Karlsruher Institut für Technologie

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

Nachklausur Algorithmen I
WS 2020/2021

16. März 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 26 Seiten (11 Blätter) mit 9 Aufgaben.

Aufgabe	1	2	3	4	5	6	7	8	9	**Gesamt**
Erreichte Punkte										
Erreichbare Punkte	16	17	25	18	23	28	56	18	39	**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 (16 Punkte)

a) Es sei $g(n) \in \mathcal{O}(n)$ und $f(n) \in \Theta(n)$.

i) Gibt es Funktionen g und f mit $\forall n \in \mathbb{N}_+ : g(n) = f(n)$? Falls ja, geben Sie ein Beispiel an, ansonsten begründen Sie Ihre Antwort! **(3 Punkte)**

ii) Gibt es Funktionen g und f mit $\forall n \in \mathbb{N}_+ : g(n) > f(n)$? Falls ja, geben Sie ein Beispiel an, ansonsten begründen Sie Ihre Antwort! **(4 Punkte)**

b) Zeigen Sie, dass $\forall X \in \mathbb{R}_+ : \mathcal{O}(\log(n + X)) = \mathcal{O}(\log n)$! **(9 Punkte)**

Aufgabe 2: Listen und Hashing (17 Punkte)

a) Gegeben seien folgende Datenstrukturen:

1. ein unbeschränktes Feld
2. eine einfach verkettete Liste
3. eine doppelt verkettete Liste
4. eine Hashtabelle

Wählen Sie für die folgenden Szenarien jeweils die geeignetste dieser Datenstrukturen zur laufzeit- und speichereffizienten Umsetzung aus! Beschreiben Sie eventuell notwendige Anpassungen und begründen Sie kurz Ihre Wahl!

i) Szenario 1: Sie sollen eine First-In-First-Out-Warteschlange (FIFO Queue) implementieren. **(5 Punkte)**

ii) Szenario 2: Sie sollen eine große dünn-besetzte Matrix speichern, d.h. die meisten Einträge der Matrix sind null. Die Dimension ändert sich nicht, jedoch sollen Sie beliebige Einträge der Matrix ändern und auslesen können. **(5 Punkte)**

Matrikelnummer: _____

b) Im Folgenden soll Hashing mit linearer Suche angewandt werden. Die Hashfunktion und Ausgangszustand der Hashtabelle sind gegeben als:

Hashfunktion

Element	a	b	c	d	e	u	x	y	z
Hash-Wert	0	3	2	7	3	2	4	1	6

Hashtabelle

Index	0	1	2	3	4	5	6	7
Wert			c	b			z	

Führen Sie nun nacheinander die folgenden Operationen durch und geben Sie den Zustand der Hashtabelle nach Ausführen jeder Operationen an! **(7 Punkte)**

Hashtabelle

Index	0	1	2	3	4	5	6	7
`insert(a)`								
`insert(e)`								
`insert(u)`								
`remove(e)`								
`remove(c)`								

Weiterer Platz, falls Sie Ihre Antwort korrigieren wollen. Falls Sie diesen nutzen, kennzeichnen Sie eindeutig, welche Lösung gewertet werden soll:

Hashtabelle

Index	0	1	2	3	4	5	6	7
`insert(a)`								
`insert(e)`								
`insert(u)`								
`remove(e)`								
`remove(c)`								

Aufgabe 3: Sortieren (25 Punkte)

a) Nennen Sie einen Vor- und einen Nachteil von Radixsort gegenüber vergleichsbasierten Sortierverfahren! **(4 Punkte)**

Vorteil Radixsort:

Nachteil Radixsort:

b) Gegeben sei die 0-indizierte Folge $\langle 1, 2, 3, 4, 5 \rangle$. Geben Sie eine Permutation dieser Folge an, bei der der 2-Wege-Quicksort die meisten Vergleichsoperationen zur Sortierung benötigt! Nehmen Sie hierbei an, dass die Partitionierung von Quicksort stabil ist! Wählen Sie als Pivot immer:

i) das erste Element der betrachteten Teilfolge! **(2 Punkte)**

ii) das Element mit Index $\lfloor n/2 \rfloor$, wobei n die Länge der 0-indizierten Teilfolge ist! **(5 Punkte)**

Matrikelnummer: _____

c) Implementieren Sie die Funktion `iter_mix_sort` in Pseudocode! Die Funktion erhält ein 0-indiziertes Array `a` der Länge $n = 2^k$ ($k \in \mathbb{N}$) und soll es in-place sortieren.

Führen Sie hierzu ein Mischen der Teilfolgen wie bei Mergesort durch, aber ohne Rekursion zu verwenden! Der zusätzliche Speicherbedarf muss in $\mathcal{O}(1)$ sein und die Laufzeit in $\mathcal{O}(n \log n)$.

Sie können hierzu die Funktion `mix(a, off0, off1, off2)` verwenden. Unter der Annahme, dass die Bereiche $[\texttt{off}_0, \texttt{off}_1 - 1]$ und $[\texttt{off}_1, \texttt{off}_2 - 1]$ in `a` jeweils sortiert sind, steht nach Ausführung von `mix` die gemischte Folge im Bereich $[\texttt{off}_0, \texttt{off}_2 - 1]$ von `a`. Die Funktion `mix` benötigt $\mathcal{O}(1)$ Speicher und $\mathcal{O}(\texttt{off}_2 - \texttt{off}_0)$ Laufzeit. **(14 Punkte)**

```
Function mix(a: [T; n], off_0: int, off_1: int, off_2: int);
Function iter_mix_sort(a: [T; n]) {




}
```

Matrikelnummer: _____

Aufgabe 4: Prioritätslisten (18 Punkte)

a) Nennen Sie zwei Algorithmen aus der Vorlesung, die mit Hilfe von Prioritätslisten implementiert werden können! **(2 Punkte)**

b) In dieser Aufgabe sollen n Job-IDs mit einer Prioritätsliste verwaltet werden. Jeder Job-ID ist eine Priorität $p \in \mathbb{N}$ zugeordnet. Es ist ein $k \in \mathbb{N}$ bekannt mit $\forall p : 0 \leq p \leq k$. Sie sollen eine Prioritätsliste mit Speicherbedarf $\mathcal{O}(k + n)$ entwerfen, die folgende Operationen unterstützt:

- `insert(pq, prio, id)` fügt eine Job-ID `id` mit Priorität `prio` zur Prioritätsliste `pq` hinzu in Laufzeit $\mathcal{O}(1)$.
- `deleteMin()` gibt eine Job-ID mit minimaler Priorität zurück und löscht dieses aus `pq` in Laufzeit $\mathcal{O}(k)$.

i) Geben Sie die Variable(n) der Prioritätsliste in Pseudocode an! **(6 Punkte)**

```
Struct PriorityQueue {


}
```

ii) Implementieren Sie die Operation `insert` in Pseudocode! **(4 Punkte)**

```
Function insert(pq: PriorityQueue, prio: int, id: int) {




}
```

iii) Implementieren Sie die Operation `deleteMin` in Pseudocode! Sie dürfen davon ausgehen, dass die Prioritätsliste nicht leer ist. **(6 Punkte)**

```
Function deleteMin(pq: PriorityQueue) : int {




}
```

Matrikelnummer: _____

Aufgabe 5: Graphen (23 Punkte)

a) Nennen Sie je einen Vorteil und einen Nachteil von Adjazenzlisten gegenüber Adjazenzfeldern! **(4 Punkte)**

Vorteil Adjazenzlisten:

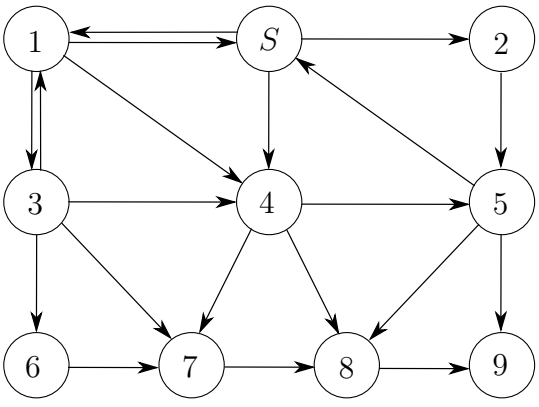
Nachteil Adjazenzlisten:

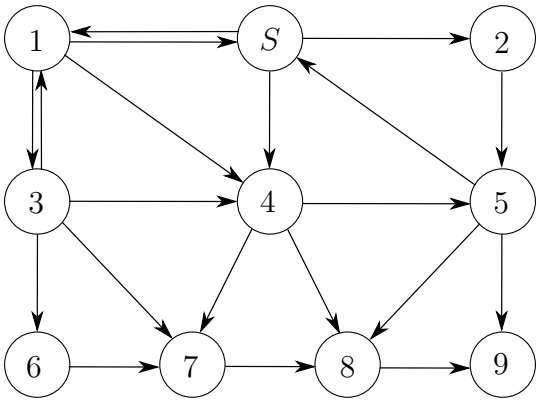
b) Führen Sie auf dem folgenden Graphen eine Breitensuche vom Startknoten S aus durch! Falls mehrere Knoten als Nächstes traversiert werden können, wählen Sie den Knoten mit geringstem Index!

- Markieren Sie alle Baumkanten der Breitensuche mit b und alle Rückwärtskanten mit r!
- Wie viele Vorwärtskanten ergeben sich?

Falls Sie die Antwort korrigieren müssen, können Sie den Graphen auf der rechten Seite nutzen. Kennzeichnen Sie eindeutig, welche Lösung gewertet werden soll! **(8 Punkte)**

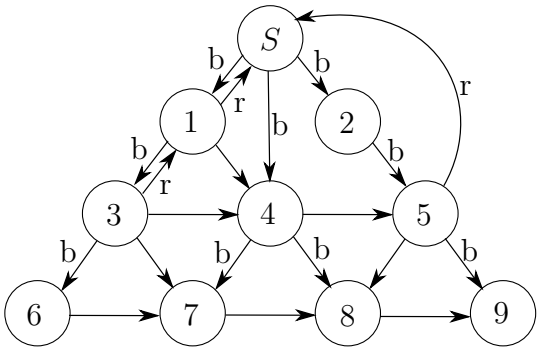
Anzahl Vorwärtskanten:





(falls Korrektur notwendig)

Solution:



Matrikelnummer: _____

c) Gegeben sei ein gerichteter, ungewichteter Graph $G = (V, E)$ mit $V = \{0, ..., N-1\}$. Die Funktion `getAdjacent` erhält den Graphen sowie einen seiner Knoten `b : int` als Eingabe.

Die Funktion `getAdjacent` soll alle Knoten a mit $(a, b) \in E$ mithilfe der Funktion `print(a: int)` ausgeben. Die Laufzeit der Funktion soll in $\mathcal{O}(|V|)$ und der zusätzliche Speicher soll in $\mathcal{O}(1)$ liegen.

- Nennen Sie eine aus der Vorlesung bekannte Graphrepräsentation, mit welcher `getAdjacent` in $\mathcal{O}(|V|)$ implementiert werden kann! Fügen Sie diese Datenstruktur in Pseudocode dem Kopf der Funktion hinzu! *Hinweis: Achten Sie darauf, die Struktur(en) im Kopf in der korrekten Größe zu deklarieren.* **(5 Punkte)**
- Implementieren Sie den Algorithmus in Pseudocode basierend auf Ihrer gewählten Datenstruktur! **(6 Punkte)**

Geeignete Datenstruktur: _____

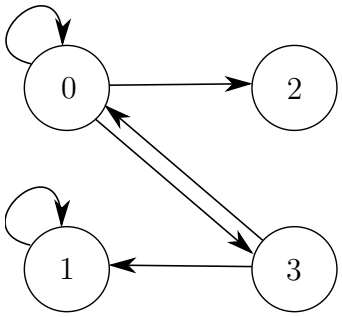
```
Function getAdjacent(b: int, _____) {


}
```

Matrikelnummer: _____

Aufgabe 6: Graphen und Adjazenzlisten (28 Punkte)

Gegeben sei ein gerichteter Graph $G = (V, E)$ mit $V = \{0, \ldots, N-1\}$. Gesucht ist der umgekehrte Graph $G' = (V, E')$ mit $E' = \{(b, a) \in V \times V \mid (a, b) \in E\}$.

a) Gegeben sei der abgebildete Graph $G_a = (V_a, E_a)$ mit $V_a = \{0, 1, 2, 3\}$. Zeichnen Sie daneben den umgekehrten Graphen G'_a! **(4 Punkte)**



b) Der Graph liegt als Adjazenzlistenrepräsentation vor. Die Listenelemente sind als `Node` implementiert.

```
Struct Node {
      prev : Node,
      next : Node,
      v : int
}
```

i) Implementieren Sie die Funktion `insertNode(head : Node, new : Node)` in Pseudocode, sodass diese in Laufzeit $\mathcal{O}(1)$ das Listenelement `new` an das *Ende* der doppelt verketteten Liste mit erstem Element `head` einfügt! **(6 Punkte)**

```
Function insertNode(head: Node, new: Node) {

}
```

ii) Die Funktion `reverseGraph` erhält in `graph` einen Graphen in Adjazenzlistenrepräsentation. Sie soll in Laufzeit $\mathcal{O}(|V| + |E|)$ den umgekehrten Graphen von `graph` in Adjazenzlistenrepräsentation zurückgeben.

- Implementieren Sie die Funktion `reverseGraph` in Pseudocode! Modifizieren Sie dafür `out`, sodass diese den umgekehrten Graphen von `graph` darstellt!
- Bestimmen und begründen Sie möglichst enge Laufzeitschranken für Ihren Algorithmus in $\mathcal{O}$-Notation!

Die Listen (auch leere Listen) starten mit einem Dummy-Header-Element, das mit `initDummyNode` erzeugt werden kann. Sie dürfen `insertNode` nutzen und davon ausgehen, dass `insertNode` konstante Laufzeit hat. **(18 Punkte)**

```
Function initDummyNode() : Node {
    n : Node
    n.v ← -1
    n.prev ← n
    n.next ← n
    return n
}
```

```
Function reverseGraph(graph: [Node; N]) : [Node; N] {
  // Initialisiere Adjazenzlistenrepräsentation out
  out : [Node; N]
  for v in [0, N-1] do
    out[v] ← initDummyNode()
  end

  // Füllen Sie hier out mit dem umgekehrten Graph:









  return out
}
```

Laufzeitanalyse:

Matrikelnummer: _____

Aufgabe 7: Kürzeste Wege (56 Punkte)

a) Geben Sie in $\mathcal{O}$-Notation möglichst enge Schranken für das Worst-Case-Laufzeitverhalten des Dijkstra- und Bellman-Ford-Algorithmus für Graphen $G = (V, E)$ mit positiven Kantengewichten an! Der Dijkstra-Algorithmus sei mit Binärheap-Prioritätslisten wie in der Vorlesung implementiert. **(4 Punkte)**

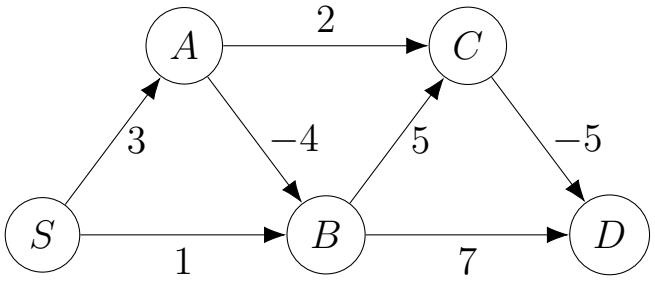
Dijkstra	Bellman-Ford

b) Was wird im Dijkstra-Algorithmus als Priorität in der Prioritätsliste abgelegt? **(2 Punkte)**

c) Welche besondere Funktion neben `insert` und `deleteMin` muss eine Datenstruktur für die Prioritätsliste im Dijkstra-Algorithmus zur Verfügung stellen? Wofür wird diese besondere Funktion im Dijkstra-Algorithmus verwendet? **(3 Punkte)**

d) Wie kann die Ausführung des Dijkstra-Algorithmus beschleunigt werden, wenn nur die Distanz vom Startknoten $s \in V$ zu einem bestimmten Knoten $v \in V$ statt zu allen Knoten gesucht wird? Begründen Sie, warum Ihre Lösung weiterhin ein korrektes Ergebnis liefert! **(6 Punkte)**

e) Führen Sie den Bellman-Ford-Algorithmus auf dem gegebenen gerichteten Graphen $G = (V, E)$ aus, um alle kürzesten Wege vom Startknoten S aus zu ermitteln! Geben Sie nach jedem Durchlauf der äußeren Schleife den Vorgänger *Pre* und die Distanz *Dist* eines jeden Knotens an! **(14 Punkte)**



Iteration	S		A		B		C		D	
	Pre	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*
0	-	0	-	∞	-	∞	-	∞	-	∞
1										
2										
3										
4										

Ersatztabelle zur Durchführung des Bellman-Ford-Algorithmus. Markieren Sie eindeutig, welche Lösung gewertet werden soll!

Iteration	S		A		B		C		D	
	Pre	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*
0	-	0	-	∞	-	∞	-	∞	-	∞
1										
2										
3										
4										

Matrikelnummer: _____

f) Sei $G = (V, E)$ ein gerichteter Graph mit Gewichtungsfunktion $w : E \to \mathbb{R}$.

Sei $C = (v_0, ..., v_{k-1}, v_k)$ mit $v_k = v_0$ und $k > 1$ ein negativer Kreis in G.

i) Geben Sie das Gewicht W_c von C an! **(3 Punkte)**

$W_c =$

ii) Sei $s \in V$, und seien alle Knoten $v \in V$ von s aus erreichbar. Für alle $v \in V$ sei $d(v)$ die Länge eines einfachen, kürzesten Pfades von s nach v, der durch $n - 1$ Relaxationen aller Kanten, also durch Ausführung des Bellman-Ford-Algorithmus, gefunden wird.

Zeigen Sie: $\exists$ Kante $(u, v) \in C : d(u) + w(u, v) < d(v)$

- Sie können den Beweis durch Widerspruch führen und W_c nutzen.
- Für $i > k$ beschreibt v_i den Knoten $v_{i \bmod k}$ mit Index $(i \bmod k)$ in C, also z.B. $v_{k+1} \equiv v_1$, bzw. (v_k, v_{k+1}) entspricht der Kante (v_0, v_1).

(24 Punkte)

Aufgabe 8: Minimale Spannbäume (MST) (18 Punkte)

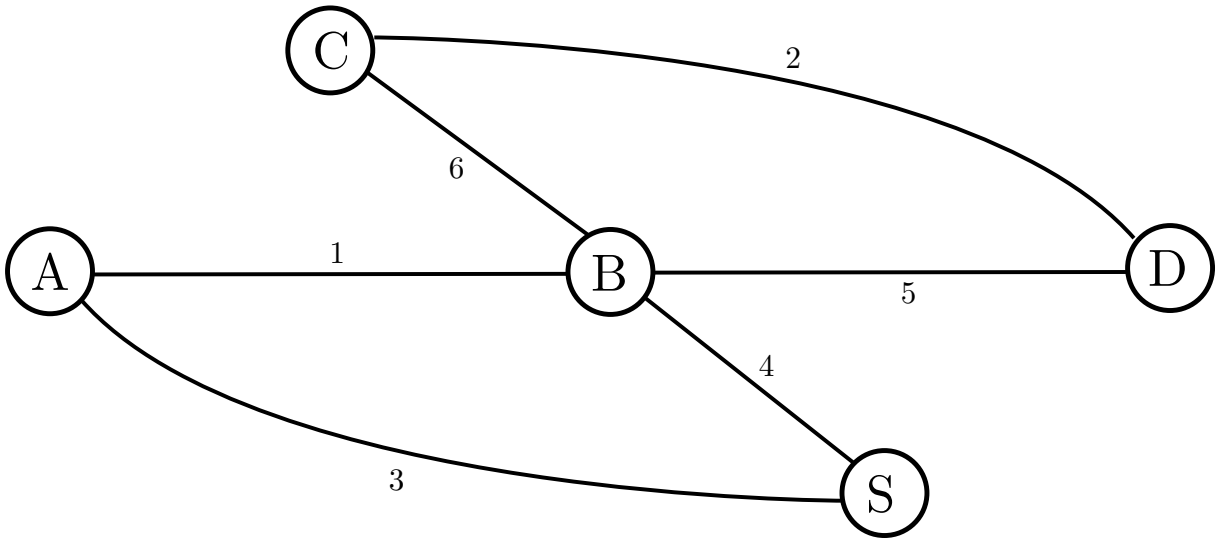
a) Nennen und beschreiben Sie kurz die Eigenschaft minimaler Spannbäume (MST), die dem Jarník-Prim-Algorithmus zu Grunde liegt! **(3 Punkte)**

b) Der in der Vorlesung vorgestellte Jarník-Prim-Algorithmus verwendet eine adressierbare Prioritätsliste. Welche Werte speichert dabei die Prioritätsliste mit welcher Priorität? **(4 Punkte)**

Werte:

Priorität:

c) Bestimmen Sie den minimalen Spannbaum (MST) des folgenden Graphen mit Hilfe des Kruskal-Algorithmus! Geben Sie in der Tabelle pro Schritt die betrachtete Kante an, sowie ob diese zum MST hinzugefügt wurde! **(6 Punkte)**



Matrikelnummer: _____

Schritt	1	2	3	4	5	6
Kante	{ , }	{ , }	{ , }	{ , }	{ , }	{ , }
Im MST (✓/X)						

d) Welche Aufgabe hat die Union-Find-Datenstruktur im Kruskal-Algorithmus? **(2 Punkte)** ☐

e) Wie beeinflusst Pfadkompression die Laufzeit der find-Operation auf Union-Find-Datenstrukturen (ohne „union by rank“) mit n Einträgen? **(3 Punkte)** ☐

Aufgabe 9: Suchbäume (39 Punkte)

a) Sei T ein Suchbaum, der die Elemente der Menge M speichert. Welches Element aus M gibt `T.locate(k)` zurück? **(4 Punkte)**

`T.locate(k)` =

b) Geben Sie einen binären Suchbaum mit minimaler Höhe für die Sequenz $\langle 4, 3, 6, 9, 7, \infty \rangle$ an! **(5 Punkte)**

Matrikelnummer: _____

c) Für einen (a, b)-Baum soll die Operation `select(k: int) : Element` aus n gespeicherten Elementen das Element mit Rang $k \in [1, \ldots, n]$ zurückgeben.

i) Wie kann diese Operation in einem (a, b)-Baum ohne Augmentierungen asymptotisch optimal implementiert werden? Geben Sie auch die Laufzeit an! **(4 Punkte)**

ii) Durch eine geeignete Augmentierung kann die asymptotische Laufzeit von `select` verbessert werden, ohne die asymptotische Laufzeit anderer Operationen zu verschlechtern. Erweitern Sie die Datenstrukuren in Pseudocode um dafür notwendige Variablen und beschreiben Sie, was diese speichern! **(6 Punkte)**

```
Struct ABTree{
   elements: List<Element>,
   root: ABHandle,
   height: int,




}

Struct ABHandle: ABItem or Element   // Siehe Hinweis iii)

a, b : int // Konstanten (a, b)

Struct ABItem{
   degree: int,
   splitters: [Key; b - 1],
   children: [ABHandle; b],




}
```

iii) Implementieren Sie die Operation `select` für den augmentierten (a, b)-Baum in Pseudocode und mit Laufzeit $\mathcal{O}(\log n)$!

Beachten Sie, dass `ABHandle` entweder `ABItem` oder `Element` darstellt. Sie können es entsprechend verwenden, wenn klar ist, welchen Typ ein `ABHandle` darstellt (siehe Codebeispiel).

Hinweis: Wie ergibt sich der Typ von `ABHandle` aus der Tiefe im Baum?

(20 Punkte)

Codebeispiel für `ABHandle`:

```
abItem : ABItem ← ...
childHandle: ABHandle ← abItem.children[0]
// Nur erlaubt, wenn childHandle kein Blatt ist:
grandChildHandle : ABHandle ← childHandle.children[0]
// Nur erlaubt, wenn dazu childHandle.children[0] Blatt ist:
grandChildElement: Element ← childHandle.children[0]
```

```
Function select(t: ABTree, k: int) : Element {
```

```
}
```