mehrere Knoten als Nächstes traversiert werden können, wählen Sie den Knoten mit geringstem Index!

- Markieren Sie alle Baumkanten der Tiefensuche mit b und alle Cross-Kanten mit x!
- Welche Kante wird den Baumkanten als Letztes hinzugefügt?

Falls Sie den zweiten Graphen zur Fehlerkorrektur nutzen, kennzeichnen Sie eindeutig, welche Lösung gewertet werden soll! **(8 Punkte)**

Letzte Baumkante:





f) Gegeben sei ein gerichteter Graph $G = (V, E)$ mit $V = \{0, ..., N-1\}$ und $|E| = k$. Die Funktion `matrixToField` soll als Eingabe die Adjazenzmatrix `M` von G sowie ein leeres Adjazenzfeld bestehend aus `V` und `E` erhalten, und das Adjazenzfeld möglichst laufzeiteffizient mit der Repräsentation für den Graphen G ausfüllen.

- Implementieren Sie die Funktion in Pseudocode!
- Bestimmen und begründen Sie möglichst enge Laufzeitschranken für Ihren Algorithmus in $\mathcal{O}$-Notation!

Hinweis: Mit `M[i][j]` können Sie auf den Eintrag m_{ij} der Matrix M zugreifen.
(15 Punkte)

```
Function matrixToField(M: [[int; N]; N],
                       V: [int; N+1],
                       E: [int; k])
{
      V[0] ← 0




}
```

Matrikelnummer: _____

Aufgabe 7: Kürzeste Wege (30 Punkte)

a) Gegeben sei die folgende Implementierung von Dijkstras Algorithmus für gerichtete Graphen $G = (V, E)$ mit Knoten $V = \{0, \ldots, N-1\}$, Startknoten `s` und Kantengewichtungsfunktion `w(e: Edge):float`.

```
Function Dijkstra(s: int) : ([float; |V|], [int; |V|])
parent ← [int; |V|]
parent[s] ← s
d← [float; |V|]
d ← {∞,...,∞}
d[s] ← 0
Q ← NodePQ
Q.insert(s, d[s])
while Q ≠ ∅ do
    u ← Q.deleteMin()
    for Edge e = (u,v) in E do
        if d[u] + w(e) < d[v] then
            d[v] ← d[u] + w(e)
            parent[v] ← u
            if v in Q then
                Q.decreaseKey(v, d[v])
            else
                Q.insert(v, d[v])
return (d, parent)
```

Wir betrachten die Operationen `deleteMin`, `decreaseKey` und `insert`.

i) Geben Sie in $\mathcal{O}$-Notation möglichst enge Schranken an für die Anzahl der Aufrufe der Operationen! Begründen Sie kurz Ihre Antworten! **(7 Punkte)**

deleteMin	decreaseKey	insert

Begründung:

ii) Geben Sie in $\mathcal{O}$-Notation möglichst enge Schranken an für die Worst-Case-Laufzeit je einer Ausführung der Operation, falls `Q` als addressierbare Binärheap-Prioritätsliste implementiert ist! *(ohne Begründung)* **(3 Punkte)**

`deleteMin`	`decreaseKey`	`insert`

iii) Geben Sie in $\mathcal{O}$-Notation möglichst enge Schranken an für die Worst-Case-Laufzeit je einer Ausführung der Operation, falls `Q` basierend auf einer doppelt verketteten Liste implementiert ist! Begründen Sie Ihre Antworten! **(9 Punkte)**

`deleteMin`	`decreaseKey`	`insert`

Begründung:

Matrikelnummer: _____

b) Zeichnen Sie einen gerichteten, gewichteten Graphen $G = (V, E)$ mit $V = \{A, B, C\}$, der keinen eindeutigen kürzesten Weg von Knoten A zu Knoten C hat! **(3 Punkte)**

c) Betrachten Sie folgende Distanzmatrix eines gerichteten Graphen $G = (V, E)$ mit den Knoten $v_0, v_1, ..v_4 \in V$. Ein Eintrag a_{ij} mit $i, j \in [0; 4]$ in der Matrix gibt das Kantengewicht der Kante (v_i, v_j) an. Ein Wert von ∞ bedeutet, dass keine Kante von v_i nach v_j existiert:

$$A = \begin{pmatrix} 5 & \infty & \infty & 8 & \frac{1}{3} \\ 1 & \infty & 10 & \infty & 3 \\ \infty & 11 & \infty & \frac{2}{3} & 1 \\ \frac{8}{7} & \infty & \infty & -1 & 5 \\ \infty & \infty & \infty & -2 & \infty \end{pmatrix}$$

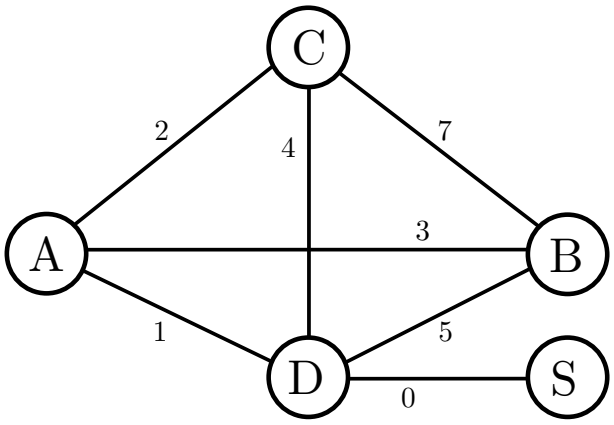
i) Welcher Algorithmus aus der Vorlesung ist geeignet, effizient kürzeste Wege zwischen beliebigen Knoten in diesem Graphen zu finden? Begründen Sie ihre Antwort! **(2 Punkte)**

ii) Gibt es einen kürzesten Weg von Knoten v_0 zu Knoten v_2? Begründen Sie Ihre Antwort! **(6 Punkte)**

Matrikelnummer: _____

Aufgabe 8: Minimale Spannbäume (MST) (14 Punkte)

a) Bestimmen Sie den minimalen Spannbaum des folgenden Graphen mit Hilfe des Jarník-Prim-Algorithmus! Setzen Sie die Tabelle fort, indem Sie pro Schritt die zum MST hinzugefügte Kante sowie den aktuellen Inhalt der Prioritätsliste in der Form (Knoten, Gewicht) angeben! **(6 Punkte)**



	Hinzugefügte Kante	PQ $\{(X, 42), \dots\}$
1. Schritt	{S,D}	{(A,1), (C,4), (B,5)}
2. Schritt	{ , }	
3. Schritt	{ , }	
4. Schritt	{ , }	

b) Erklären Sie die folgenden zwei Eigenschaften minimaler Spannbäume (MST)! **(4 Punkte)**

i) Schnitt-Eigenschaft:

ii) Kreis-Eigenschaft:

c) Führen Sie auf der im Folgenden tabellarisch dargestellten Union-Find-Datenstruktur (ohne Pfadkompression und ohne Union by rank) die angegebenen Operationen aus! **(4 Punkte)**

Node	1	2	3	4	5	6
Parent	1	1	4	4	4	5

i) find(6) =

ii) find(2) =

iii) union(2, 6)

Matrikelnummer: _____

Node	1	2	3	4	5	6
Parent						

Aufgabe 9: Suchbäume (16 Punkte)

a) Geben Sie die minimale und maximale Höhe eines binären Suchbaums mit n Elementen in Θ-Notation an! **(4 Punkte)**

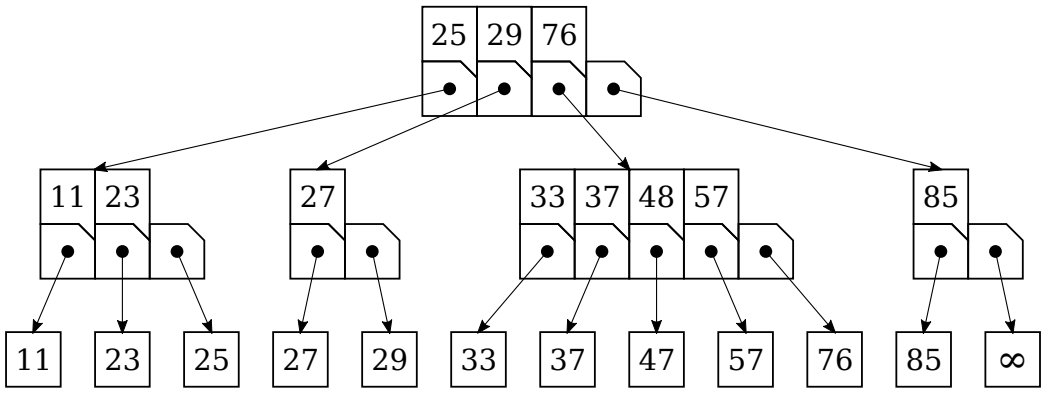
minimale Höhe $h_{\min} \in$

maximale Höhe $h_{\max} \in$

b) Ist die Konstruktion eines binären Suchbaums im Allgemeinen aus einer Sequenz von n Elementen in Laufzeit $\mathcal{O}(n)$ möglich? Begründen Sie! **(6 Punkte)**

Matrikelnummer: _____

c) Führen Sie auf dem folgenden $(2, 5)$-Baum die Operation `insert(61)` aus und geben Sie den resultierenden Baum an! **(6 Punkte)**



Aufgabe 10: Greedy-Münzwechselproblem (31 Punkte)

Beim Münzwechselproblem muss ein gegebener Betrag aus einer minimalen Anzahl Münzen verschiedener Wertigkeit zusammengestellt werden.

Gegeben seien n Münzwerte $c_0, \ldots, c_{n-1}$, wobei $\forall i \in \{0, \ldots, n-2\}$ gilt:

- $c_0 = 1$
- $c_i < c_{i+1}$
- $\frac{c_{i+1}}{c_i} \in \mathbb{N}_+$

Ein Vektor $\boldsymbol{v} \in \mathbb{N}^n$ beschreibt die Anzahlen v_i der gewählten Münzen und der Vektor $\boldsymbol{c} \in \mathbb{N}^n$ beschreibt die Wertigkeit der Münzen c_i. Er ist eine Lösung des Münzwechselproblems für den Betrag $A \in \mathbb{N}_+$, wenn gilt:

$$\boldsymbol{v} \cdot \boldsymbol{c} = \sum_{i=0}^{n-1} v_i \, c_i = A.$$

Eine Lösung $\boldsymbol{v}$ sei optimal, falls $\sum_{i=0}^{n-1} v_i$ minimal ist.

a) Geben Sie für den Betrag $A = 345$ eine optimale Lösung $\boldsymbol{v}$ an, wenn folgende Münzwerte zur Verfügung stehen: **(5 Punkte)**

c_0	c_1	c_2	c_3	c_4	c_5
1	5	10	50	100	200

$\boldsymbol{v} =$

b) Zeigen Sie, dass für eine optimale Lösung $\boldsymbol{v}$ folgende Aussage gilt: **(10 Punkte)**

$$\forall i \in \{0, \ldots, n-2\} : v_i \leq \frac{c_{i+1}}{c_i} - 1$$

Hinweis: Sie können den Beweis zum Beispiel als Widerspruchsbeweis führen.

c) Diese Aufgabe soll zeigen, dass ein Greedy-Algorithmus, der stets die größtmögliche Münze wählt, immer die optimale Lösung findet. Dafür sollen Sie die folgenden Eigenschaften zeigen.

i) Zeigen Sie: Für jeden Betrag $A \in \mathbb{N}_+$ gibt es eine Lösung. **(2 Punkte)**

ii) Sei k maximal sodass $A \geq c_k$. Sei $A' = A - c_k$ und $\boldsymbol{v}'$ eine optimale Lösung für A'.

Sei $\boldsymbol{v}$ mit $v_k = v'_k + 1$ und $v_{j \neq k} = v'_j$.

Zeigen Sie: $\boldsymbol{v}$ ist eine optimale Lösung für A. **(14 Punkte)**

Hinweise:

- Sie können den folgenden Satz verwenden: Sei $\boldsymbol{v}$ eine optimale Lösung für A und sei k maximal sodass $A \geq c_k$. Dann gilt $v_k > 0$.
- Zeigen Sie erst, dass $\boldsymbol{v}$ eine Lösung für A ist, und dann, dass es eine optimale Lösung für A ist, z.B. durch Widerspruch.