

ALGORITHMEN

HS 2024

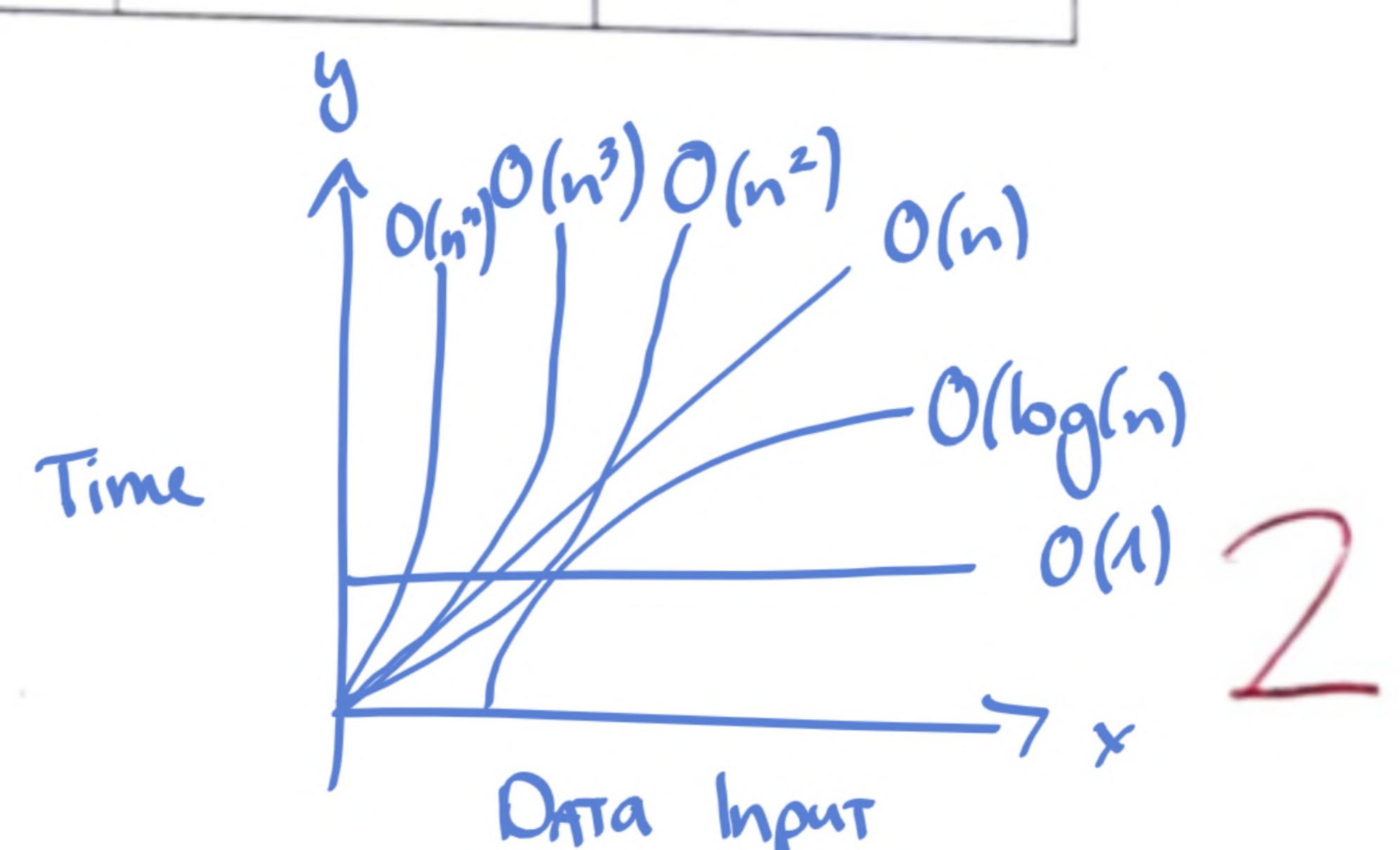
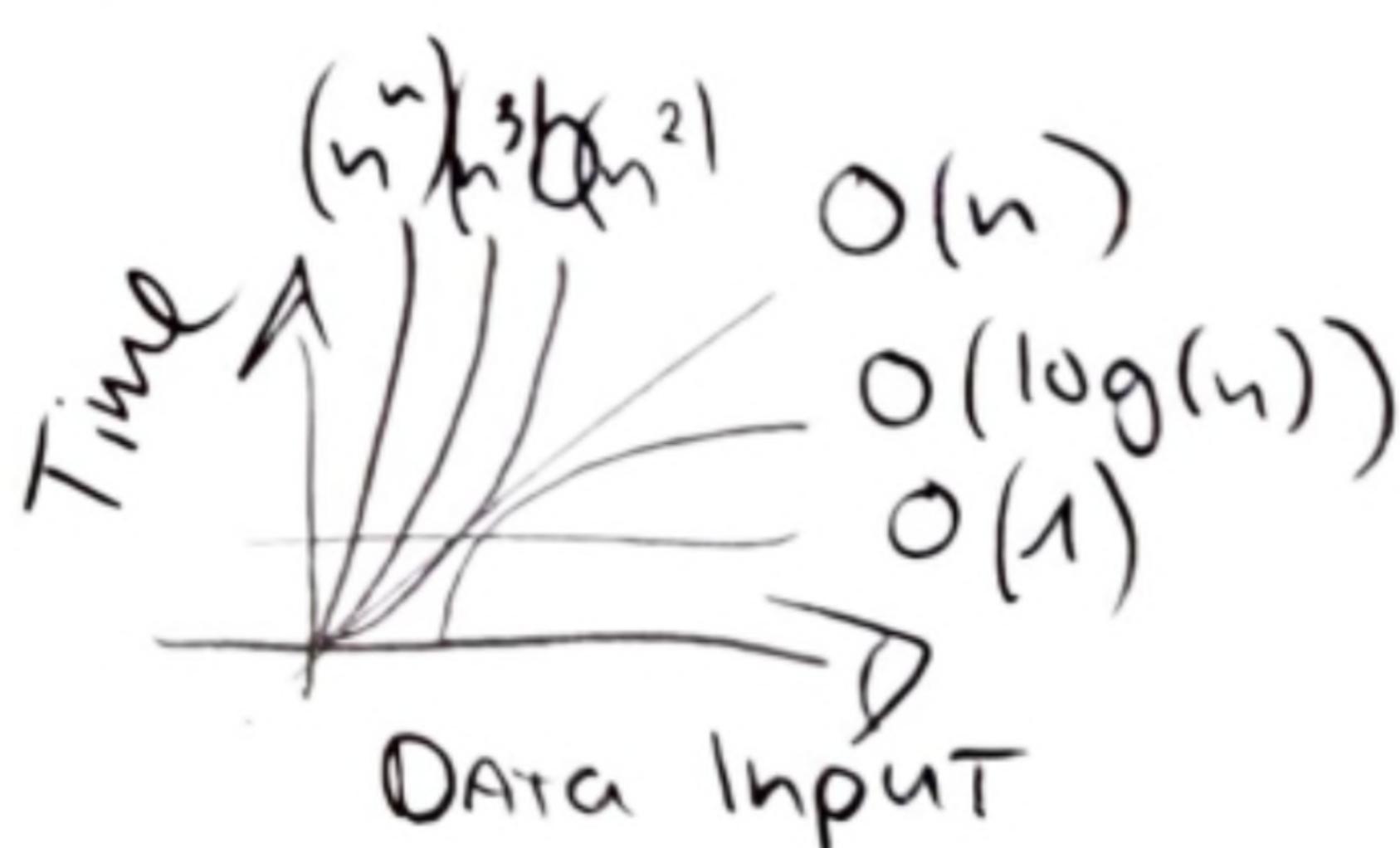
Modulendprüfung AD im HS 2024 – Teil 3: Algorithmen

Aufgabe 1: Algorithmus (3 Punkte)

Sind untenstehende Aussagen richtig? Kreuzen Sie entsprechend an. (3 Punkte)

Hinweis: Ein richtiges Kreuz ergibt 0.5 Punkte, ein falsches -0.5 Punkte. Sie erhalten bei dieser Aufgabe insgesamt aber *keine* negativen Punkte.

	richtig	falsch
Ein Algorithmus endet garantiert nach endlich vielen Schritten.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Ein Algorithmus löst ein Problem bzw. eine Problemklasse.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Algorithmen mit einer exponentiellen Laufzeitkomplexität sind in der Praxis kaum von Bedeutung.	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Mit Hilfe der Ordnung eines Algorithmus kann man seine Laufzeit bzw. seinen Speicherbedarf exakt berechnen.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Jede Methode in Java, welche einen Rückgabewert liefert, realisiert einen Algorithmus.	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Algorithmen und Datenstrukturen hängen eng zusammen und sind demnach gleichwertig.	<input checked="" type="checkbox"/>	<input type="checkbox"/>



Stabil = Wenn bei gleichen Werten die ursprüngliche Reihenfolge beibehalten wird.

Aufgabe 2: Sortieren (8 Punkte)

- a) Sortieren Sie die nachfolgenden Zahlenwerte *aufsteigend*. Ihre Sortierung soll das Verhalten eines Algorithmus illustrieren, welcher **stabil** ist. Anmerkung: Schreiben Sie bei den Zahlen auch den kleinen Index hin; dieser steht für die Ausgangsposition 1...10 vor dem Sortieren. (2 Punkte)

9 ₁	2 ₂	10 ₃	17 ₄	15 ₅	9 ₆	99 ₇	14 ₈	9 ₉	17 ₁₀
2 ₂	9 ₁	9 ₆	9 ₉	10 ₃	14 ₈	15 ₅	17 ₄	9 ₁₀	99 ₇

2

- b) Geben Sie einen Sortieralgorithmus an, welcher *garantiert* eine Laufzeitkomplexität von $O(n \cdot \log n)$ besitzt. (1 Punkt)

Mergesort, hat garantiert immer $O(n \cdot \log(n))$

0

- c) Geben Sie einen *stabilen* Sortieralgorithmus an, welcher sich *einfach parallelisieren* lässt. (1 Punkt)

Mergesort auch möglich: Radix Sort

1

- d) Wie *heisst* der nachfolgende Sortieralgorithmus? (2 Punkte)

Insertion sort

2

```
public static void sort(final int[] a) {
    int elt;
    int j;

    for (int i = 1; i < a.length; i++) {
        elt = a[i];
        j = i;
        while ((j > 0) && (a[j - 1] > elt)) { // ←
            a[j] = a[j - 1];
            j--;
        }
        a[j] = elt;
    }
}
```

Die Schleife läuft so lange, wie das vorherige Element größer als das aktuelle ist. Verschiebt Elemente nach rechts um Platz zu machen.

- e) Was würde passieren, wenn in der mit dem Pfeil markierten Zeile neu folgende Bedingung stehen würde: $((j > 0) \&\& (a[j - 1] \leq elt))$ (2 Punkte)

Das Array würde unsortiert bleiben. Die Schleife würde abgebrochen, sobald ein kleineres oder gleiches Element gefunden wird. Der Algo funktioniert nicht mehr.

Korrekt, weil keine Verschiebung stattfindet.

Insert. Sort → läuft von links nach rechts, schreibt Werte nach links durch Vergleicht

Selection Sort → 2 Schleifen, sucht immer Minimum im unsortierten Bereich

5

Bubble Sort → Viele Swaps, vergleicht direkt benachbarte Werte & tauscht direkt

Quicksort → Pivot-Wahl, dann Partitionierung
HSLU

Seite 3/12

Heapsort

Mergesort → Teil in 2-er Hälften, führt dann merge() durch.

Selection Sort

weil am Ende nur ein Element übrig bleibt. Das ist automatisch korrekt.

```

for (int i = 0; i < array.length - 1; i++) { i läuft von 0 bis letzter Wert im Array.
    int minIndex = i; wir gehen davon aus, dass das kleinste Element an Position i ist.
    for (int j = i + 1; j < array.length; j++) { Array im sortierten Bereich links mit i & j im unsortierten Rechts. Suche nach kleinstem Element.
        if (array[j] < array[minIndex]) { wenn j kleiner als momentane minIndex
            minIndex = j; Dann ist neuer minIndex der Wert von j.
        }
    }
    swap(i, minIndex);
}

```

2 Schleifen & Minimum gesucht!

Bubble Sort

```

for (int i = 0; i < a.length - 1; i++) {
    for (int j = 0; j < a.length - i - 1; j++) { Bereich wird jedes Mal kleiner.
        if (a[j] > a[j+1]) {
            swap(j, j+1);
        }
    }
}

```

benachbarte Elemente werden verglichen & sofort getauscht.

Insertion Sort!

Ist der Name vom Array (Index!!!)

```

for (int i = 1; i < a.length; i++) { wir starten bei 1, weil 0 (erstes Element) bereits "sortiert" ist.
    int elt = a[i]; elt = element. Spezifiziert den Inhalt auf dem Index, womit es dann weitere Inhalte vergleicht auf Indexen links.
    int j = i; Das erste Element, das wir einsortieren sollen.
    while (j > 0 && a[j-1] > elt) { solange elt grösser ist als das Element links
        a[j] = a[j-1]; Verschiebung des grösseren Elements nach Rechts um einen Index
        j--;
    }
    a[j] = elt; Wir setzen elt am neuen richtigen Index ein.
}

```

Aufgabe 3: Komplexität (12 Punkte)

a) Bestimmen Sie die *Ordnung* bezüglich Laufzeit der nachfolgenden Methoden. (4 Punkte)

Hinweise:

- Die Methoden werden jeweils mit $n \geq 0$ aufgerufen.
- Die Hilfsmethode $\text{doA}(n)$ besitzt die Ordnung $O(1)$.
- Die Hilfsmethode $\text{doB}(n)$ besitzt die Ordnung $O(n)$.
- Die Hilfsmethode $\text{doC}(n)$ besitzt die Ordnung $O(n^2)$.

Hier genau auf n achten
& Zahlen. Gut lesen!
Verwirrungsgefahr

} Immer nebeneinander behalten.
Don't rush!!!

i:

```
public void method1(final int n) {
    doA(n); O(1)
    doB(5); O(n) → da n=5 ist es O(1)!!
    doC(n); O(n²)
}
```

$$O(1) + O(n) + O(n^2)$$

$$O(1)$$

Ordnung: $O(n^3)$

$$O(n^2)$$

$$O$$

ii:

```
public void method2(final int n) {
    for (int j = 0; j < n; j++) {
        doC(n + 1); O(n²)
    }
}
```

$$O(n) + O(n^2)$$

Ordnung: $O(n^3)$

$$1$$

iii:

1x aufgerufene Schleife

```
public void method3(final int n) {
    for (int i = 0; i < n; i++) { // O(n)
        for (int j = 1; j < (n + 7); j++) { // O(n)
            doA(n); O(1) n // O(1)
            doA(n); O(n) // O(1)
        }
    } O(n) + O(1) + O(n) + O(n)
}
```

Ordnung: $O(n^4)$

$$O(n^2)$$

$$0$$

iv:

```
public void method4(final int n) {
    for (int i = 0; i < 911; i++) {
        doA(i); O(1) × 911
    }
    doA(n); O(1)
}
```

$$O(1) + O(1)$$

Ordnung: $O(1)$

$$1$$

- b) Wir betrachten verschiedene Ordnungen bzw. deren Wachstum. Welche Ordnung wächst am schnellsten, zweitschnellsten usw. Geben Sie entsprechende Rangliste in der Spalte «Wachstum» an. (2 Punkte)

Ordnung	Wachstum 1. = schnellstes 7. = langsamstes	Beispiel gemäss Aufgabe c)
$O(n)$	4. 5.	P - R: Durchsuche mit Automaten
$O(\log n)$	6. ✓	R - U: Heap
$O(n^3)$	2. ✓	S ✓ Algo von Floyd
$O(n \cdot \log n)$	5. 4.	Q ✓ Mergesort Mergesort
$O(n^2)$	3. ✓	T ✓ Selection Sort
$O(1)$	7. ✓	U - P: Entfernen des kleinsten Elementes
$O(n!)$	1. ✓	V ✓ TSP

- c) Geben Sie in obiger Tabelle in der Spalte «Beispiel» je ein Beispiel an, indem Sie mit den Grossbuchstaben (P bis V) auf nachfolgende Algorithmen referenzieren. (3 Punkte)

P: Entfernen des kleinsten Datenelementes aus einer geordneten Liste

Q: Mergesort

R: Durchsuchen einer Zeichenkette mit einem optimierten Suchautomaten

S: Algorithmus von Floyd

T: Selection Sort

U: Einfügen eines Datenelementes in die Datenstruktur Heap

V: TSP (Travelling Salesman Problem)

1

0

- d) Bei drei verschiedenen Algorithmen misst man für $n = 10'000$ je eine Rechenzeit von ca. 10 Millisekunden. Die Algorithmen verhalten sich für dieses n gemäss ihrer Ordnung. Geben Sie an, was für Rechenzeiten etwa für $n = 100'000$ zu erwarten sind. (3 Punkte)

	Ordnung	Rechenzeit	
		$n = 10'000$	$n = 100'000$
i:	$O(n)$	10 ms	100ms $\quad 10\text{ms} \cdot 10$
ii:	$O(n^3)$	10 ms	300ms $\left(\frac{100'000}{10'000}\right)^3 \rightarrow 10^3 = 1000$
iii:	$O(n \cdot \log n)$	10 ms	50ms $10\text{ms} \cdot 12.5 = 125\text{ms}$

1

0

0

0.4

4

sucht zuerst alle Nodes die auf der momentanen Tiefe bevor er nach Tiefer sucht.
es gibt auch **BFS**: Breadth-First-Search (meistens auf Basis von Queue)

Modulendprüfung AD im HS 2024 – Teil 3: Algorithmen geht so tief wie möglich auf einem AST & dann zurück & sucht nächste

(Stichwort: Tiefensuche (**DFS**: Depth-First-Search))

Aufgabe 4: Rekursion und Graphen-Algorithmen (11 Punkte)

Folgender Graph ist vorgegeben, welcher die untenstehende Klasse Rekursion erzeugt:

- $B \rightarrow C, D, E, F$

- $C \rightarrow$ keine weiteren Knoten

- $D \rightarrow G$

- $G \rightarrow$ keine weiteren Knoten

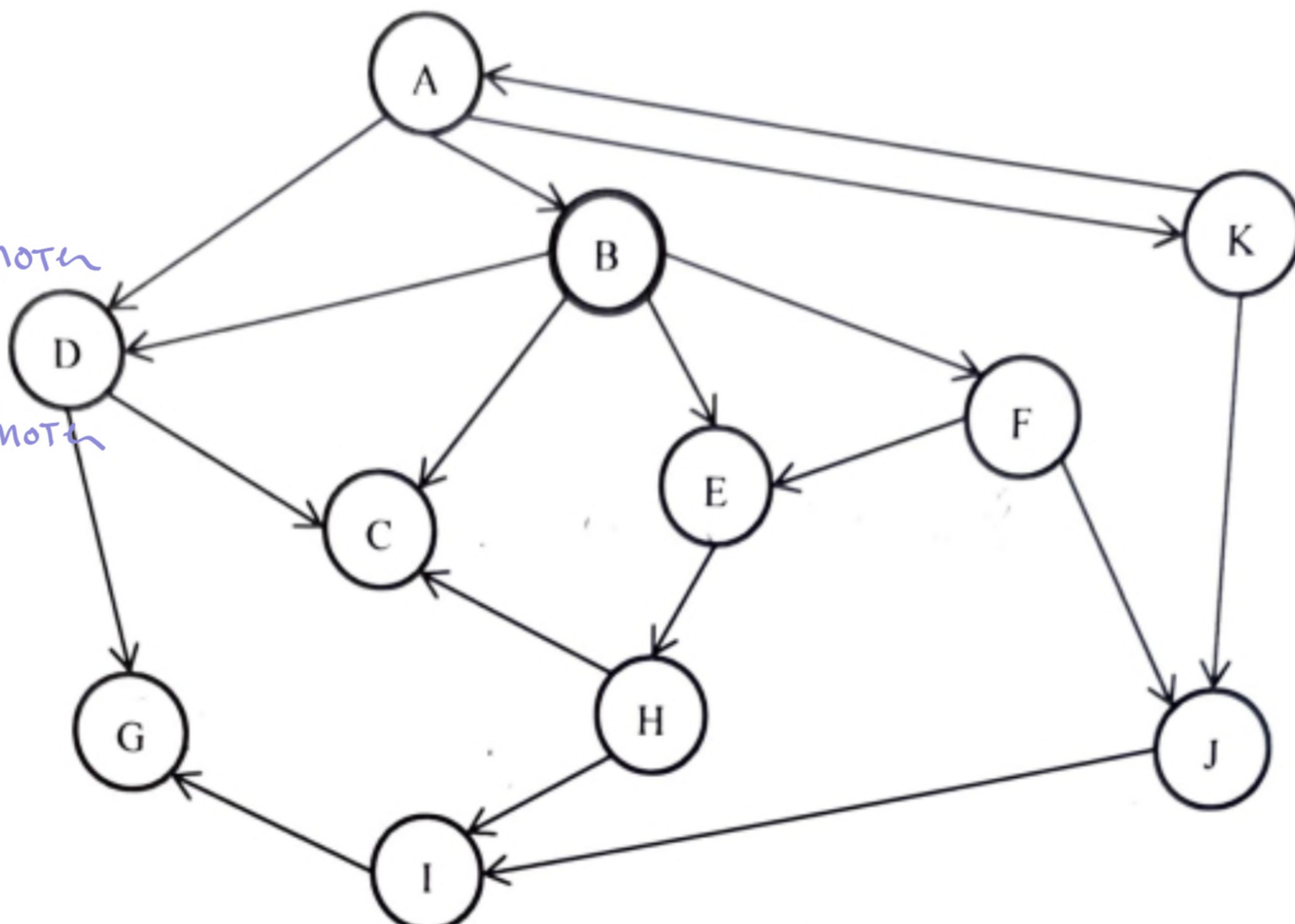
- $E \rightarrow H, (C)$

- $H \rightarrow I$

- $I \rightarrow (G)$

- $F \rightarrow (E), J$

- $J \rightarrow I$



Zur Hilfe: alphabetische Reihenfolge der 11 Knoten: **A B C D E F G H I J K**

```
public class Rekursion {
```

```

    public static void traverseGraph(final Graph g, final Node start) {
        1. Knoten wird besucht (→ gray) start.setColor(Color.GRAY);           // Node entdeckt
        2. Name wird ausgegeben. System.out.println(start.getName());           // Node verarbeiten bzw. ausgeben
        3. Knoten wird verarbeitet (→ black) start.setColor(Color.BLACK);       // Node verarbeitet
        4. Für alle benachbarten Knoten for (Node n : g.getAllAdjNodes(start)) { // n's in alphabetischer Reihenfolge
            in alphabetischer Reihenfolge, die noch if (n.getColor().equals(Color.WHITE)) {
            weiss sind, dasselbe machen.         traverseGraph(g, n);
            }
        }
    }

    call 1
    public static void main(String[] args) {
        Graph graph = new Graph();                                         // Graph erzeugen
        Node a = new Node("A");                                            // Knoten "A", "B" usw. erzeugen
        Node b = new Node("B");                                            // Knoten und Kanten hinzufügen
        ...
        call 2
        traverseGraph(graph, b);                                           // Graph von "B" aus traversieren
    }
}

Der Graph wird von Knoten B aus rekursiv durchlaufen.
```

- a) In welcher Reihenfolge werden die Knoten beim Aufruf von `traverseGraph(graph, b)` ausgegeben? (4 Punkte)

1. (zuerst)	2.	3.	4.	5.	usw.				
B	C	D	G	E	H	I	F	J	

2

- b) Gemäss welcher Suche werden die Knoten besucht? (1 Punkt)

DFS - Tiefensuche

0

= Aufruf Methode rekursiv

- c) Wie viele «Stack Frames» der Methode `traverseGraph()` liegen während der Abarbeitung von `traverseGraph(graph, b)` im Maximum gleichzeitig auf dem «Call Stack»? Machen Sie dazu auch eine nachvollziehbare Illustration. (3 Punkte) → im Max: also beim Tiefsten Pfad.
Das ist hier $B \rightarrow E \rightarrow H \rightarrow I$

Es liegen ca. 5 Stack Frames.

0

Illustration:

- Stack 1: Main-Methode Aufruf
- 2: `TraverseGraph(B)`
- 3: `TraverseGraph(E)`
- 4: `TraverseGraph(H)`
- 5: `TraverseGraph(I)`

1!

- d) Die rekursive Methode liesse sich auch relativ einfach iterativ implementieren. Welche Datenstruktur wäre dabei zentral? (1 Punkt)

Stack (bei BFS wäre es Queue)

0

Teilgraph eines zusammenhängenden, ungerichteten Graphen.

- e) Findet man mit `traverseGraph()` in unserem Beispiel auch einen Spannbaum (ja/nein)? Begründen Sie Ihre Antwort. (2 Punkte)

Nein, weil nicht alle Knoten erreicht werden + gibt es zwischen A & K einen Zyklus.

Ein Spannbaum ist ein Teilgraph eines (zusammenhängenden) ungerichteten Graphen, der folgende Bedingungen erfüllt:	
Merkmal	Erklärung
Alle Knoten enthalten	Der Spannbaum enthält alle Knoten des ursprünglichen Graphen
Zusammenhängend	Es gibt einen Pfad zwischen allen Knoten im Spannbaum
Zyklusfrei	Es gibt keinen Zyklus – also keine Rückverbindungen
Hat genau $n-1$ Kanten	Ein Spannbaum mit n Knoten hat immer genau $n-1$ Kanten

Modulendprüfung AD im HS 2024 – Teil 3: Algorithmen

= Deterministischer endlicher Automat (DFA). Springt nicht zurück.

Aufgabe 5: Optimierter Suchautomat (7 Punkte)

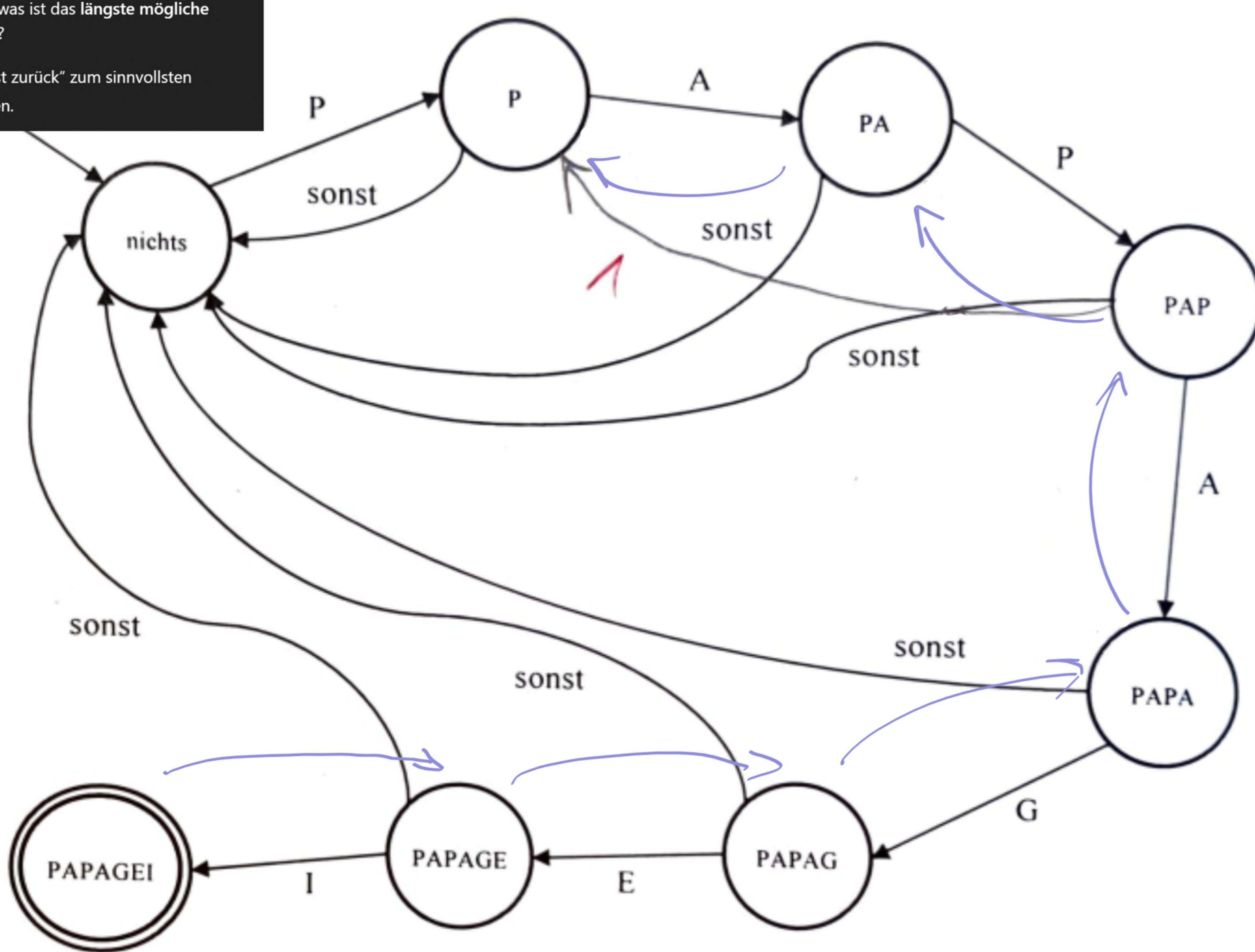
Mit Hilfe eines "optimierten Suchautomaten" möchte man nach dem Pattern "PAPAGEI" suchen:

Grundprinzip: "Was ist der längste sinnvolle Rücksprung?"

Bei jedem Zustand fragst du dich:

Wenn das nächste erwartete Zeichen fehlt, was ist das längste mögliche Teilwort, mit dem wir weitermachen können?

Das ist wie beim KMP-Algorithmus: Du „springst zurück“ zum sinnvollsten früheren Zustand, statt ganz von vorne zu starten.



1

- a) Vervollständigen Sie beim obigen Automaten die fehlenden Zustandsübergänge. Das Alphabet beinhaltet alle Symbole der deutschen Sprache. (6 Punkte)

Hinweis: Ein falsch eingezeichneter Übergang ergibt bei der Korrektur einen Abzug. Sie erhalten bei dieser Aufgabe insgesamt aber *keine* negativen Punkte.

- b) Inwiefern ist das Suchen mit einem "optimierten Suchautomaten" vorteilhaft? Geben Sie einen klaren Vorteil an. (1 Punkt)

Man kann damit in linearer Zeit suchen ($O(n)$). Muss das

0

Eingabewort nur einmal durchlaufen, kein Zurücksspringen nötig.

—
1

Aufgabe 6: Transitive Hülle (4 Punkte)

gibt es einen Weg zwischen 2 Knoten, unabhängig wieviele Kanten dazwischen liegen?

```
public class GraphAlgo {
    private int noOfNodes;
    private String[] nodeName;
    private boolean[][] adjaMx;

    GraphAlgo(final String[] nodeName, final boolean[][] adjaMx) {
        this.noOfNodes = nodeName.length;
        this.nodeName = nodeName;
        this.adjaMx = adjaMx;
    }

    ...
}
```

Kopieren der
Adjazenzmatrix.
Wie dürfen nicht
auf das Original
schreiben.

```
public boolean[][] transitiveClosure() {
    boolean[][] transClosure = new boolean[noOfNodes][noOfNodes];
    for (int n = 0; n < noOfNodes; n++) {
        transClosure[n] = Arrays.copyOf(adjaMx[n], noOfNodes);
    } // Name der Kopie der Matrix.
    for (int y = 0; y < noOfNodes; y++) { // Für jeden Knoten Prüfung, ob es eine Zwischenknoten ist auf dem Weg zu anderen Knoten.
        for (int i = 0; i < noOfNodes; i++) { // Für jeden Startknoten
            if (transClosure[i][y]) { // Gibt es einen Weg von i nach y?
                // für jeden Ziellknoten for (int j = 0; j < noOfNodes; j++) {
                    if (transClosure[...y...][...j...]) { // Gibt es einen Weg y nach j? Wenn ja, dann
                        transClosure[...i...][...j...] = true; // gibt es auch einen Weg von i nach j.
                    }
                }
            }
        }
    }
    return transClosure; // Information, ob ein direkter oder indirekter Weg existiert.
}

...
}
```

a) Ergänzen Sie oben bei allen Punkten (...) den fehlenden Code. (3 Punkte)

b) Weshalb ist die Methode transitiveClosure() für grosse Graphen *nicht praktikabel*? (1 Punkt)

Weil die Methode 3 verschachtelte Schleifen verwendet mit $O(n^3)$. Bei grossen Graphen (viele Knoten) dauert das sehr lange und ist nicht effizient.

9,5!
1,5

Aufgabe 7: Syntaxdiagramm (8 Punkte)

Hinweis: Die Fragen d) und e) und können unabhängig von den vorangehenden beantwortet werden!

Mit Hilfe eines Syntaxdiagramms **Preisangabe** möchte man definieren, wie eine Preisangabe genau anzugeben ist. Anmerkung: Eine korrekte Preisangabe entspricht einem Wort w einer entsprechenden formalen Sprache $L(\Lambda)$.

Beispiele für korrekte Preisangaben: Fr. 1.50

Fr. 0.55

Fr. 10.-

Fr. 99.95

Fr. 0.00

Beispiele für falsche Preisangaben:

1.50

Fr. fehlt.

Fr. 1.51

Die Preisangabe endet nicht auf '0' oder '5'.

Fr.05.50

Eine führende '0' ist unzulässig, Abstand fehlt.

Fr. 1.5

Rappen-Angabe muss zweistellig sein (oder '-').

Fr. 100.-

Franken-Angabe darf höchstens zweistellig sein.

Das Alphabet A kennt folgende 14 Terminalsymbole:

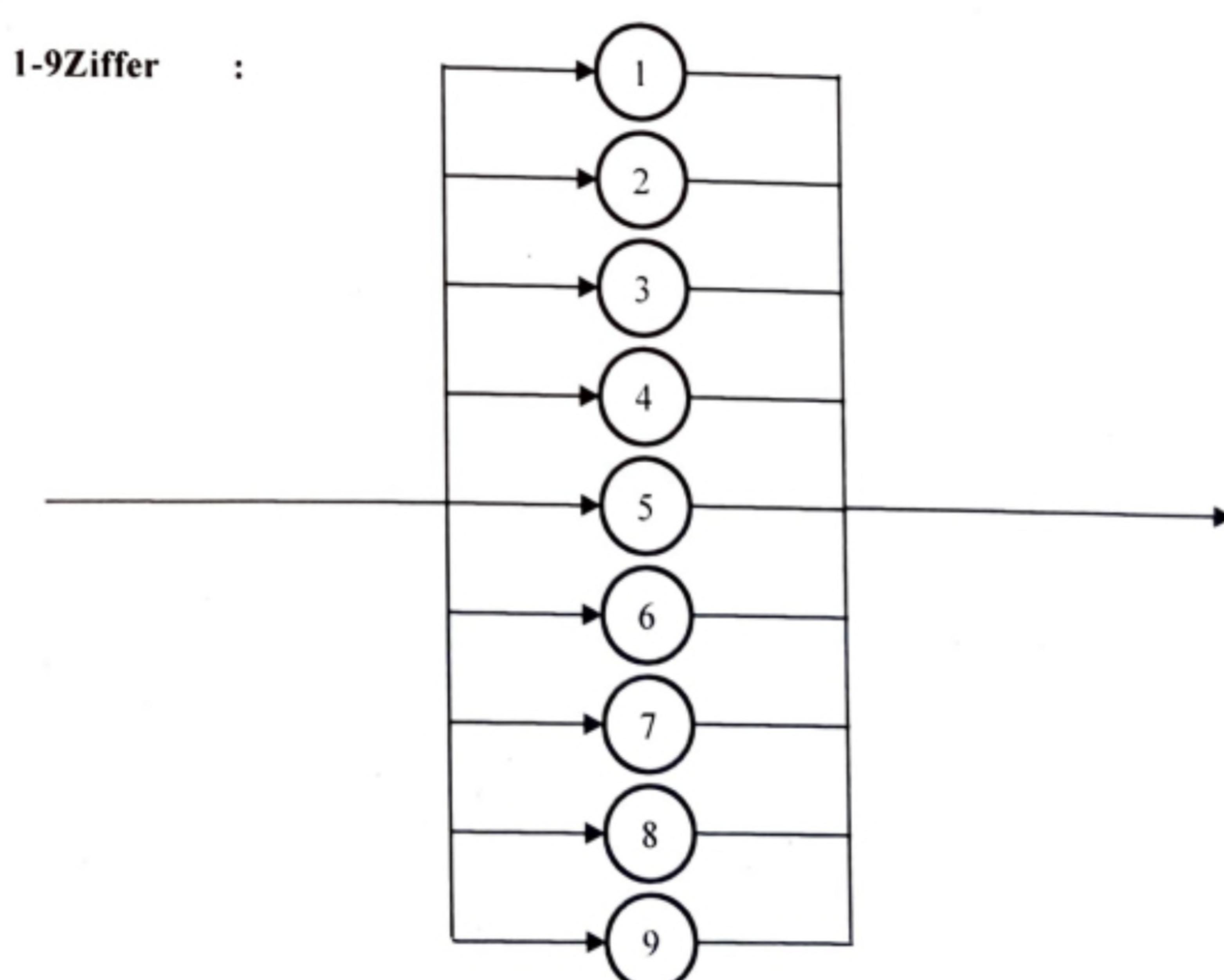
$$A = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \text{Fr.}, ., -, \cdot \}$$

Anmerkung: Das letzte Symbol '·' steht für Abstand bzw. Space.

Das Syntaxdiagramm **Preisangabe** ist bereits wie folgt vorgegeben:

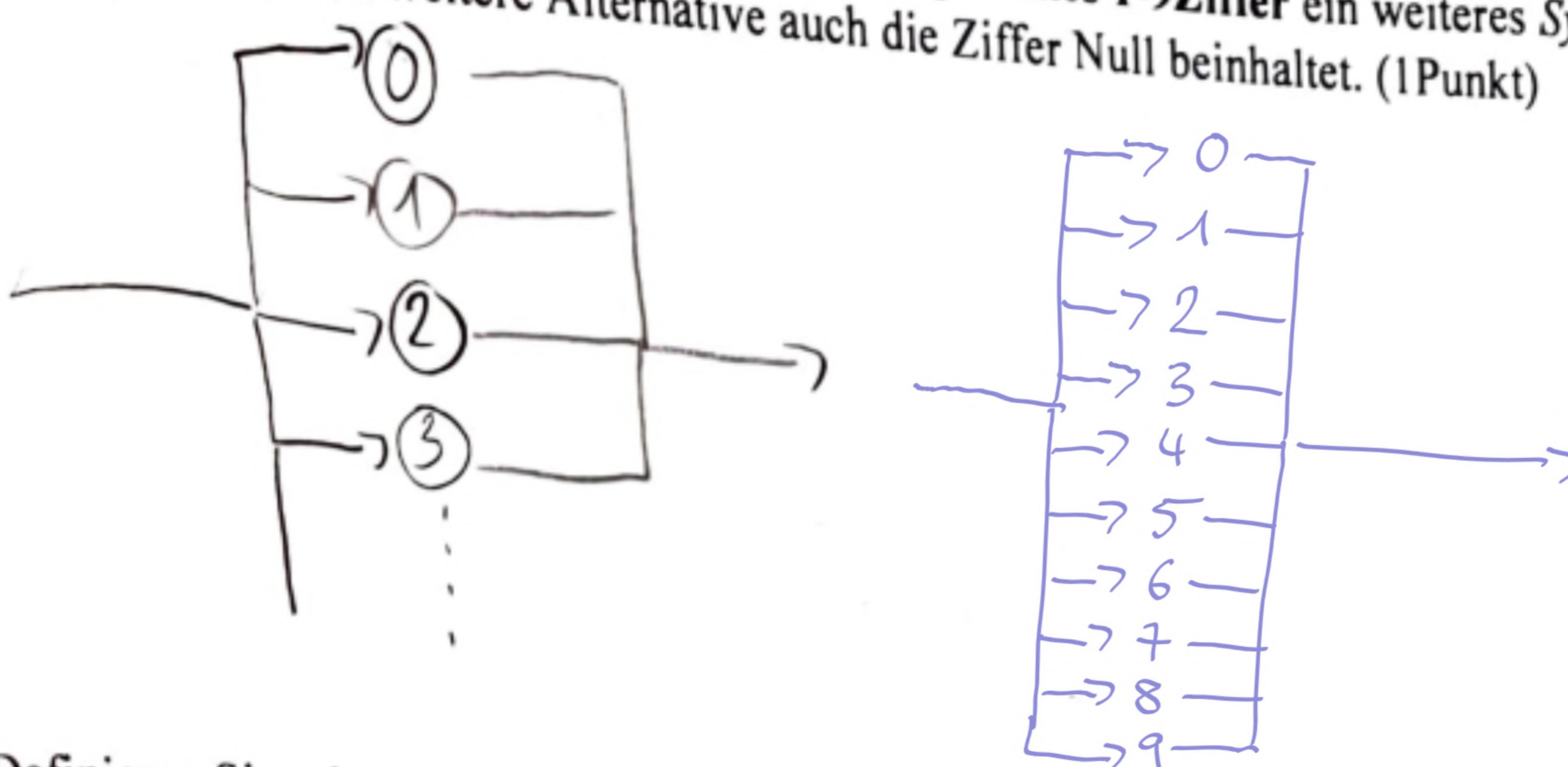


Ebenso ist ein Syntaxdiagramm **1-9Ziffer** vordefiniert:



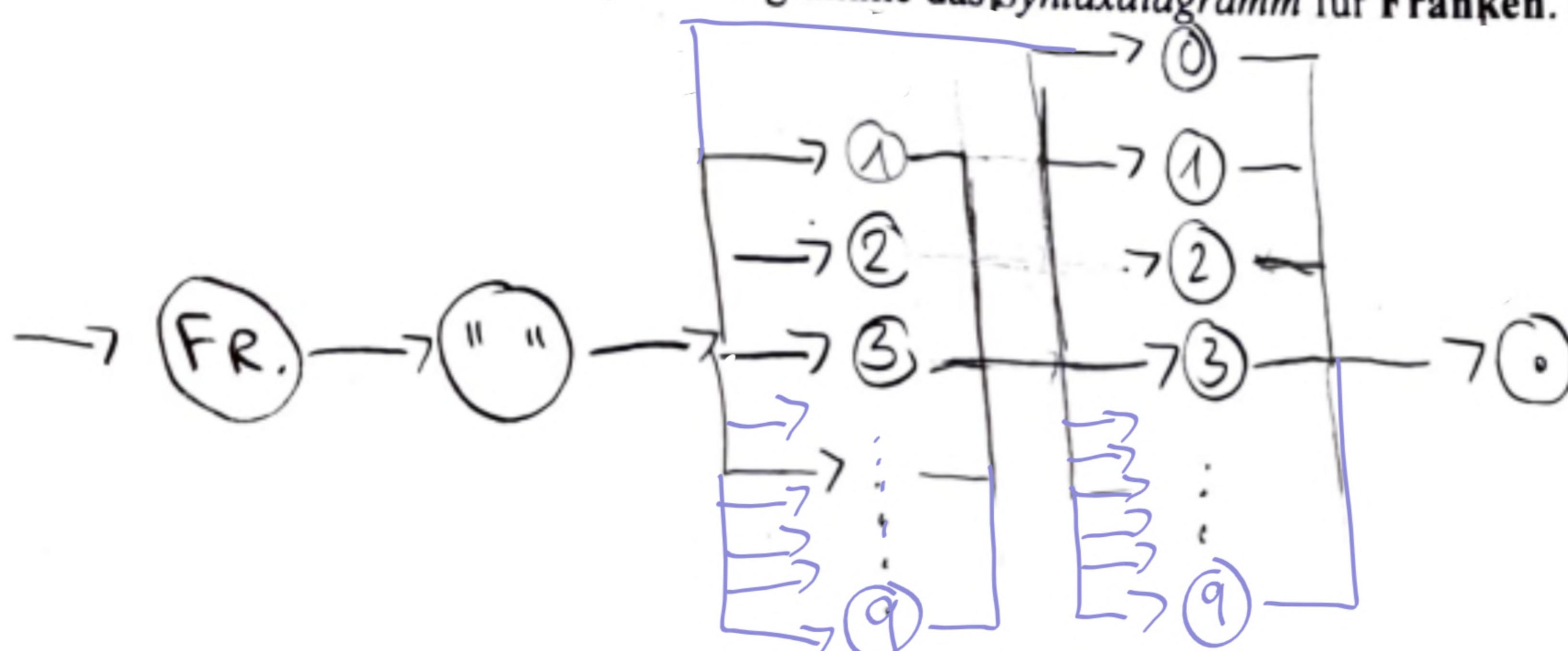
Auszeichnen

- a) Definieren Sie mit Gebrauch des obigen Syntaxdiagrammes 1-9Ziffer, welches als weitere Alternative auch die Ziffer Null beinhaltet. (1 Punkt)



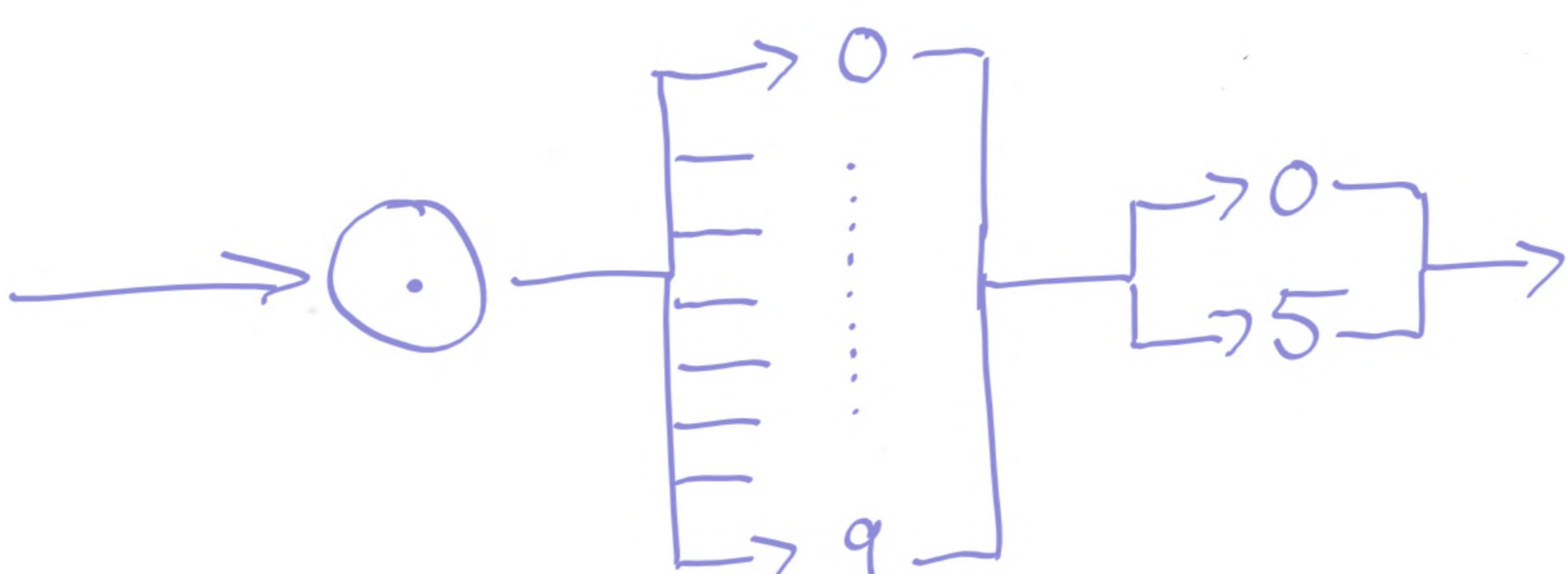
0

- b) Definieren Sie mit Hilfe der Syntaxdiagramme das Syntaxdiagramm für Franken. (2 Punkte)



0

- c) Definieren Sie das fehlende Syntaxdiagramm für Rappen. (3 Punkte)



0

- d) Geben Sie eine alternative Möglichkeit zum Syntaxdiagramm an, womit man die Grammatik dieser formalen Sprache auch definieren könnte. (1 Punkt)

EBCN = erweiterte Backus-Naur-Förm

1

- e) In dieser Aufgabe kam ein Lösungsprinzip zur Anwendung, welches generell bei Algorithmen von Bedeutung ist. Wie heisst dieses Prinzip? (1 Punkt)

Verwendung von Grammatiken
↓ Transform & Conquer

0

1

Aufgabe 8: Rekursion und Methoden-Implementation (7 Punkte)

Folgendes Rekursionsschema definiert eine Funktion $f(a, b)$ für natürliche Zahlen a und b :

$$f(a, b) = f(a, b-1) + a \quad \text{bedeutet: Die Funktion berechnet } a+a+\dots+a \text{ insgesamt } b-\text{mal!} \Rightarrow f(a, b) = a \cdot b$$

$$f(a, 0) = 0$$

- a) Berechnen Sie die *Funktionswerte*: (2 Punkte)

$$f(4, 0) = \dots \quad 0$$

$$\begin{array}{l} a=4 \\ b=0 \end{array} \left\{ \begin{array}{l} f(4, -1) + 4 \\ \text{Stack 1} \end{array} \right.$$

$$f(4, 1) = f(4, 0) + 4 = 4$$

$$\begin{array}{l} a=4 \\ b=1 \end{array} \left\{ \begin{array}{l} f(4, 1) + 4 \\ 4 \\ , f(4, 0) + 0 \end{array} \right. \rightarrow = 4$$

$$f(4, 2) = f(4, 1) + 4 = 8$$

$$\begin{array}{l} a=4 \\ b=2 \end{array} \left\{ \begin{array}{l} f(4, 2) + 4 \\ 4 \\ , f(4, 1) + 4 \\ 4 \end{array} \right. \rightarrow = 8$$

- Void gibt nichts zurück!!*
- b) Implementieren Sie entsprechend eine rekursive Methode namens `funcRec(...)`, und zwar als öffentliche Klassenmethode. Sie können davon ausgehen, dass beide Parameter ≥ 0 sind. (3 Punkte)

```
public static int funcRec(int a, int b) {
    if (b == 0) {
        return 0;
    } else {
        return funcRec(a, b - 1) + a;
    }
}
```

0, 5

0, 5

0

- c) Sie haben wohl herausgefunden, um welche grundlegende mathematische Funktion es oben geht. Implementieren Sie nochmals eine gleichwertige Methode, aber jetzt viel einfacher. (2 Punkte)

Da wir erkannt haben, dass $f(a, b) = a \cdot b$ ist, implementieren wir das nun ohne Rekursion.

```
public static int funcSimple(int a, int b) {
    return a * b;
}
```

0

---- Ende dieses Prüfungsteils ----

Algorithmen & Datenstrukturen (AD)

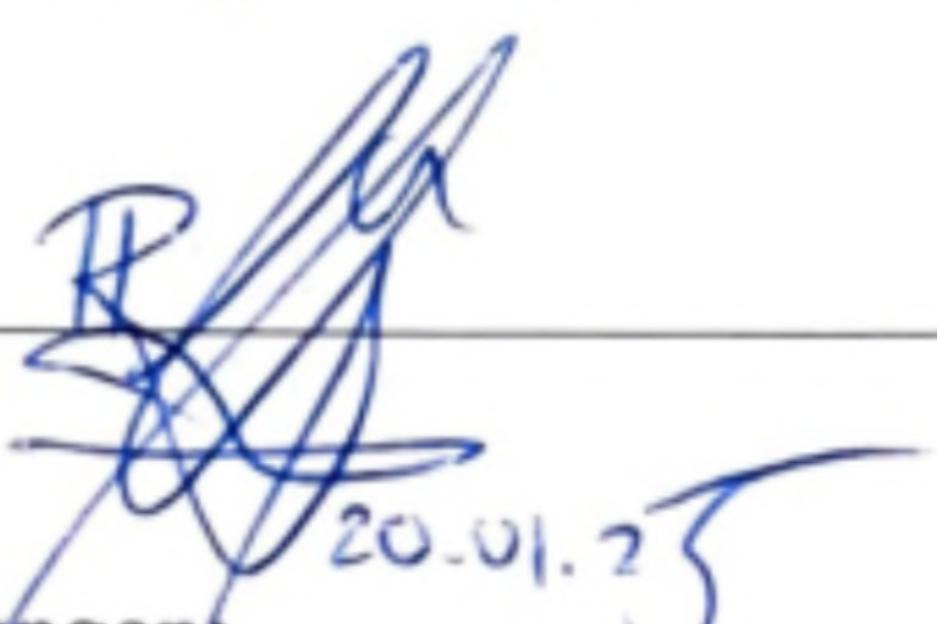
Modulendprüfung AD - HS 2024

Teil 2 von 3 – Nebenläufigkeit

Roger Diehl – Version 1.0

Name: Degtyareva
(Bitte mit Druckbuchstaben schreiben)

Vorname: Dominika

Unterschrift: 

Rahmenbedingungen:

1. Prüfungszeit: total 180 Minuten, für diesen Teil ca. **50** Minuten.
2. Diese Prüfung besteht aus drei Teilen und es können 160 Punkte erreicht werden, davon in diesem Prüfungsteil **50** Punkte. Jeder (Teil-)Aufgabe ist eine maximal erreichbare Punktzahl zugeordnet.
3. Schreiben Sie Ihren Namen und Vornamen mit Druckbuchstaben oben auf dieses Blatt. Mit der Unterschrift bezeugen Sie, dass Sie diesen Prüfungsteil persönlich und nur mit erlaubten Hilfsmitteln bearbeitet haben. Blätter ohne diese Angaben werden nicht bewertet.
4. Es handelt sich um eine schriftliche Prüfung ohne Einsatz des Computers oder elektronischer Hilfsmittel. Sie dürfen keine Unterlagen verwenden.
5. Sollte eine Aufgabenstellung Unklarheiten aufweisen, treffen und dokumentieren Sie Annahmen.
6. Schreiben Sie verständlich und gut leserlich. Missverständliche Lösungen werden nicht berücksichtigt.
7. Benutzen Sie den Freiraum unter den Aufgaben für Ihre Lösung.

Gutes Gelingen & viel Erfolg!

Für die Korrektur (nicht ausfüllen!)

1	2	3	4	5	6	7	Punkte	Visum
1	4	1	4	1	-	2	1	14

Prolog: Interface CountDownLatch

Für alle Aufgaben dieses Prüfungsteils ist das Interface CountDownLatch vorgegeben. Bitte lesen Sie die folgende Java Dokumentation durch.

```
1  /**
2   * Ein CountDownLatch ist eine einfache Schranke, an der beliebig viele
3   * Threads warten können. Bei der Erzeugung wird ein Startwert für einen
4   * internen Zähler mitgegeben. Sobald der Wert 0 erreicht wird, öffnet
5   * sich die Schranke und alle wartenden Threads können weiterlaufen.
6   */
7  public interface CountDownLatch {
8
9    /**
10     * Bewirkt, dass der aktuelle Thread wartet, bis der Latch Zähler
11     * auf 0 heruntergezählt hat, sofern der Thread nicht unterbrochen
12     * wird. Ist der Zähler 0, können Threads passieren.
13     *
14     * @throws java.lang.InterruptedException wenn der aktuelle Thread
15     * während des Wartens unterbrochen wird.
16     */
17    void await() throws InterruptedException;
18
19    /**
20     * Dekrementiert den Latch Zähler und gibt alle wartenden Threads
21     * frei, wenn der Zählerstand 0 erreicht.
22     */
23    void countDown();
24
25    /**
26     * Gibt den Latch Zählerstand zurück.
27     *
28     * @return Zählerstand.
29     */
30    int getCount();
31
32 }
```



Ich habe die Java Dokumentation durchgelesen. (1 Punkt)



sleep() nutzt nicht dieses Prinzip!

Aufgabe 2: Guarded Blocks (5 Punkte)

- a) Ergänzen und korrigieren Sie in der Klasse CountDownLatchImpl die Methode:

```
public void await() throws InterruptedException
```

Für die Methode gilt folgende Funktionalität (4 Punkte):

- Die Implementation ist gemäss den Anforderungen aus der Java Dokumentation CountDownLatch umzusetzen (siehe Prolog). ~~else throw Exception~~
- Die Methode ist blockierend, wenn der Zähler grösser als 0 ist, d.h. Threads müssen warten. ~~join/sleep~~
- Das InterruptedException Handling dürfen Sie weglassen.

```

public class CountDownLatchImpl implements CountDownLatch {

    private int countValue;

    @Override
    public void await() throws InterruptedException {
        !
• while ✓
        if (countValue > 0) {
            // hier muss der Thread warten
            wait();
        }
    }

    public synchronized void countDown() {
        if (countValue > 0) {
            countValue--;
        }
        if (countValue == 0) {
            notifyAll(); notwendig um wait() & notifyAll() aufzurufen.
        }
    }

    // Weitere Implementationen aus dem Interface...
}

```

Pseudocode

- b) Markieren Sie deutlich die Stelle in Ihrem Code, wo die Threads vom Running-Zustand in den Object-Wait-Pool versetzt werden. (1 Punkt)

In der Zeile `wait()`

Hier wechselt der Thread vom Running-Zustand in den Waiting-Zustand.

Aufgabe 3: Thread Start (8 Punkte)

Gegeben ist das Programm HorseRace, welches ein Pferderennen simuliert.

Tipp: Beachten Sie das Interface CountDownLatch (siehe Prolog).

Tipp: Ihnen steht die API-Dokumentation im Anhang zur Verfügung.

- a) Wie viele Threads werden für die Pferde gestartet? Geben Sie die Zeilenummer an. (2 Punkte)

5 Threads ✓

- b) Wofür wird der CountDownLatch startLine (Definition in Zeile 14) gebraucht? (2 Punkte)

STARTLine blockiert den Main-Thread auf Zeile 12, bis alle Pferde bereit sind. Jeder Horse ruft in run() (Zeile 43)

- c) Wofür wird der CountDownLatch startSignal (Definition in Zeile 5) gebraucht? (2 Punkte)

Ist sozusagen die Variable countValue von vorher, die herunterzählt bis STARTLine "geöffnet" wird. ✓
→ wer?

- d) Wenn das Programm HorseRace gestartet wird, passiert folgendes. Man sieht, dass die Pferde bereit sind (LOG.info Zeile 41), das Rennen wird gestartet (LOG.info Zeile 15) und dann ist das Programm zu Ende.

```
2025-01-03 13:30:35.560 [virtual-29] INFO - Horse 5 ist bereit...
2025-01-03 13:30:35.560 [virtual-24] INFO - Horse 1 ist bereit...
2025-01-03 13:30:35.560 [virtual-25] INFO - Horse 2 ist bereit...
2025-01-03 13:30:35.560 [virtual-28] INFO - Horse 4 ist bereit...
2025-01-03 13:30:35.560 [virtual-27] INFO - Horse 3 ist bereit...
2025-01-03 13:30:35.614 [main] INFO - Los!
```

BUILD SUCCESS

Warum sieht man die Pferde nicht im Ziel (LOG.info Zeile 47)? (2 Punkte)

Es werden virtuelle Threads gestartet. Das Hauptprogramm (Main) wartet nicht auf deren Ende. JVM beendet sich, bevor alle Threads fertig sind.

```

1 public class HorseRace { // Hier wird das Rennen vorbereitet & gestartet.
2
3     private static final Logger LOG =
4         LoggerFactory.getLogger(HorseRace.class); // Logger für Infos im Log zu schreiben
5     private static final int HORSES = 5; // Hier werden Threadanzahl angegeben. Es laufen 5 Pferde
6                                                 // gegen einander.
7     public static void main(final String[] args) { Warteschranke in Main().Wartet, bis alle
8         final CountDownLatch startLine = new CountDownLatchImpl(HORSES); Pferde bereit sind.
9         final CountDownLatch startSignal = new CountDownLatchImpl(1); // 1 Startsignal für
10        for (int i = 1; i <= HORSES; i++) { // Schleife erzeugt bis 5 Horses, alle Pferde gleichzeitig
11            Thread.startVirtualThread( Erzeugt ein Horse-Objekt, das einen Namen hat & Zugriff auf startLine & startSignal hat.
12                new Horse("Horse " + i, startLine, startSignal));
13            }
14        startLine.await(); // Main-Thread blockiert hier, bis alle Pferde .countDown() auf startLine gemacht haben.
15        LOG.info("Los!");
16        startSignal.countDown(); // Hier beginnt für alle Pferde das Rennen.
17    }
18 }

```

```

23 public final class Horse implements Runnable { // Wie verhält sich ein Pferd?
24
25     private static final Logger LOG =
26         LoggerFactory.getLogger(Horse.class);
27     private final String name; // Name des Pferdes
28     private final CountDownLatch startSignal; // Schranke für "bereit"
29     private final CountDownLatch startLine; // Startsignal
30
31     public Horse(final String name, // Konstruktor
32                  final CountDownLatch startLine,
33                  final CountDownLatch startSignal) {
34         this.name = name;
35         this.startLine = startLine;
36         this.startSignal = startSignal;
37     }
38
39     @Override
40     public void run() { es braucht Runnable für run()!!
41         LOG.info("{} ist bereit...", name);
42         try {
43             startLine.countDown(); // Sagt dem startLine, dass dieses Pferd bereit ist.
44             startSignal.await(); // Pferd wartet hier auf das Startsignal von main()
45             int time = 10000 + ThreadLocalRandom.current().nextInt(5000);
46             TimeUnit.MILLISECONDS.sleep(time); // Pferd schläft (=rennt) für 10-15s. Simuliert Random
47             LOG.info("{} ist in {} Sec im Ziel ", name, time / 1000f); Dauer des Rennens
48         } catch (InterruptedException ex) {
49             LOG.info("Wettkampf abgebrochen!");
50         }
51     }
52 }

```

Aufgabe 4: Thread Pools (7 Punkte)

Kreuzen Sie in den Tabellen die richtigen Felder an. Es sind auch mehrere Kreuze pro Zeile möglich. Für alle Aussagen gilt, dass die Aufgaben nebenläufig ausgeführt werden.

Tipp: Ihnen steht die API-Dokumentation im Anhang zur Verfügung.

EXECUTORS.newCachedThreadPool(): erzeugt neue Threads nach Bedarf.
Nur bei vielen I/O-Tasks, die kurz sind.

EXECUTORS.newFixedThreadPool(n): Feste Anzahl Threads. Bei rechenintensiven Tasks.

EXECUTORS.newSingleThreadExecutor(): Nur 1 einziger Thread. Aufgaben nur nacheinander abgearbeitet.

EXECUTORS.newVirtualThreadPerTaskExecutor(): Wenn man 1000 gleichzeitige Tasks hat.

Java Objekt, dass Threads startet & verwaltet. Oft über einen Thread Pool. Man übergibt ihm entweder **Runnable** oder **Callable** & es entscheidet, wann & wie ausgeführt wird.

Welchen Executor wählen Sie?

Was ist das? Annahme: generell

Für Aufgaben, die viele I/O-Operationen ausführen, aber nur von kurzer Dauer sind.

Operations
Input/Output. Alle Aktionen, bei denen ein Programm Daten liest & schreibt.

Für Aufgaben, die einen rechenintensiven Algorithmus ausführen, um möglichst schnell ein Resultat zu finden.

Wenn Aufgaben nur einen Teil der Prozessorkerne auslasten dürfen.

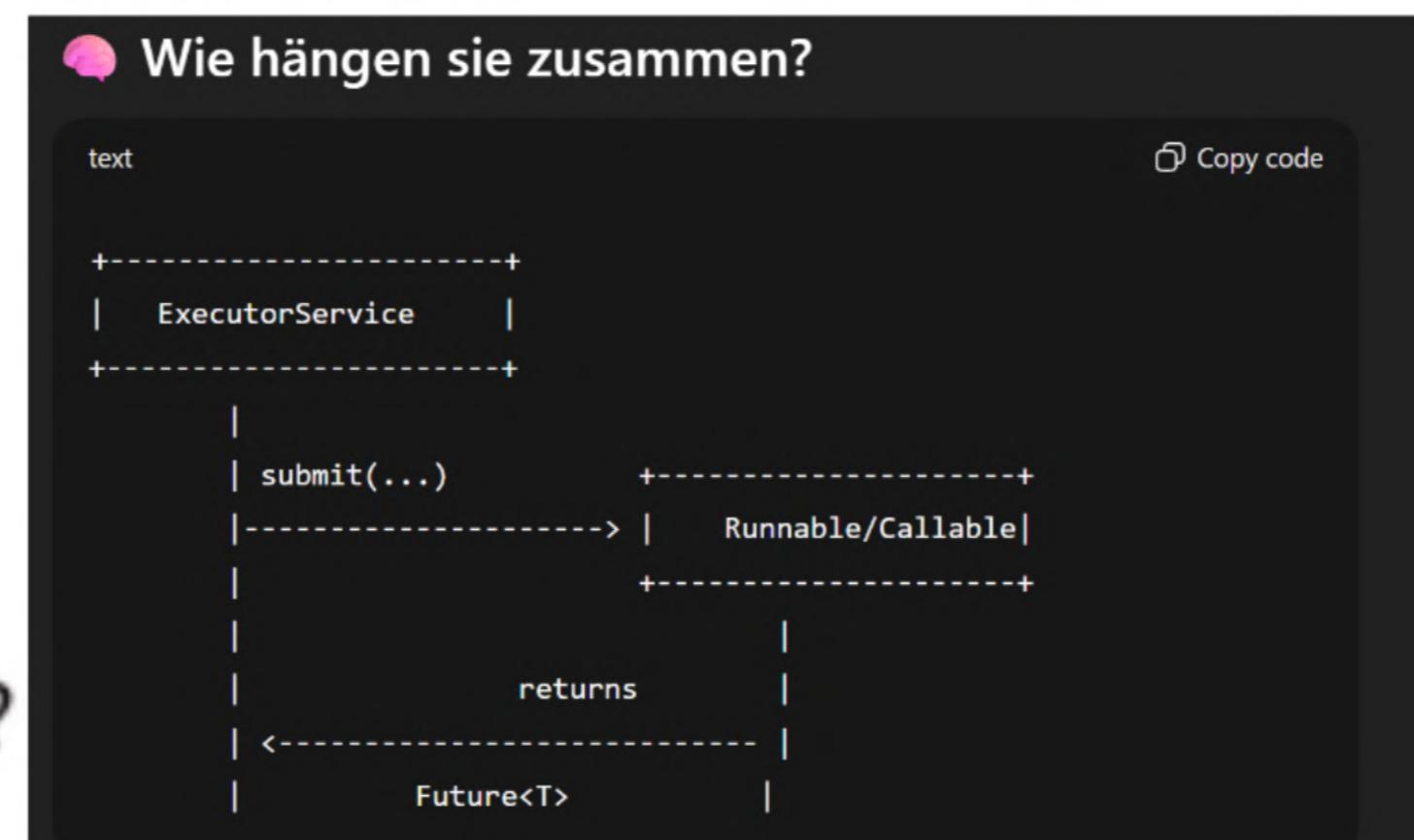
Für kurze Aufgaben in einer Serveranwendung, um möglichst viele (tausende) Clients gleichzeitig zu bedienen.

Für die Hauptaufgabe einer Applikation, die nebenläufig ausgeführt wird und solange läuft, wie die Applikation.

→ da Hintergrundaufgabe

Begriff	Merksatz
Runnable	„Tu etwas“ – ohne Rückgabewert
Callable<T>	„Tu etwas und gib mir was zurück“
Future<T>	„Ich warte auf das Ergebnis (von Callable z.B.)“
ExecutorService	„Ich verwalte Threads für dich“

Welche Executor Methode wählen Sie?



Für Aufgaben, die das Interface **Callable** implementieren.

Für Aufgaben, die das Interface **Runnable** implementieren.

Aufgabe 5: Atomare (Atomic) Datentypen (8 Punkte)

Tipp: Ihnen steht die API-Dokumentation im Anhang zur Verfügung.

- a) Ergänzen Sie in der Klasse ~~AtomicCountDownLatch~~ im Konstruktor die Platzhalter ① und ② sowie alle notwendigen Attribute gemäss den Anforderungen aus der Java Dokumentation CountDownLatch (siehe Prolog). (3 Punkte)

- b) Implementieren Sie in der Klasse ~~AtomicCountDownLatch~~ die Methoden (5 Punkte):

```
public void await() throws InterruptedException
public int getCount()
```

Für die Methoden gilt folgende Funktionalität:

- Die Implementation ist gemäss den Anforderungen aus der Java Dokumentation CountDownLatch umzusetzen (siehe Prolog).
- Die Methoden sind Thread-sicher.
- Das InterruptedException Handling dürfen Sie weglassen.

```
public class AtomicCountDownLatch implements CountDownLatch {
    private final AtomicInteger count;

    public AtomicModuloCounter(final int countValue) {
        This.count = countValue;
        ① = new AtomicInteger(②);
    }

    @Override
    public void await() throws InterruptedException {
        while (count.get() > 0) {
            Thread.sleep(1);
        }
    }

    @Override
    public int getCount() {
        Return count.get();
    }

    // Weitere Implementationen aus dem Interface...
}
```

STICHWORT

Aufgabe 6: Callable und Future (9 Punkte)

Die Klasse `AthleteTask` und die Klasse `SprintRace`, simulieren einen 100-Meter-Lauf.

Tipp: Beachten Sie das Interface `CountDownLatch` (siehe Prolog).

Tipp: Ihnen steht die API-Dokumentation im Anhang zur Verfügung.

Beantworten Sie die folgenden Fragen:

Antwort:

a) Was steht im lauffähigen Code an der Stelle des Platzhalters ① in den Zeilen 1 und 15? (1 Punkt)	<i>Integer</i>
b) Ergänzen Sie in der Zeile 18 den Platzhalter ② (1 Punkt)	<i>int</i> ✓
c) Was muss in den Zeilen 31-33 beim Platzhalter ③ ergänzt werden, damit keine Compiler Warnungen auftreten? (2 Punkte)	<i>< Integer ></i>
d) Ergänzen Sie in den Zeilen 36-38 den Platzhalter ④ (2 Punkte)	<i>Future<Integer></i>
e) Ersetzen Sie die ??? in Zeile 45 mit passendem Text (2 Punkte)	<i>Siegerzeit</i>
f) In welcher Methode liegt die Ursache, wenn der Wettkampf abgebrochen wird (<code>ExecutionException</code> in Zeile 46/47)? (1 Punkt)	<i>call()</i>

```

1 public final class AthleteTask implements Callable<①> {
2
3     private static final Logger LOG =
4         LoggerFactory.getLogger(AthleteTask.class);
5     private final String name;
6     private final CountDownLatch startSignal;
7
8     public AthleteTask(final String name,
9                        final CountDownLatch startSignal) {
10        this.name = name;
11        this.startSignal = startSignal;
12    }
13
14    @Override
15    public ① call() throws Exception {
16        LOG.info("{} ist bereit...", name);
17        startSignal.await();
18        ② time = 9700 + ThreadLocalRandom.current().nextInt(300);
19        TimeUnit.MILLISECONDS.sleep(time);
20        LOG.info("{} ist in {} Sec im Ziel ", name, time / 1000f);
21        return time;
22    }
23 }
```

Modulendprüfung AD im HS 2024 – Teil 2: Nebenläufigkeit

```
24 public final class SprintRace {  
25  
26     private static final Logger LOG =  
27         LoggerFactory.getLogger(SprintRace.class);  
28  
29     public static void main(final String[] args) {  
30         final CountDownLatch startSignal = new AtomicCountDownLatch(1);  
31         final Callable<Long> a1 = new AthleteTask("Carl Lewis", startSignal);  
32         final Callable<Long> a2 = new AthleteTask("Noah Lyles", startSignal);  
33         final Callable<Long> a3 = new AthleteTask("Usain Bolt", startSignal);  
34         try (final ExecutorService executor =  
35             Executors.newCachedThreadPool()) {  
36             final Future<Long> resultA1 = executor.submit(a1);  
37             final Future<Long> resultA2 = executor.submit(a2);  
38             final Future<Long> resultA3 = executor.submit(a3);  
39             LOG.info("Los!");  
40             startSignal.countDown();  
41             int values[] =  
42                 {resultA1.get(), resultA2.get(), resultA3.get()};  
43             Arrays.sort(values);  
44             int result = values[0];  
45             LOG.info("??? = {}", result / 1000f);  
46         } catch (ExecutionException | InterruptedException ex) {  
47             LOG.info("Wettkampf abgebrochen!");  
48         }  
49     }
```

Modulendprüfung AD im HS 2024 – Teil 2: Nebenläufigkeit

basierend auf "Divide & Conquer", hilft Aufgaben auf mehrere Threads zu verteilen. Effizienter als ein normaler ThreadPool.

Aufgabe 7: Fork-Join (4 Punkte) Fork: Aufgabe teilt sich rekursiv in kleinere Teilaufgaben auf.

Eine Klasse RaceTask soll mit dem Fork-Join Frameworks ein Rennen simulieren. Kreuzen Sie die richtigen Aussagen an. Es können auch mehrere Aussagen korrekt sein. Es gibt keine negativen Punkte. Aber nur für vollständig korrekte Antworten gibt es Punkte.

- a) Von welcher Klasse des Fork-Join Frameworks leiten Sie die Klasse RaceTask ab, wenn sie eine Zeit (Integer) zurückgeben muss? (1 Punkt)

RecursiveAction : Für Aufgaben ohne Rückgabewert	<input type="checkbox"/>
RecursiveTask : Für Aufgaben mit Rückgabewert	<input checked="" type="checkbox"/>
CountedCompleter : Für spezielle Synchronisationsfälle	<input type="checkbox"/>

✓ / 1

- b) In der Klasse RaceTask ist eine Methode zu implementieren, welche die gelaufene Zeit zurückgibt. Wie sieht der Methodenkopf aus? (1 Punkt)

protected Integer call()	<input type="checkbox"/>
protected Integer invoke() → wird von ForkJoinPool selbst aufgerufen, nicht von mir	<input type="checkbox"/>
protected Integer compute() → gehört zu ForkJoin!	<input checked="" type="checkbox"/>

—

- c) Um ein Rennen zu starten, muss der RaceTask gestartet werden. Dazu benötigt man den Threadpool des Fork-Join Frameworks:

```
final ForkJoinPool pool = new ForkJoinPool();
final RaceTask task = new RaceTask(...);
```

Mit welchen Methoden des Threadpools starten Sie den Task, um die kürzest gelaufene Zeit zu erhalten? (1 Punkt)

pool.invoke(task); gibt das Ergebnis direkt zurück	<input checked="" type="checkbox"/>
pool.execute(task); feuert nur ab, liefert kein Ergebnis	<input type="checkbox"/>
pool.submit(task); liefert ein ForkJoinTask, aus dem man das Ergebnis holen kann.	<input checked="" type="checkbox"/> ✓

—

- d) Welche Methode aus c) liefert direkt die schnellste gelaufene Zeit zurück? (1 Punkt)

pool.execute(task);	<input type="checkbox"/>
pool.submit(task);	<input type="checkbox"/>
pool.invoke(task);	<input checked="" type="checkbox"/>

—

Modulendprüfung AD - HS 2024**Teil 1 von 3 – Datenstrukturen**

Ingmar Baetge – Version 1.0

Name: Degtyareva
(Bitte mit Druckbuchstaben schreiben)Vorname: DominikaUnterschrift: 
20.01.25

Rahmenbedingungen:

1. Prüfungszeit: total 180 Minuten, für diesen Teil ca. **50** Minuten.
2. Diese Prüfung besteht aus drei Teilen und es können 160 Punkte erreicht werden, davon in diesem Prüfungsteil **50** Punkte. Jeder (Teil-)Aufgabe ist eine maximal erreichbare Punktzahl zugeordnet.
3. Schreiben Sie Ihren Namen und Vornamen mit Druckbuchstaben oben auf dieses Blatt. Mit der Unterschrift bezeugen Sie, dass Sie diesen Prüfungsteil persönlich und nur mit erlaubten Hilfsmitteln bearbeitet haben. Blätter ohne diese Angaben werden nicht bewertet.
4. Es handelt sich um eine schriftliche Prüfung ohne Einsatz des Computers oder elektronischer Hilfsmittel. Sie dürfen keine Unterlagen verwenden.
5. Sollte eine Aufgabenstellung Unklarheiten aufweisen, treffen und dokumentieren Sie Annahmen.
6. Schreiben Sie verständlich und gut leserlich. Missverständliche Lösungen werden nicht berücksichtigt.
7. Benutzen Sie den Freiraum unter den Aufgaben für Ihre Lösung.

Gutes Gelingen & viel Erfolg!

Für die Korrektur (nicht ausfüllen!)

1	2	3	4	5	6				Punkte	Visum
6,5	2	0,5	0	0	5				14	

Aufgabe 1: Semantiken von Datenstrukturen (8 Punkte)

Sie kennen verschiedene Datenstrukturen mit ihren typischen Methoden. Ergänzen Sie in der Tabelle die fehlenden Bezeichnungen und Methodenköpfe in Java Syntax **inklusive der generischen Typinformation** und dem Rückgabetyp. (8 Punkte)

Bezeichnung Semantik inkl. Generics	Methode zum Einfügen Methodenkopf mit Generics	Methode zum Entnehmen Methodenkopf mit Generics
Queue<E>	boolean offer(E e)	E poll()
List<E>	boolean add(E e)	E remove()
Stack<E>	E push(E e)	E pop()
Map<K,V>	boolean put(K k, V v)	Remove (K u)

Map/Su?

Set : add(), remove(), get(), size()

List : add(), remove(), get(), size()

Map : put(), remove(), get(), size()

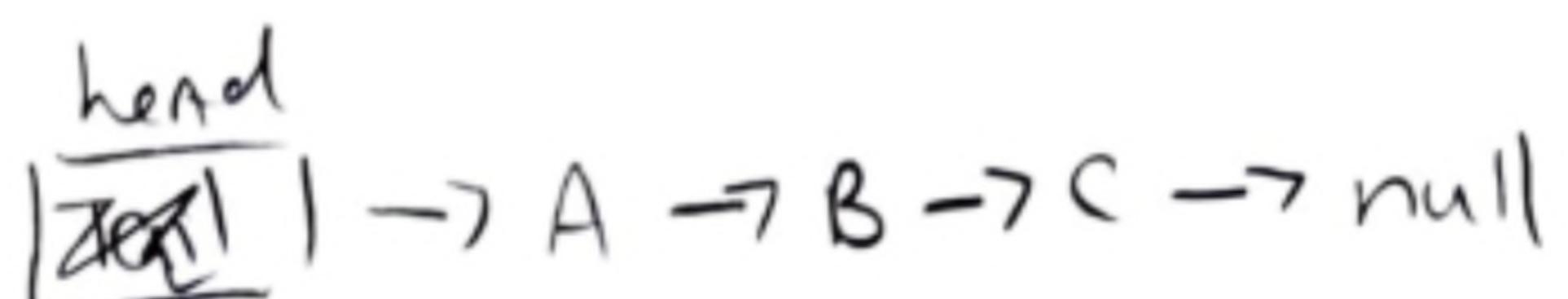
Queue: offer(), poll(), peek(); isEmpty()

Stack : push(), pop(), peek(), isEmpty()

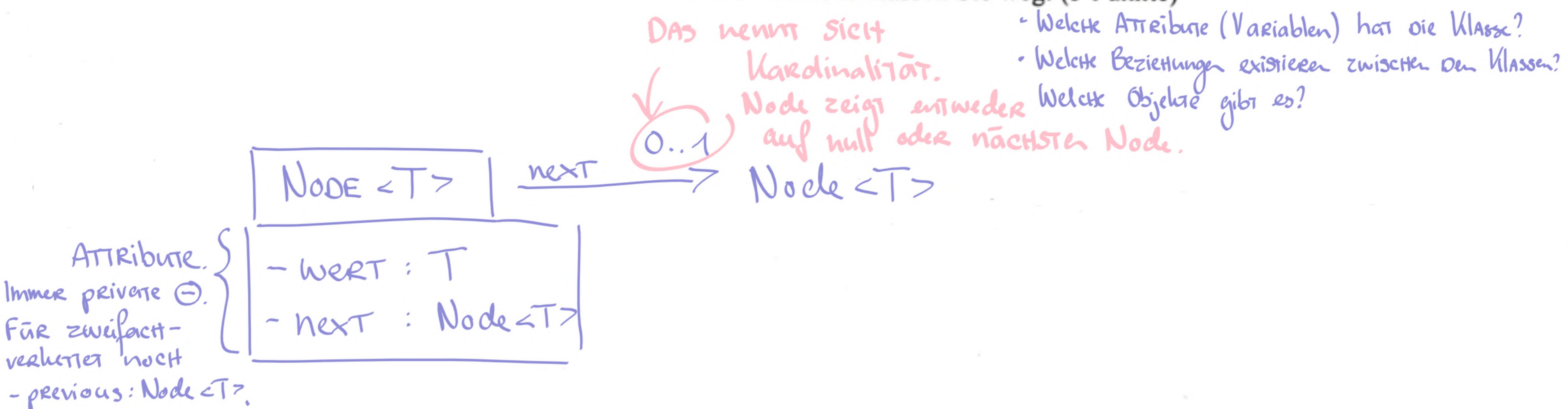
Klassenname
- attribut1 : Typ
- attribut2 : Typ

⊖ : private
⊕ : public

Aufgabe 2: Einfach verkettete Liste (8 Punkte)

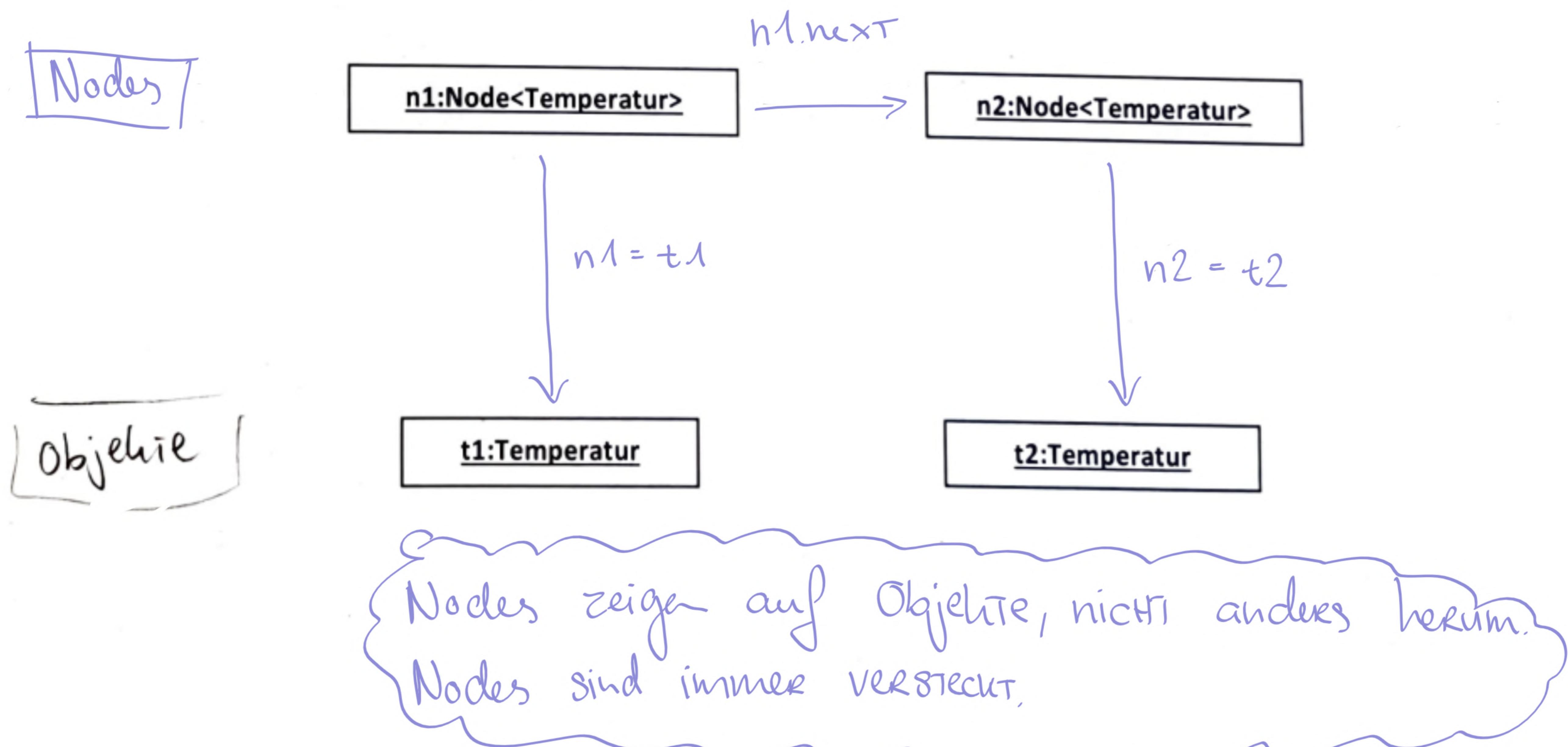


- a) Skizzieren Sie ein UML-Klassendiagramm nur für die Klasse **Node** einer **einfach verketteten Liste** mit dem generischen Typ **T**. Verlangt sind alle Attribute inklusiv exakter Typen und die Beziehungen (als Pfeile), inklusive aller **Kardinalitäten**. Die Methoden lassen Sie weg. (5 Punkte)



Pfeiltyp	Bedeutung	Wann verwenden?
Voller Strich (→)	Assoziation	Wenn eine Klasse ein Attribut hat, das eine Referenz auf eine andere Klasse ist
Gestrichelter Pfeil (↔)	Abhängigkeit (Dependency)	Wenn eine Methode eine andere Klasse benutzt (z.B. als Parameter, Rückgabewert)

- b) Zwei Temperatur-Objekte (**t1, t2**) sind mit Hilfe von zwei Node-Objekten (**n1, n2**) in einer **einfach verketteten Liste** nach dem Modell von a) abgelegt. Ergänzen Sie im folgenden UML-Objektdiagramm sämtliche existierenden **Beziehungen**, welche Sie mit den **Namen der Attribute** aus Aufgabe a) beschriften. (3 Punkte)



Aufgabe 3: Analyse einer Datenstruktur (9 Punkte)

Gegeben ist die folgende Datenstruktur.

```
01 public class ArrayDataStructure {  
02  
03     private final char[] data;  
04     private int head;  
05     private int tail;  
06     private int size;  
07  
08     public ArrayDataStructure(final int size) {  
09         this.data = new char[size];  
10         this.head = this.data.length;  
11         this.tail = 0;  
12         this.size = 0;  
13     }  
14     3)           ↗ Einfügen-Methode einer Queue  
15     2)           ↗ public void offer(final char item) {  
16         this.head++;  
17         this.data[this.head] = item;  
18         this.size++;  
19     }  
20     3)           ↗ Entnehmen-Methode einer Queue  
21     2)           ↗ public char poll() {  
22         final char value = this.data[this.tail];  
23         this.data[this.tail] = ' ';  
24         this.tail++;  
25         this.size--;  
26         return value;  
27     }  
28  
29     public boolean isEmpty() {  
30         return this.size == 0;  
31     }  
32  
33     public boolean isFull() {  
34         return this.size == this.data.length;  
35     }  
36             ↗ gibt es nur mit Queue. Stack wäre isEmpty.  
37     public int size() {  
38         return this.size;  
39     }  
40 }
```

Sie dürfen voraussetzen, dass der Quellcode fehlerfrei kompilierbar ist (**keine** Syntaxfehler).

Queue - FIFO

Modulendprüfung AD im HS 2024 – Teil 1: Datenstrukturen

ARRay

ist damit die Queue gemeint?

- a) Welche Datenstruktur wurde hier (mit semantischen Fehlern!) und in welcher Technik implementiert?
(1 Punkt)

ArrayQueue → Ringbuffer

Nein, nicht jede `ArrayQueue` ist automatisch ein Ringpuffer.
Aber: Eine effiziente `ArrayQueue` sollte einer sein.

- b) Welche **vier** semantischen Fehler sind enthalten? Beschreiben Sie in Stichworten und geben Sie jeweils die betroffene Methode an! (4 Punkte)

1) Falscher Startwert für head: head darf nicht am Schluss vom Array starten.
Korrekt: `THIS.head = 0;`

2) Noch kein Ringverhalten: Wenn das Array voll ist, kommt es zum `IndexOutOfBoundsException`.
Korrekt:

```
this.head = (this.head + 1) % this.data.length;  
this.tail = (this.tail + 1) % this.data.length;
```

3) Buffer voll, Buffer leer?: `Offer()` fügt einfach ein, ohne zu prüfen, ob Buffer voll ist.
`Poll()` liest einfach, ohne zu überprüfen, ob leer ist.
Korrekt: Vor `offer()` prüfen mit `isFull()` & vor `poll()` mit `isEmpty()`

4) ???

- c) Wählen Sie **einen** der vier Fehler von b) aus und erläutern Sie nachvollziehbar die Korrektur. (2 Punkte)

- d) Könnte man in dieser Datenstruktur auch auf die Speicherung des Attributs `size` verzichten?
Begründen Sie! (2 Punkte)

Ja, man kann auf `size` verzichten.

Die Anzahl der gespeicherten Elemente kann zur Laufzeit berechnet werden mit:

$$(\text{head} - \text{tail} + \text{data.length}) \% \text{data.length}$$

Aufgabe 4: Auf Array basierender int-Stack (10 Punkte)

FIL
↓

Gegeben ist die folgende Implementation eines auf Array basierenden int-Stacks:

```
01 public final class ArrayIntStack {  
02  
03     private final int[] array;  
04     private int top;  
05  
06     public ArrayIntStack(final int capacity) {  
07         this.array = new int[capacity];  
08         → this.top = this.array.length; // Zeigt immer auf den letzten Platz  
09     }  
10  
11     public void push(final int item) throws StackException {  
12         if (this.isFull()) {  
13             throw new StackException("Stack is full.");  
14         }  
15         this.array[--this.top] = item;  
16     }  
17  
18     public int pop() throws StackException {  
19         if (this.isEmpty()) {  
20             throw new StackException("Stack is empty.");  
21         }  
22         return this.array[this.top++];  
23     }  
24     ⚠ Es werden keine Werte gelöscht!!  
25     public boolean isFull() {  
26         return this.top <= 0;  
27     }  
28  
29     public boolean isEmpty() {  
30         return this.top > this.array.length - 1;  
31     }  
32 }
```

Sie dürfen voraussetzen, dass die verwendete StackException existiert, und der Quellcode fehlerfrei kompilierbar ist (**keine** Syntaxfehler).

- a) Welchen Inhalt haben die Attribute **array[]** und **top** des **stack**-Objektes **nach** der Ausführung des folgenden Codes? Tragen Sie die Werte unten in die Felder ein! (3 Punkte)

```
ArrayIntStack stack = new ArrayIntStack(4); // Array wird mit 4 Plätzen reserviert.
stack.push(5); [-,-,-,5] top 3
stack.push(2); [-,-,2,5] top 2
stack.push(0); [-,0,2,5] top 1
stack.push(7); [7,0,2,5] top 0
stack.pop();          top 1
stack.push(2); [2,0,2,5] top 2
stack.pop();          top 1
```

array[0]	array[1]	array[2]	array[3]
2	0	2	5

top
1

- b) Der Stack enthält eine semantische Unschönheit, welche sich bei einem Klassentyp (statt wie hier **int**) sogar auf die Speicherverwaltung auswirken würde. Erklären Sie! (2 Punkte)

Wenn das Array nicht aus primitiven DATENTYPEN (**int[]**) sondern aus z.B. **Integer[]** besteht, werden beim Kopieren des Arrays nur die Referenzen auf die Objekte kopiert, nicht aber die Objekte selbst.

- c) Korrigieren Sie den Fehler von b)! Geben Sie die **Zeile** und den **vollständigen Code** an, mit welcher Sie diese ersetzen. (5 Punkte)

Zeile: 10

Code: THIS.array = array.clone();

Aufgabe 5: Hashbasierte Datenstrukturen (6 Punkte)

- a) Wie berechnen Sie mit Java von einem Datenelement **data** (beliebiger Klassentyp) den Index für eine hashbasierte Datenstruktur mit einer Anzahl von **size** Plätzen? (2 Punkte)

int index = Math.abs(data.hashCode()) % size;

verhindert negative Hashwerte

liefert Hashwert

bringt den Wert in einen gültigen Bereich.

- b) Was sind Nachteile der Kollisionsbehandlung durch Sondieren? (2 Punkte)

Die Performance verschlechtert sich mit Füllgrad. Werte sind schwer auffindbar, da nicht am Hash-Index stehen. Das Löschen hinterlässt Lücken.

- c) Gegeben ist eine Hashtabelle mit 10 Plätzen und Bucketlists für die Kollisionsbehandlung. Es werden die folgenden Datenwerte eingetragen (mit aus Hashwert berechnetem Index):

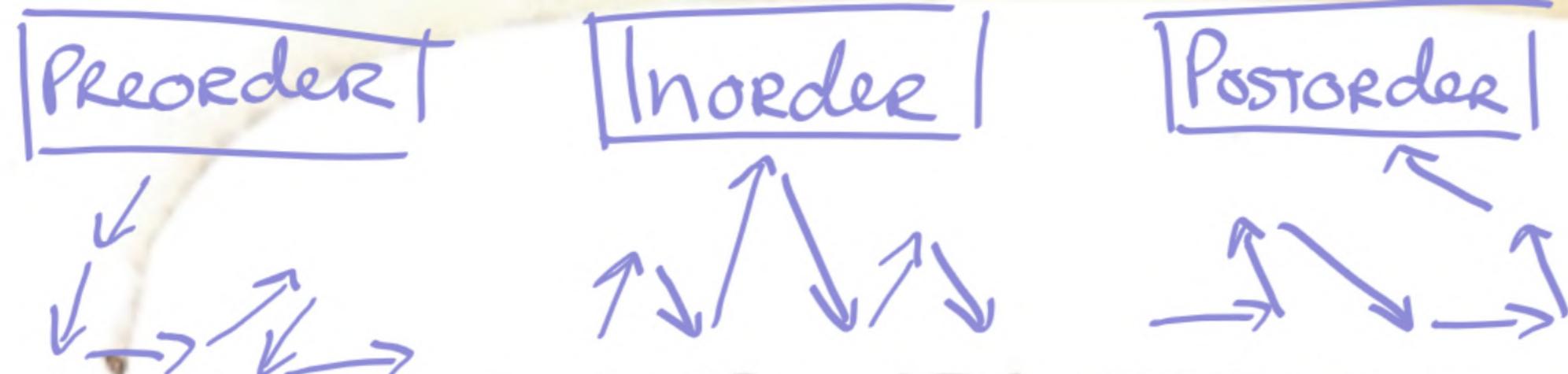
- A – Index 0
- B – Index 1
- D – Index 3
- H – Index 3
- L – Index 3

heißt: mehrere Elemente an einem Index möglich. Gegen teil zur Sondierung.

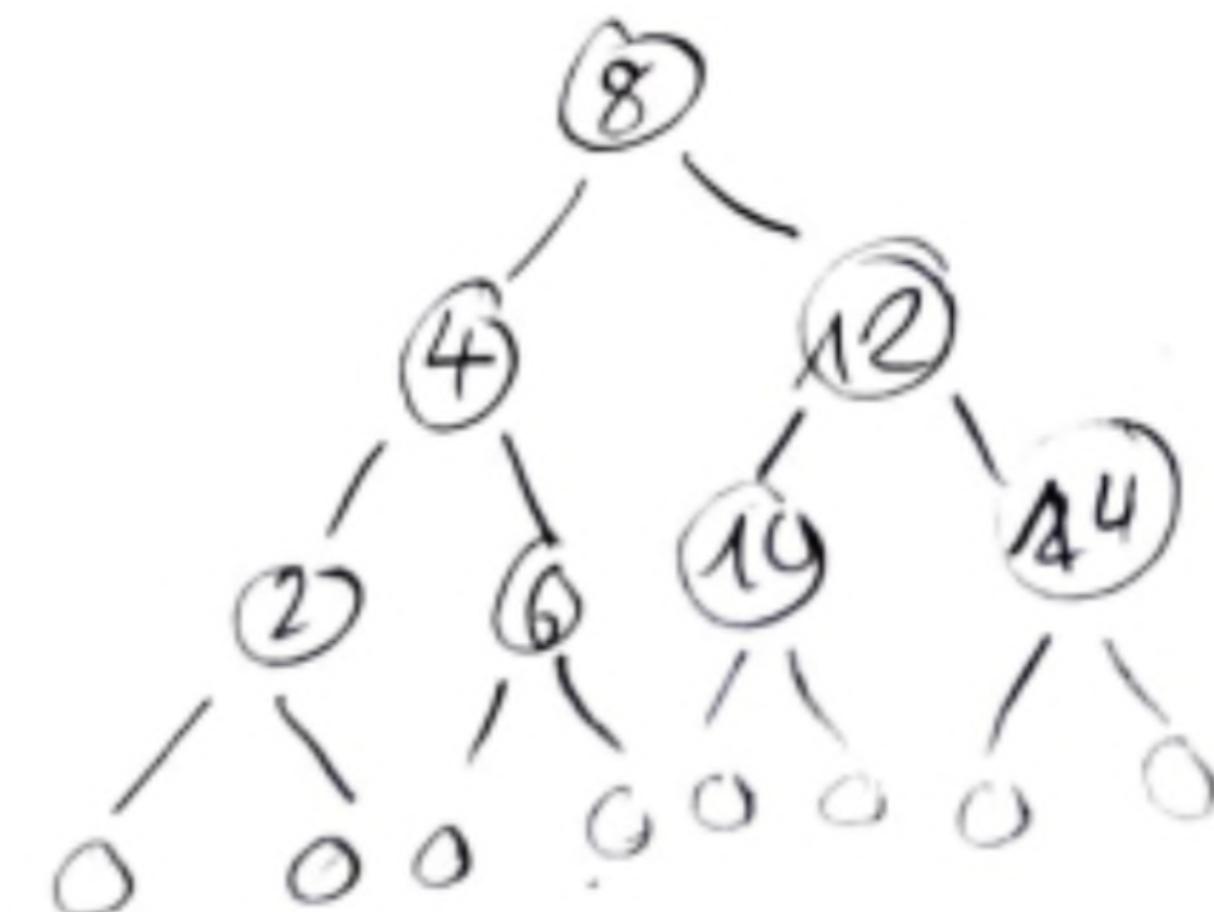
Ergänzen Sie unten grafisch den Zustand der Datenstruktur **nach dem Befüllen!** (2 Punkte)

0	1	2	3	4	5	6	7	8	9
A	B		D						

↓
H
↓
L



Modulendprüfung AD im HS 2024 – Teil 1: Datenstrukturen



Aufgabe 6: Bäume (9 Punkte)

- a) Beurteilen Sie die folgenden Aussagen zu Baum-Datenstrukturen. (2 Punkte)

Hinweis: Ein richtiges Kreuz ergibt 0.5 Punkte, ein falsches -0.5 Punkte. Sie erhalten bei dieser Aufgabe insgesamt aber *keine* negativen Punkte.

Richtig Falsch

Gibt man die Schlüsselemente eines Suchbaums mit einer Postorder-Traversierung aus, dann erhält man die Schlüssel in sortierter Reihenfolge. → Nur bei Inorder

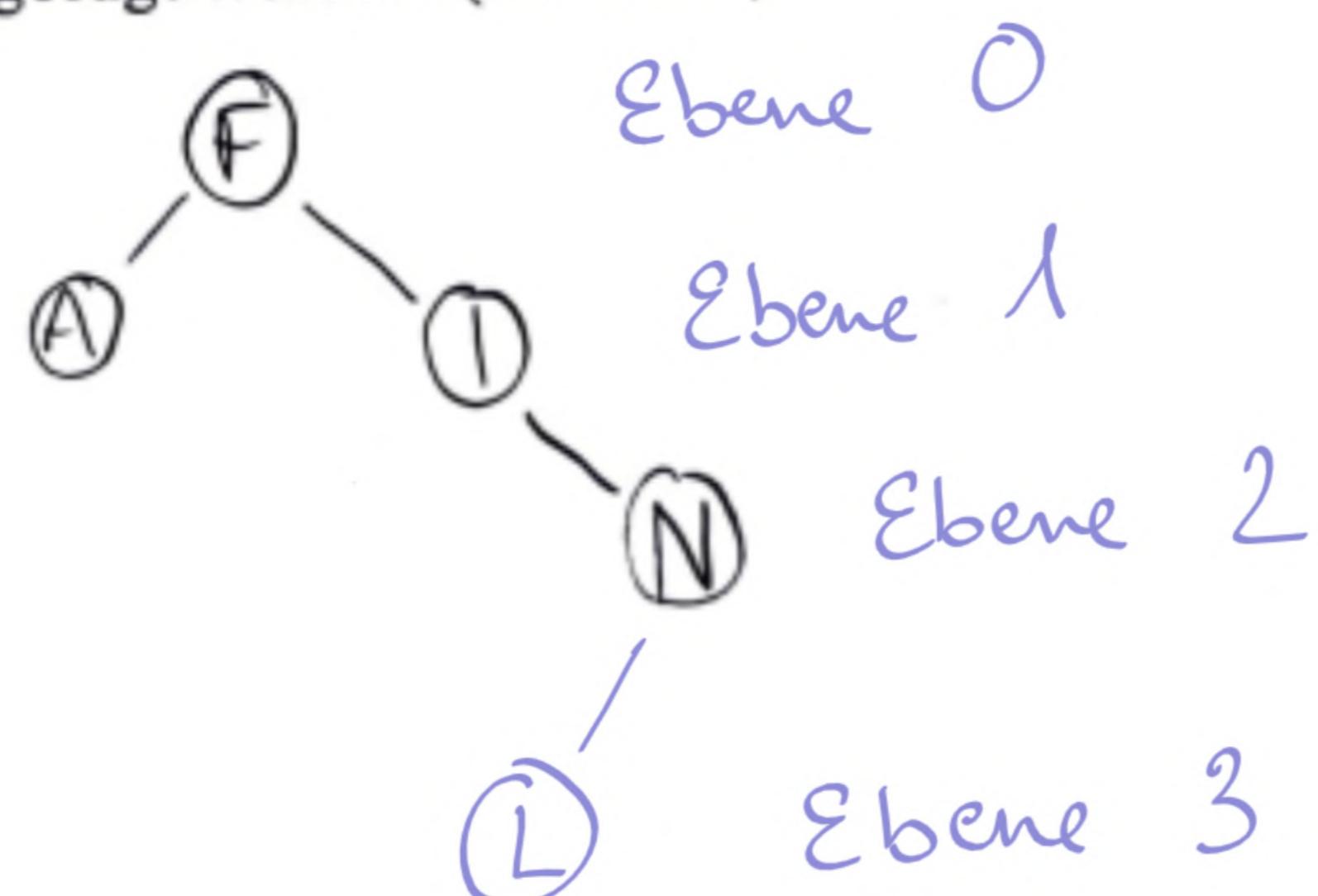
In einem höhenbalancierten Baum gilt in jedem Knoten: Die Höhendifferenz des linken und rechten Teilbaums ist maximal 1.

In einem beliebigen Suchbaum kann man Schlüsselemente mit dem Aufwand $O(\log n)$ finden. Nur bei balancierten Bäumen. Sonst $O(n)$

Ein vollständiger Binärbaum der Höhe 4 hat 31 Knoten.

bei Höhe 5

- b) Skizzieren Sie den resultierenden **binären Suchbaum**, wenn die Buchstaben 'F', 'I', 'N', 'A', 'L' in der vorgegebenen Reihenfolge eingefügt werden. (2 Punkte) A B C D E F G H I J K L M N O P



- c) Geben Sie die Preorder-Reihenfolge der Elemente des Baums aus b) an (1 Punkt)

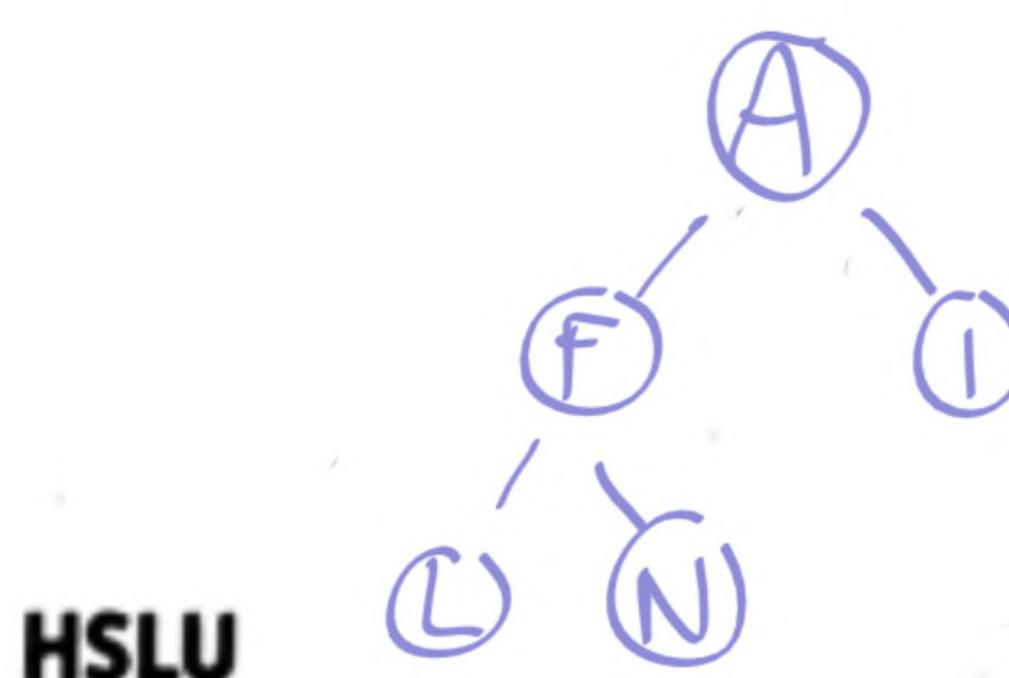
F, A, I, N, L

- d) Auf welcher Ebene befindet sich das Element 'L'. im obigen Baum? (1 Punkt)

Dritte Ebene

- e) In welcher Reihenfolge müssten die Buchstaben 'A', 'F', 'I', 'L', 'N' in einen **binären Suchbaum** eingefügt werden, so dass dieser nach **jedem Einfügeschritt linksbündig** voll ist? (3 Punkte)

A, F, I, L, N



---- Ende dieses Prüfungsteils ----