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 33 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]
```

Name: MUSTERLÖSUNG 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)**

Musterlösung

Ja, z.B. $g(n) = n$ und $f(n) = n$
Punkte:
- Volle Punkte für ja plus funktionierendes Beispiel
- Nur "Ja": Keine Punkte.
- Theoretische Begründung ohne Beispiel: -1P.
- Ja und korrektes beispiel, aber noch zusätzlicher Text mit Fehlern: Abzug nach ermessen -0 bis -3P (falls zu viel falsch leider 0P insgesamt)
- "Ja", aber Beispiel für g oder f gar nicht in O(n) bzw Theta(n): 0P.

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)**

Musterlösung

Ja, zB wenn $g(n) = 3n$ und $f(n) = 2n$
Punkte: (wie oben)
- Volle Punkte für ja plus funktionierendes Beispiel
- Nur "Ja": Keine Punkte.
- Theoretische Begründung ohne Beispiel: -1P.
- Ja und korrektes beispiel, aber noch zusätzlicher Text mit Fehlern: Abzug nach ermessen -0 bis -3P (falls zu viel falsch leider 0P insgesamt)
- "Ja", aber Beispiel für g oder f gar nicht in O(n) bzw Theta(n): 0P.

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

Musterlösung

Mögliche Lösungen:
1. Umformung im O kalkül drin

$$
\begin{aligned}
O(\log(n+X)) &= O\left(\log\left(n\ \frac{n+X}{n}\right)\right) && (1)\\
&= O\left(\log n + \log\frac{1+X/n}{1}\right) && (2)\\
&= O\left(\log n\right), && (3)
\end{aligned}
$$

da $\frac{1+X/n}{1} \to 1$ $(n \to \infty)$. Punkte:

- Erweiterung $n+X = n\frac{n+X}{n}$ oder $n+X = n(1+x/n)$: 3 Punkte
- Erweiterung log aufteilen: 3P
- das im limes gleichsetzen mit $\log n$: 3 Punkte (entweder explizit wie hier mit begründung, oder es muss irgendwo der limes mit $n \to \infty$ dabei stehen.)

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

2. Über teilmengen in beide Richtungen
a) $\mathcal{O}(\log n) \subseteq \mathcal{O}(\log(n+X))$
b) $\mathcal{O}(\log(n+X)) \subseteq \mathcal{O}(\log n)$
Punkte: 4 Punkte pro Richtung, 1P dafür dass man beides irgendwie angefangen hat. Pro richtung noch 1P für korrekte definition falls rest nur quatsch.

2.1 Limes-Betrachtung
a)

$$\lim_{n\to\infty} \frac{\log n + X}{\log n} = ... = 1 \tag{4}$$

b)

$$\lim_{n\to\infty} \frac{\log n + X}{\log n} = ... = 1 \tag{5}$$

funktioniert beides zB mit den selben umformungen wie oben. Satz von l'Hospital auch erlaubt.
Punkte:

- 1P für nur ansatz $\lim_{n\to\infty} \frac{\log n}{\log(n+X)}! < \infty$ bzw $! > 0$, falls klar ist für welche Teilmengenrichtung, 3P für Umformen / zeigen warum.
- Erweiterung rechenfehler $n + X = n(1 + x)$: -1P (macht berechnungen danach zumindest nicht einfacher)
- ërweiterungßähler und nenner e-hoch machen -3P, das darf man nicht
- kein konkreter wert ausgerechnet: -1P
- -1P falls teilrichtung unklar
- Falls für eine Richtung der tatsächliche Grenzwert angegeben ist, darf die andere Richtung auch in Prosa argumentiert werden. Es muss aber begründet werden warum das auch andersrum geht.
- $\lim \frac{\log n}{\log n+X} \approx \lim \frac{\log n}{\log n}$: 0P für Umformung.
- $\lim_{n\to a}$ für a irgendwas anderes als ∞: -1P für Bedingung und -1P für Umformung.

2.1.1 Satz von de l'hospital:

$$\lim_{n\to\infty} \frac{\log n}{\log n + x} = \frac{1/n}{1/n} = 1 \tag{6}$$

Punkte:

- Für die Umformung muss auch l'hosp oder so dabei stehen. Falls einfach so die logs durch brüche ersetzt werden -2P.
- 4P für eine Richtung, 5P für zweite Richtung (kann auch als text begründet werden)

Musterlösung

2.2 Teilmengen mit $\mathcal{O}$-Definition:

- $\mathcal{O}(\log n) \subseteq \mathcal{O}(\log(n+X))$:
  $\log n \leq \log(n+X)$
- $\mathcal{O}(\log(n+X)) \subseteq \mathcal{O}(\log n)$
  z.B. Sei $n_0 = max(2, X+1)$. Dann $\forall n \geq n_0$: $\log(n+X) \leq \log(n+n) = \log(2) + \log(n) \leq \log n + \log n = 2\log n = c \log n$ für $c = 2$.
  Punkte:
  - falls Umformung nur für $X \leq 1$ -1P. Pro Rechenfehler -1P
  - grober patzer in umformung: bis zu -3P für rechnung.
    z.B.
    $\log(n+x) = \log(n) \cdot \log(x)$ ist zu viel quatsch, 0 von 3 punkte für die umformung.
  - "für n groß genugßtatt konkretes n_0 anzugeben auch ok.

gemischt (zB eine Richtung mit limeskalkül, eine richtung mit definition)

Name: MUSTERLÖSUNG Matrikelnummer: _____

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)**

Musterlösung

Einfach verkettete Liste , da alle Operationen konstante Laufzeit haben und der Speicherverbrauch gering ist (weniger als doppelt verkettete Liste) .

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)**

Musterlösung

Hashtabelle ermöglicht kompakte Speicherung und erwartet konstanten Zugriff .

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)**

Musterlösung

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

Name: MUSTERLÖSUNG Matrikelnummer: _____

Aufgabe 3: Sortieren (25 Punkte)

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

Musterlösung

Vorteil	Nachteil
• asymptotische schneller in Abhängigkeit von n ($\mathcal{O}(kn)$) statt $\mathcal{O}(n \log n)$)	• mehr Annahmen (funktioniert nur direkt für Integer) • bei langen Schlüsseln oft langsamer • nicht robust gegen beliebige Eingabeverteilungen • Cache-Effizienz schwieriger

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)**

Musterlösung

Für die maximale Anzahl an Vergleichen muss als Pivot immer das größte oder kleinste Element der verbleibenden Teilfolge gewählt werden. Dadurch ergeben sich $2^4 = 16$ mögliche Lösungen:

$[1, 2, 3, 4, 5]$	$[1, 5, 2, 3, 4]$	$[5, 1, 2, 3, 4]$	$[5, 4, 1, 2, 3]$
$[1, 2, 3, 5, 4]$	$[1, 5, 2, 4, 3]$	$[5, 1, 2, 4, 3]$	$[5, 4, 1, 3, 2]$
$[1, 2, 5, 3, 4]$	$[1, 5, 4, 2, 3]$	$[5, 1, 4, 2, 3]$	$[5, 4, 3, 1, 2]$
$[1, 2, 5, 4, 3]$	$[1, 5, 4, 3, 2]$	$[5, 1, 4, 3, 2]$	$[5, 4, 3, 2, 1]$

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

Musterlösung

Im Prinzip gleiche Lösungsmenge wie bei i), es kommt nur eine feste Permutation durch die Pivotwahl dazu (die Indizes der Pivots in der ursprünglichen Folge sind $2, 3, 1, 4, 0$):

$[1, 3, 5, 4, 2]$	$[2, 4, 5, 1, 3]$	$[3, 4, 1, 5, 2]$	$[4, 2, 5, 1, 3]$
$[2, 1, 5, 4, 3]$	$[3, 1, 5, 4, 2]$	$[3, 4, 5, 1, 2]$	$[4, 3, 1, 2, 5]$
$[2, 3, 5, 4, 1]$	$[3, 2, 1, 5, 4]$	$[3, 5, 1, 2, 4]$	$[4, 5, 1, 2, 3]$
$[2, 4, 1, 5, 3]$	$[3, 2, 5, 1, 4]$	$[4, 2, 1, 5, 3]$	$[5, 3, 1, 2, 4]$

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], off0: int, off1: int, off2: int);
Function iter_mix_sort(a: [T; n]) {
}
```

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

Mögliche Lösung (s ist die Größe der zu mischenden Teilfolgen):

$p \leftarrow \log_2 n$
for $p' \in [0, p-1]$ **do**
  $s \leftarrow 2^{p'}$
  for $o \in [0, n/(2s) - 1]$ **do**
    $\texttt{off} \leftarrow 2o \cdot s$
    $\texttt{mix}(\texttt{arr}, \texttt{off} + 0s, \texttt{off} + 1s, \texttt{off} + 2s)$
  end
end

Mögliche Lösung (s ist die Anzahl der zu mischenden Teilfolgen):

$p \leftarrow \log_2 n$
for $p' \in [p, 1]$ **do**
  $s \leftarrow 2^{p'}$
  for $o \in [0, s/2 - 1]$ **do**
    $\texttt{off} \leftarrow 2o \cdot n/s$
    $\texttt{mix}(\texttt{arr}, \texttt{off} + 0n/s, \texttt{off} + 1n/s, \texttt{off} + 2n/s)$
  end
end

Alisas Lösung:

offset $\leftarrow$ 1
while *offset* $<$ n **do**
  for *j*$\leftarrow$ *0; j* $<$ *n; j* += *offset* **do**
    mix(a, j, j + offset, j + 2·offset)
  end
  offset $\leftarrow$ 2 $\cdot$ offset
end

Alternativ statt while(offset $<$ n): for(i $\leftarrow$ 0; i $<$ $\log_2(n)$; i++)

Aufgabe 4: Prioritätslisten (18 Punkte)

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

Musterlösung

Heapsort, Dijkstra, Jarnik-Prim

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 {
}
```

Musterlösung

```
Struct PriorityQueue {
    p: [List<int>; k],
}
```

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

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

Musterlösung

```
Function insert(pq: PriorityQueue, prio: int, id:
int) {
   pq.p[prio].pushBack(id)
}
```

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

Name: MUSTERLÖSUNG Matrikelnummer: _____

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

Musterlösung

```
Function deleteMin(pq: PriorityQueue) : int {
    for i ← 0..k do
    |   l ← pq.p[i]
    |   if not l.isEmpty() then
    |   |   return l.popBack()
    |   end
    end
}
```

Aufgabe 5: Graphen (23 Punkte)

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

Vorteil Adjazenzlisten:

Nachteil Adjazenzlisten:

Musterlösung

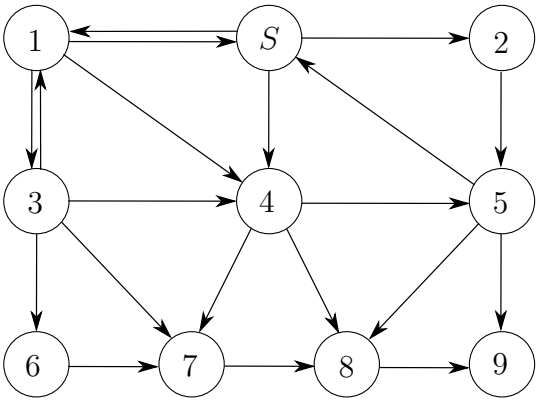
Punkte: je 2 Punkte. Falls mehrere antwort und falsche dabei 0P pro. Falls mehrere und manche richtig, manche irrelevant: noch 1P.
Vorteile: Einfacheres / Schnelleres Einfügen / Löschen von Kanten
Punkte: nur ßugriffszeit"bzw ßchnellere zugriffszeit"nicht präzise genug, 0P
Nachteile: Mehr Speicherplatzbedarf, mehr Cache-Misses
Punkte: Nur "cache misses"noch 1P, weil cache misses haben beide

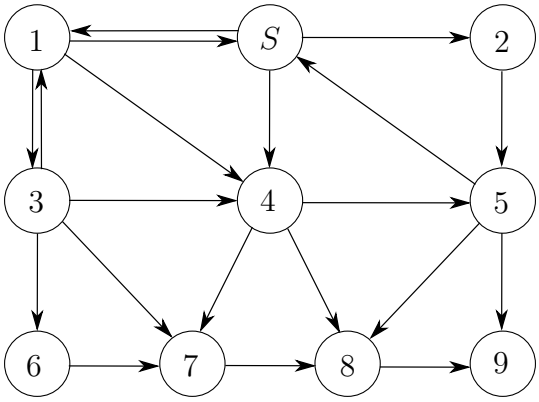
b) Führen Sie auf dem folgenden Graphen eine Breitensuche vom Startknoten S aus durch! Falls mehrere Knoten als Nächstes traversiert werden können, wählen Sie den Knoten mit geringstem Index!

- 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:

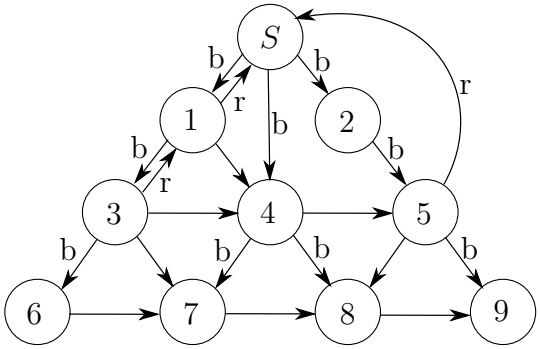




(falls Korrektur notwendig)

Name: MUSTERLÖSUNG Matrikelnummer: _____

Solution:



Musterlösung

Lösungsbild im latex-source auskommentiert.
Anzahl Vorwärtskanten: 0
Punkte: 4P Baumkanten, 3P Rückwärtskanten. -0.5 je falscher oder fehlender b, -1 je falscher oder fehlender r (insbesondere -1P für b/r andersrum beschriftet) , 1P anz vorwärtskanten

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: _____

Musterlösung

Adjazenzmatrix. Kopf: `M : [[int; N]; N]`
`bool` statt `int` auch ok, `|V|` statt `N` auch ok.
Punkte:
- Nennen: 3P
- Kopf: 2P
  1P für korrekte Größe (- 0.5P für off by one, 0P falls nicht symmetrisch, -1P falls V statt N)
  1P für 2D-Array-Struktur erkennbar

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

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

Function getAdjacent(b: int, M : [[int; N]; N])
{
```
for i in 0 .. N-1 do
| if M[i][b] == 1 then
| | print(i)
| end
end
```
}
Punkte:

- print(mat[i][v]) statt print(i): -2P
  print(mat[i]) statt print: -2.5P, das ist nicht definiert.
- komische art auszugeben statt print: -3P
- mat[v][i] statt mat[i][v]: -2P
- Fehlende if-Abfrage: -3P
- if-Abfrage falsch rum: -2P
- 1-Indiziert: -1P / off by one in print: -1P
  Note: falls jemand 1 indiziert aber dann in print(i-1) macht, kein abzug.
- N-1 oder N+1 Schleifendurchläufe: -1P
- i=0, for mat[i][b] do (korrekte if schleife) i += 1: -3P
- Matrix vom typ bool statt int und dann nur if mat[i][v], oder if mat[i][v] == True auch ok.

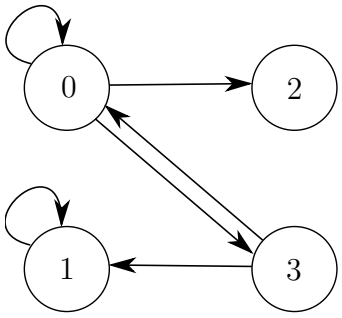
Falls mit anderer Datenstruktur

- im prinzip volle punkte möglich
- for-schleife sowie konkrete implementierung der Struktur muss nachvollziehbar sein. Im Kopf A : Adjazenzliste und dann foreach (x,b) in A: print(x) 0P, weil weder erkennbar wie A implementiert noch wie darüber iteriert wird, außerdem enthält A aus der vl gar nicht direkt solche tupel.

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)**



Musterlösung

Kanten $0 \to 2$ und $3 \to 1$ umdrehen.
Schleifen verloren? -2P
Generell -1 pro fehlender oder falscher Kante, -1P pro zusätzlichem knoten.

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) {
}
```

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

new.next ← head
new.prev ← head.prev
head.prev.next ← new
head.prev ← new
Punkte:

- -2P falls stattdessen an den Anfang angefügt:
  new.prev ← head
  new.next ← head.next
  head.next.prev ← new
  head.next ← new
- -4P falls nur new.next und new.prev gesetzt
- -1P falls letzte zwei Zeilen vertauscht.
- -2P pro fehlender zeile

ii) Die Funktion `reverseGraph` erhält in `graph` einen Graphen in Adjazenzlisten-repräsentation. 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:

Musterlösung

Algorithmus: Punkte: 12 Punkte für den Algorithmus, 6 Laufzeit

```
for a in 0 .. |V|-1 do
   header ← graph[a]
   n ← header.next
   while n.v != -1 do
      b ← n.v
      node : Node
      node.v ← a
      insert(out[b], node)
      n ← n.next
   end
end
return out
```

Punkte:

- für genaue Punkteverteilung muss man wahrscheinlich einfach ein paar Klausuren anschauen.
- Laufzeit vom algorithmus zu langsam: -3P
- Direkt graph[a] als ersten Listenknoten benutzt anstatt vom header wegzugehen: -2P
- Abbruchbedingung eins zu früh / spät: -2P. Abbruchbedingung so, dass while-schleife nie läuft: -4P.
- neue Node hat falschen Wert bei v: -3P
- Off-By-One / Out of Bounds: -1P pro.

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

Laufzeit: Punkte: 6P Laufzeitanalyse

Die Operationen in den Schleifen haben alle konstante Laufzeit Punkte: 1
Die äußere For-Schleife läuft $|V|$ mal durch Punkte: 1
Die innere While-Schleife läuft *über alle Ausführungen der for-Schleife gesammelt* $\mathcal{O}(|E|)$ mal durch. Punkte: 3P. -2P falls nur gesagt wird die While-Schleife läuft je For-Durchlauf $\mathcal{O}(|V|)$ mal durch, das ist nicht möglichst eng
$\Rightarrow$ Was nur in der For-Schleife steht wird $\mathcal{O}(|V|)$ mal ausgeführt. Punkte: zwischenschritt ohne punkte
Was in der While-Schleife steht wird $\mathcal{O}(|E|)$ mal ausgeführt. Punkte: zwischenschritt ohne punkte
Daraus ergibt sich $\mathcal{O}(|V| + |E|)$ bzw. alternativ $\mathcal{O}(\max(|V|, |E|)$ Punkte: 1P

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

Musterlösung

Dijkstra: $\mathcal{O}((|V| + |E|) \log |V|)$
Bellman Ford: $\mathcal{O}(|V| \cdot |E|)$
Punkte: je 2P, je -1P falls Betrag vergessen und falls m / n statt V / E benutzt.

- O((m+n) log m): 1 von 2 Punkten
- O((m+n) log n): auch 1 von 2 Punkten.
- mehr Falsch: 0P.

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

Musterlösung

Die kürzeste bisher gefundene Distanz bis zum Knoten.
Punkte: 1P für bisher gefundene, 1p für kürzeste distanz.
0P für kantengewichte

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)**

Musterlösung

DecreaseKey Punkte: 1
Um die bisher gefundene minimale Distanz eines Knotens zu verringern, wenn ein kürzerer Weg dorthin gefunden wurde.
Punkte: 1P Decrease, 2P wofür oder wann

- verwendet um die bisher gefundene distanz zu verringern.
- verwendet, wenn zu einem knoten ein kürzerer weg gefunden wurde. (2P)
- um die priorität zu aktualisieren: 0P (hat nichts mit dijkstra insbesondere zu tun)

Name: MUSTERLÖSUNG Matrikelnummer: _____

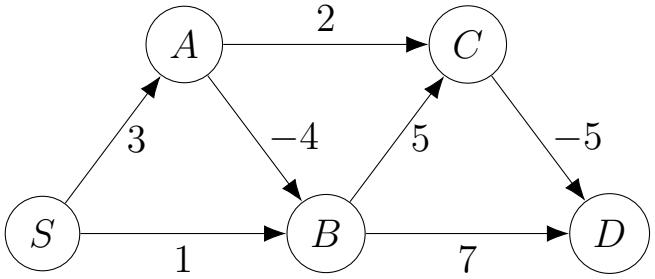
d) Wie kann die Ausführung des Dijkstra-Algorithmus beschleunigt werden, wenn nur die Distanz vom Startknoten $s \in V$ zu einem bestimmten Knoten $v \in V$ statt zu allen Knoten gesucht wird? Begründen Sie, warum Ihre Lösung weiterhin ein korrektes Ergebnis liefert! **(6 Punkte)**

Musterlösung

Abbrechen, sobald v gescannt wurde. Das ist möglich, weil einmal gescannte Knoten mit ihren Distanzen nicht mehr verändert werden.
Punkte: 2 Pkt für Methode, 4P für die Begründung
Punkte: Punkte für andere antworten:

- Abbruch sobald die minimale distanz zu v gefunden wurde, da die distanz danach nicht mehr verringert werden kann: 4P (0P für methode weil nicht klar ist woran man das erkennt, aber begründung im prinzip ok)
- Methode: Abbruch, sobald die Distanzen von nicht besuchten knoten über der aktuellsten distanz von v liegen: auch ok, volle punkte für methode (weil effektiv der fall sobald v besucht wird)
- Methode. Abbruch, sobald distanzen von knoten über der aktuellen distanz von v liegen: noch 1 von 2 Methodenpunkten, da älle übrigen"distanzen fehlt
- Abbruch, sobald ein Weg zu v gefunden wurde: 0P, da danach noch kürzerer weg gefunden werden könnte.
- Lösche alle Knoten, die nicht nach v führen, vom Heap: Falsch, um das zu erkennen braucht man ggf länger als einfach ausführen. Hilft nichts in vollständigem Graph.
- Generell: Methoden die nicht (immer) funktionieren: 0P für alles.

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										

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

	S		A		B		C		D		
Iteration	*Pre*	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*	*Pre*	*Dist*	
0	-	0	-	∞	-	∞	-	∞	-	∞	
1	-	0	S	3	S	1	-	∞	-	∞	
2	-	0	S	3	A	-1	A	5	B	8	
3	-	0	S	3	A	-1	B	4	C	0	
4	-	0	S	3	A	-1	B	4	C	-1	

Punkte: -0.5 pro falsch ausgefüllter Zelle

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)**

$W_c =$

Musterlösung

$W_c = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$
Punkte: -1P pro off by one
Kein Abzug falls (v_i, v_{i+1}) nicht geklammert, also egal ob $w(v_i, v_{i+1})$ oder $w((v_i, v_{i+1}))$.
Keine Punkte für $-\infty$. Der Kreis ist ein endlich langer Pfad mit endlich negativen Gewichten pro Kante.

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)

Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

- Annahme: $\forall$ Kanten $(v_i, v_{i+1}) \in C : d(v_i)+w(v_i, v_{i+1}) >= d(v_{i+1})$ Punkte: 2 Punkte
- Da C ein Kreis ist, folgt daraus für zwei aufeinanderfolgende Kanten (v_i, v_{i+1}) und (v_{i+1}, v_{i+2})
  $d(v_i) + w(v_i, v_{i+1}) + w(v_{i+1}, v_{i+2}) \geq d(v_{i+2})$ Punkte: 3 Punkte
  und daraus
  $d(v_i) + \sum_{j=0}^{k-1} w(v_j, v_{j+1}) >= d(v_i)$ Punkte: 3 Punkte. Direkt 6P falls Schritt davor übersprungen.
- also
  $d(v_i) + W_c \geq d(v_i)$. Punkte: 2P für Verbindung W_c und Summe. Direkt 8P falls Schritt davor übersprungen. Beide Schritte übersprungen: nur 4 von 8P.
- C ist negativer Kreis $\Rightarrow W_c < 0$ Punkte: 4P, davon 2 für "weil negativer Kreisünd 2 für $W_c < 0$
- Damit gilt $d(v_i) + W_c < d(v_i)$. Widerspruch. Punkte: 2P. Wenn explizit vorher auch ... $\geq$ ... steht, muss man nicht Widerspruch schreiben.

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)**

Musterlösung

Schnitt-Eigenschaft: leichteste (Kante zwischen zwei Teilmengen / Schnittkante) kann in MST verwendet werden

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

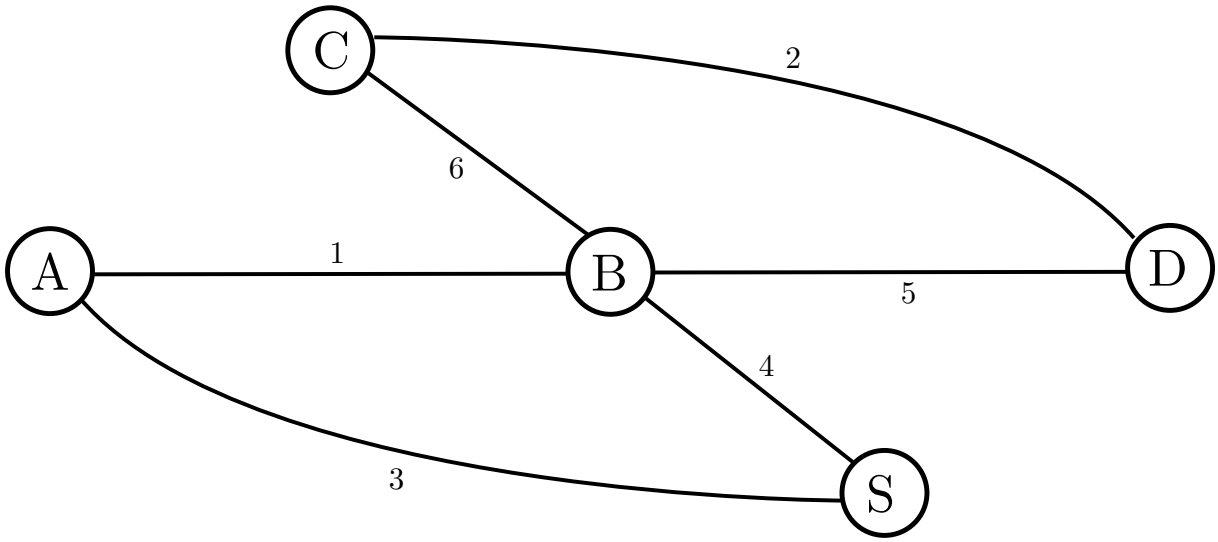
Werte:

Priorität:

Musterlösung

Werte: die vom aktuellen Teil-MST aus erreichbaren Knoten Priorität: minimales Gewicht einer Kante zwischen Teil-MST und dem Knoten

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)**



Name: MUSTERLÖSUNG Matrikelnummer: _____

Musterlösung

Schritt	1	2	3	4	5	6
Betrachtete Kante	{A,B}	{C,D}	{A,S}	{B,S}	{B,D}	{B,C}
Im MST (✓/X)	✓	✓	✓	X	✓	X

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

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

Musterlösung

Um herauszufinden, ob eine Kante zwei Teilbäume verbindet.

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

Musterlösung

Sie verbessert die amortisierte Laufzeit von $\mathcal{O}(n)$ zu $\mathcal{O}(\log(n))$.

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)` =

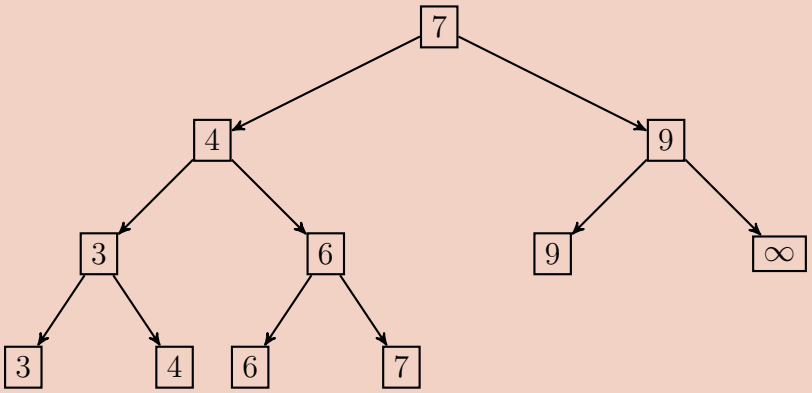
Musterlösung

`T.locate(k)` $= \min\{e \in M : e \geq k\}$

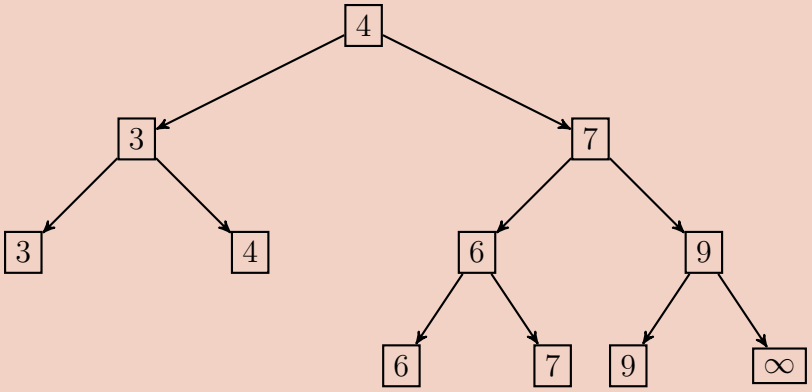
b) Geben Sie einen binären Suchbaum mit minimaler Höhe für die Sequenz $\langle 4, 3, 6, 9, 7, \infty \rangle$ an! **(5 Punkte)**

Name: MUSTERLÖSUNG Matrikelnummer: _____

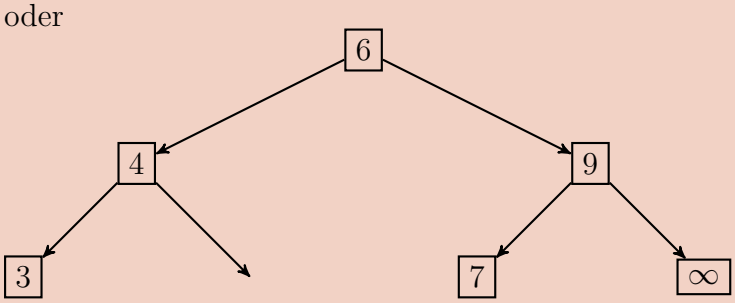
Musterlösung



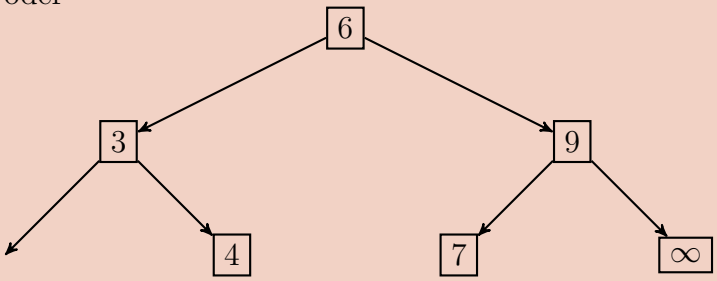
oder



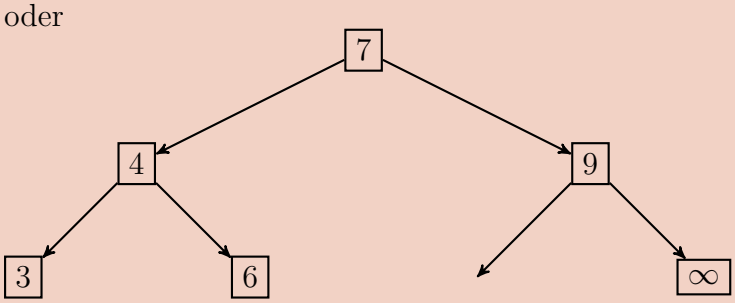
oder



oder



oder



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)**

Musterlösung

Zugriff auf i-tes Element in der Liste der Blätter , $\mathcal{O}(n)$

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],
}
```

Musterlösung

```
Struct ABItem {
   degree: int,
   splitters: [Key; b - 1],
   children: [ABHandle; b],
   n: [int; b], // Anzahl der Blätter unter diesem
    Kindknoten
}
```

Name: MUSTERLÖSUNG Matrikelnummer: _____

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 {
}
```

Musterlösung

```
Function select(t: ABTree, k: int) : Element {
    handle ← t.root
    left ← 0
    for _ ← 0..t.height do
        item ← ABItem(handle)
        for i ← 0..b do
            if left + item.n[i] < k then
                left ← left + item.n[i]
            else
                handle ← item.children[i]
                break
            end
        end
    end
    return Element(handle)
}
```