Karlsruher Institut für Technologie

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

Hauptklausur Algorithmen I
SS 2020

26. September 2020

Name:	
Matrikelnummer:	
Klausurcode:	

Beachten Sie:

- Schreiben Sie Ihren **Klausurcode** (2 Buchstaben), vollständigen **Namen** und **Matrikelnummer** in Druckschrift in das Feld auf dem Deckblatt! Sie sollten Ihren Code gut aufheben, um später Ihre Note zu erfahren.
- Schreiben Sie Ihre Matrikelnummer oben auf jedes bearbeitete Aufgabenblatt.
- Schreiben Sie Ihre Lösungen auf die Aufgabenblätter. Bei Bedarf können Sie weiteres Papier anfordern, welches Sie am Ende der Klausur mit in den Umschlag stecken.
- Halten Sie sich bei Pseudocode-Aufgaben an die Notation der Vorlesung / Übung, um Missverständnissen vorzubeugen.
- Wir akzeptieren auch englische Antworten.
- Sie dürfen **ein handbeschriebenes Din A4-Blatt** mitbringen, sonst sind **keine Hilfsmittel** zugelassen.
- Sie haben **120 Minuten** Bearbeitungszeit.
- Die Klausur umfasst 24 Seiten (12 Blätter) mit 9 Aufgaben.
- Tragen Sie bitte außerhalb von Ihrem Platz einen Mund-Nasen-Schutz.

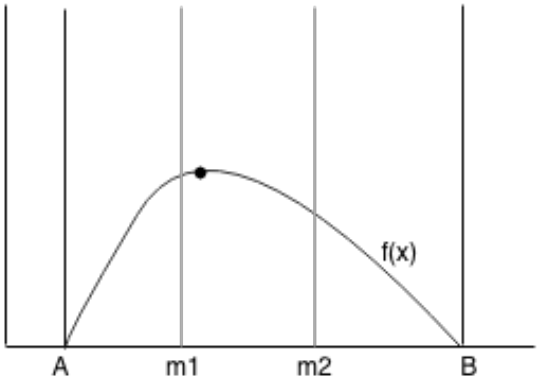
Aufgabe	1	2	3	4	5	6	7	8	9	**Gesamt**
Erreichte Punkte										
Erreichbare Punkte	15	26	24	20	22	25	30	34	44	**240**

Note

Aufgabe 1: Laufzeit (15 Punkte)

a) Zeigen Sie, dass $g(n) := n + n \log_2 n \in \mathcal{O}(n^2)$! **(5 Punkte)**

b) Die Funktion f habe im Intervall $[A, B]$ nur ein lokales Maximum. Dieses Maximum lässt sich rekursiv bestimmen, indem man das Suchintervall um ein Drittel pro Schritt verkleinert.



- Ausgehend vom Intervall $[A, B]$ bestimmt man die Funktionswerte $f(m_1)$ und $f(m_2)$ an den inneren Grenzen $m_1 = (2A + B)/3$ und $m_2 = (A + 2B)/3$.
- Wenn $f(m_1) \leq f(m_2)$, betrachtet man im Anschluss das Intervall $[m_1, B]$, ansonsten das Intervall $[A, m_2]$, und fährt darin rekursiv mit der Suche fort.
- Wenn die Intervallgrenzen weniger als ε auseinander liegen, wird dieses Intervall zurückgegeben.
- Der Algorithmus kann damit $n \approx (B - A)/\varepsilon$ verschiedene Intervalle ausgeben.

i) Stellen Sie eine Rekurrenz für die Laufzeit $T(n)$ auf! **(6 Punkte)**

ii) Welche Laufzeit erwarten Sie in $\mathcal{O}$-Notation und warum? **(4 Punkte)**

Matrikelnummer: _____

Aufgabe 2: Listen und Hashing (26 Punkte)

a) Im Folgenden soll ein Stapel (Stack) mittels eines unbeschränkten Feldes (`UArray`) implementiert werden.

i) Was ist die Worst Case-Laufzeit und die amortisierte Laufzeit für das Einfügen eines neuen Elements in einen solchen Stack in $\mathcal{O}$-Notation? **(2 Punkte)**

Worst Case:

Amortisiert:

ii) Nennen Sie einen Vorteil und einen Nachteil, einen Stack mit `UArrays` statt mit verketteten Listen zu implementieren! **(4 Punkte)**

Vorteil `UArray`:

Nachteil `UArray`:

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

Hashfunktion

Element	a	b	d	f	m	v	s	t
Hash-Wert	0	1	3	5	3	3	0	1

Hashtabelle

Index	0	1	2	3	4	5	6	7	8
Wert	a			d	m	v			

☐ Führen Sie nun nacheinander die folgenden Operationen durch und geben Sie den Zustand der Hashtabelle nach Ausführen jeder Operationen an! **(4 Punkte)**

Index	0	1	2	3	4	5	6	7	8
`insert(s)`									
`insert(f)`									
`remove(v)`									
`insert(b)`									

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

Index	0	1	2	3	4	5	6	7	8
`insert(s)`									
`insert(f)`									
`remove(v)`									
`insert(b)`									

Matrikelnummer: _____

c) Um die Ausbreitung des neuartigen Coronavirus in der Bevölkerung zu messen, wurde ein Datensatz $\mathcal{D}$ von positiv getesteten Personen erstellt. $\mathcal{D}$ enthält $n \in \mathbb{N}$ Tripel $(p, L, t) \in \mathcal{D}$ mit einer eindeutigen Personalausweisnummer $p \in \mathbb{N}$, einem Landkreis $L \in \{0, \ldots, 294\}$ und den vergangenen Tagen $t \in \mathbb{N}$ seit einem positiven Test.

Personen können mehrfach positiv getestet worden sein. Es kann daher mehrere Einträge (p_i, L_i, t_i) und (p_j, L_j, t_j) mit der gleichen Personalausweisnummer $(p_i = p_j)$ geben, welche aber von einem anderen Tag $(t_i \neq t_j)$ oder Landkreis $(L_i \neq L_j)$ stammen.

i) Geben Sie in der Funktion `findPersons` einen Algorithmus in Pseudocode an, der mit Speicherbedarf $\mathcal{O}(n)$ in erwarteter Laufzeit $\mathcal{O}(n)$ alle Personalausweisnummern der Personen ausgibt, die in den letzten 14 Tagen $(t \leq 14)$ positiv getestet wurden! Geben Sie weiterhin die Tage seit dem *letzten* positiven Test aus! Keine Personalausweisnummer soll mehr als einmal ausgegeben werden. Mit der Funktion `print` (p, t) können Sie eine Personalausweisnummer p und die vergangenen Tage t ausgeben. **(12 Punkte)**

ii) Begründen Sie das erwartete Laufzeitverhalten Ihres Algorithmus aus i)! **(4 Punkte)**

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

Aufgabe 3: Sortieren und Selektieren (24 Punkte)

a) Nennen Sie je einen Vorteil von *Quicksort* und *Mergesort*! **(2 Punkte)**

Vorteil *Quicksort*	Vorteil *Mergesort*

b) Gegeben sei die Sequenz $\langle 28, 24, 21, 70, 56, 82, 23, 61, 47, 18\rangle$. Bestimmen Sie das Element mit Rang $k = 7$ mit Hilfe des *Quickselect*-Algorithmus aus der Vorlesung! Geben Sie jeweils für jeden Rekursionsschritt die aktuell betrachtete Teilsequenz S, den aktuellen Wert für k sowie die Sequenzen $a := \langle e \in S : e < p\rangle$, $b := \langle e \in S : e = p\rangle$ und $c := \langle e \in S : e > p\rangle$ an! Die Elemente von a, b und c müssen dabei die gleiche Reihenfolge wie in S haben. Wählen Sie als Pivot immer das erste Element von S! **(10 Punkte)**

S	k	Sequenzen
$\langle 28, 24, 21, 70, 56, 82, 23, 61, 47, 18\rangle$	7	a: b: c:
		a: b: c:
		a: b: c:
		a: b: c:

Das Element mit Rang $k = 7$ ist ____ .

Matrikelnummer: _____

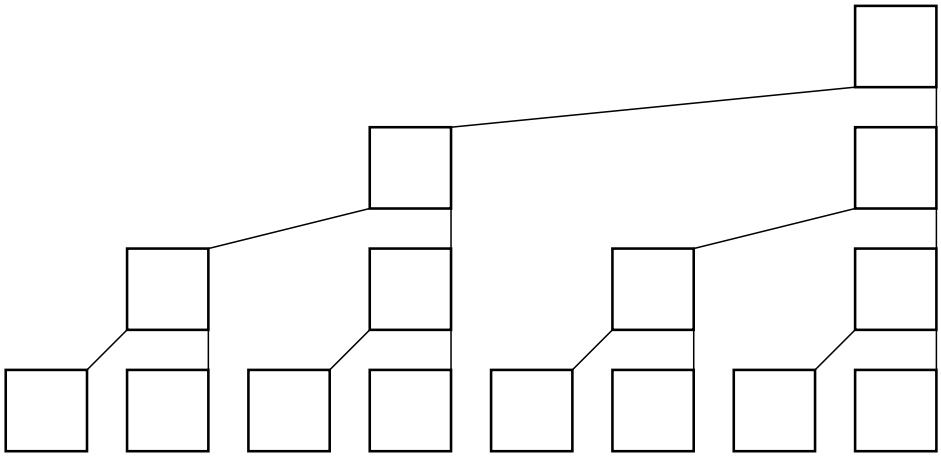
c) Sequenzen der Form $x_0 \leq \cdots \leq x_k \geq \cdots \geq x_{n-1}$ mit $0 \leq k < n$ werden *bitonisch* genannt (z. B. $\langle 1, 2, 5, 10, 9, 8, 4 \rangle$). Entwerfen Sie in Pseudocode einen Algorithmus `sortBitonicSeq`, der die Elemente einer bitonischen Sequenz in aufsteigend sortierter Reihenfolge mit der Funktion `print(i: int)` ausgibt! Die Laufzeit muss dabei in $\mathcal{O}(n)$ liegen. Die Eingabe liegt als Array a vor und darf nicht verändert werden. Für Lösungen mit mehr als $\mathcal{O}(1)$ Speicherbedarf werden nur Teilpunkte vergeben. Begründen Sie außerdem die Laufzeit Ihres Algorithmus! **(12 Punkte)**

Function sortBitonicSeq(a : [int; n])
{

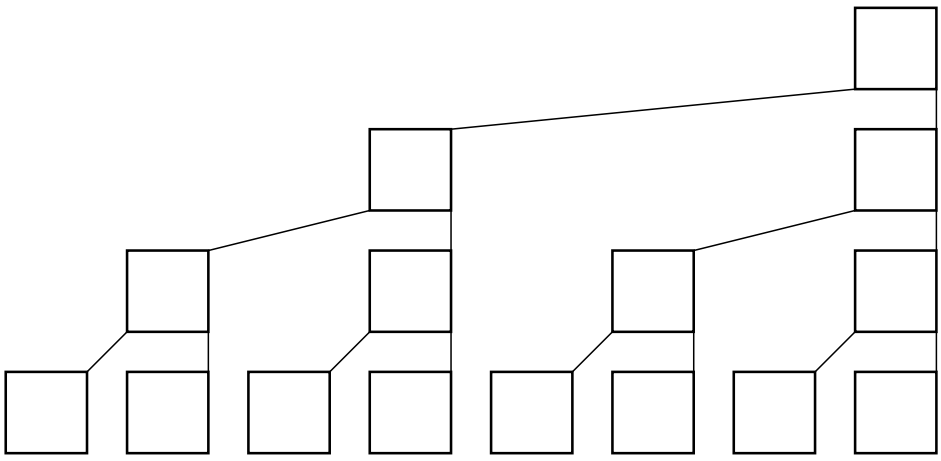
Aufgabe 4: Heaps (20 Punkte)

Kartesische Bäume zu einer Sequenz $S = \langle s_1, s_2, ... \rangle$ sind besondere binäre Min-Heaps, bei denen neben der Heap-Eigenschaft eine Ordnung für die Geschwisterknoten gilt: Jeder innere Knoten s_j enthält in seinem linken Kindteilbaum nur Elemente, die in der Sequenz s_j vorangehen, und in seinem rechten Kindteilbaum nur Elemente, die in der Sequenz s_j nachfolgen. Dadurch kann durch eine geordnete Traversierung des Binärbaums die ursprüngliche Sequenz wieder hergestellt werden.

a) Geben Sie den kartesischen Baum zur Sequenz $S = \langle 9, 3, 1, 5, 4, 2, 7, 8 \rangle$ an! **(8 Punkte)**



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



Matrikelnummer: _____

b) Füllen Sie die Methode `insert` in Pseudocode aus, sodass die Funktion `makeCartesianTree` einen kartesischen Baum aus der Sequenz S konstruiert! **(12 Punkte)**

```
Struct Node
{
    s : int
    l : Node
    r : Node
}
```

Function makeCartesianTree(S : [int; N]) : Node

```
root ← ⊥
for s ∈ S do
 |  root ← insert(root, Node(s[i], ⊥, ⊥))
end
return root
```

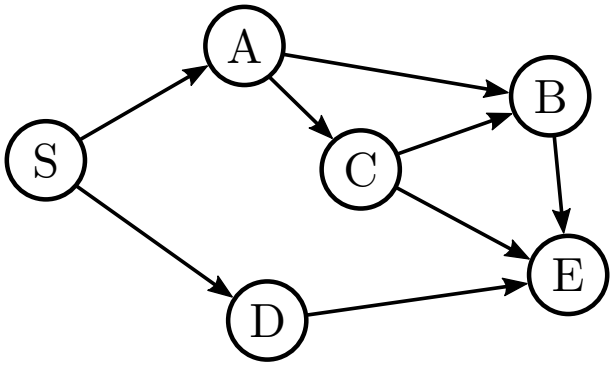
```
// Füge new.s in den Heap ein, gebe neuen Wurzelknoten zurück
Function insert(root: Node, new: Node) : Node
{




}
```

Aufgabe 5: Graphen (22 Punkte)

Gegeben sei der folgende gerichtete Graph:



a) Geben Sie die Adjazenzfeldrepräsentation und Adjazenzmatrix des Graphen an! Beachten Sie die Sortierung $S, A, B, \ldots, E$ der Knoten!

i) Adjazenzfeld: **(4 Punkte)**

V: | | | | | | | | | | | | | | |

E: | | | | | | | | | | | | | | |

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

V: | | | | | | | | | | | | | | |

E: | | | | | | | | | | | | | | |

ii) Adjazenzmatrix: **(4 Punkte)**

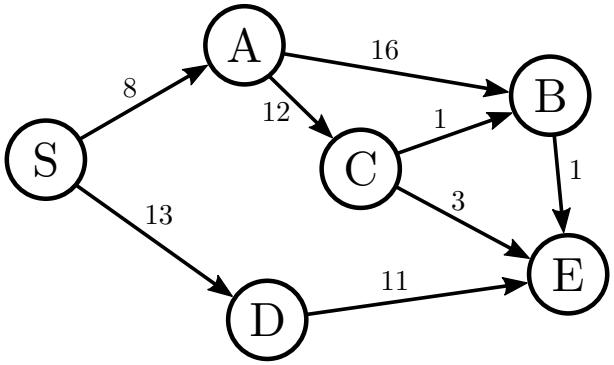
$$\begin{pmatrix} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{pmatrix}$$

Matrikelnummer: _____

b) Vergleichen Sie die Adjazenzmatrix und Adjazenfeldrepräsentation für einen Graphen $G = (V, E)$, indem Sie in $\mathcal{O}$-Notation für jede Repräsentation möglichst enge Schranken für den Speicherbedarf sowie die Worst Case-Laufzeit für das Einfügen einer neuen Kante angeben! **(4 Punkte)**

	Speicherbedarf	Einfügen
Adjazenzmatrix		
Adjazenzfeld		

c) Führen Sie auf dem folgenden Graphen Dijkstras Algorithmus aus, beginnend mit Knoten S! Füllen Sie dazu untenstehende Tabelle aus! Geben Sie pro Schritt die bisher gefundenen Entfernungen der Knoten an und markieren Sie den aktuellen Knoten, wie beispielhaft für den ersten Schritt angegeben! Sie müssen nur Werte angeben, die sich ändern. **(10 Punkte)**



	1	2	3	4	5	6	7	8	9	10
S	⓪									
A	∞	8								
B	∞									
C	∞									
D	∞	13								
E	∞									

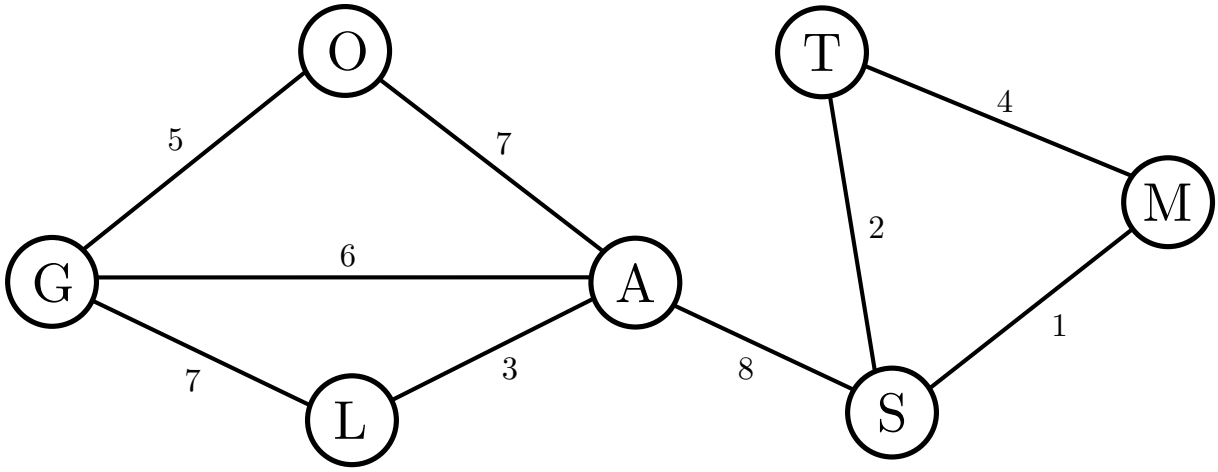
Aufgabe 6: Minimale Spannbäume (MST) (25 Punkte)

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

i) Schnitt-Eigenschaft:

ii) Kreis-Eigenschaft:

b) Bestimmen Sie den minimalen Spannbaum 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! **(9 Punkte)**



Schritt	1	2	3	4	5
Betrachtete Kante	{ , }	{ , }	{ , }	{ , }	{ , }
Im MST ($\checkmark/X$)					

Schritt	6	7	8	9
Betrachtete Kante	{ , }	{ , }	{ , }	{ , }
Teil des MST ($\checkmark/X$)				

Matrikelnummer: _____

c) Welche Datenstrukturen verwendet man für eine effiziente Implementierung des Kruskal-Algorithmus? Wofür werden diese Datenstrukturen verwendet? **(4 Punkte)**

i)

ii)

d) Gegeben sei der Algorithmus von Jarník und Prim zur Bestimmung eines minimalen Spannbaums eines gewichteten, gerichteten Graphen $G = (V, E)$ mit Knoten $V : [int; n]$ und Kanten $E : [(int, int); m]$:

Function MSTJarnikPrim(V : [int; n], E : [(int, int); m]) : [int; n-1]

```
d : [float; n] ← [∞]
s : int ← V[0]
parent : [int; n]

parent[s] ← s
d[s] ← 0

Q ← ...      // siehe i)
Q.insert(s)
while Q not empty do
    u ← Q.deleteMin()
    d[i] ← 0
    for each e ← (v,v') ∈ E where v=u do
        if weight(e) < d[v'] then
            d[v'] ← weight(e)
            parent[v'] ← u
            if v ∈ Q then Q.decreaseKey(v')
            else Q.insert(v')
return [(parent[v], v) : v ∈ V \ {s}]
```

i) Geben Sie eine für `Q` geeignete Datenstruktur an, welche die im Algorithmus benötigten Operationen effizient unterstützt! **(2 Punkte)**

ii) Welche asymptotischen Laufzeiten haben die im Algorithmus auf `Q` ausgeführten Operationen für die Datenstruktur aus i) pro Aufruf? Wie oft werden die Operationen asymptotisch aufgerufen? Geben Sie möglichst enge Schranken in $\mathcal{O}$-Notation an! **(6 Punkte)**

Operation	**Laufzeit**	**Aufrufe**
`insert`		
`decreaseKey`		
`deleteMin`		

Aufgabe 7: 2-SAT-Problem (30 Punkte)

Das 2-SAT-Problem ist ein Erfüllbarkeitsproblem für aussagenlogische Formeln, die Konjunktionen ($\wedge$) von *maximal zweistelligen* Disjunktionen ($\vee$) sind. Im Gegensatz zum allgemeinen SAT-Problem sind für das 2-SAT-Problem Polynomialzeitalgorithmen bekannt.

Diese machen sich bekannte Graphenalgorithmen zu Nutze, um zu berechnen, ob für eine gegebene aussagenlogische Formel F eine Belegung der Variablen mit Werten aus $\{0, 1\}$ so existiert, dass F erfüllt ist. Im Folgenden bezeichnen wir die Menge der Variablen einer Formel F mit $\mathbb{X}_F$.

Kleiner aussagenlogischer Evaluationsleitfaden: Eine Konjunktion ($\wedge$, *"und"*) ist genau dann *erfüllt*, wenn *alle* ihre Argumente den Wert 1 annehmen. Eine Disjunktion ($\vee$, *"oder"*) ist genau dann *nicht erfüllt*, wenn *alle* ihre Argumente den Wert 0 annehmen. Die Negation $\overline{v}$ einer Variablen v ist genau dann erfüllt, wenn v den Wert 0 annimmt. Die doppelte Negation $\overline{\overline{v}}$ vereinfachen wir zu v. Eine Implikation ($\rightarrow$, *"impliziert"*) ist genau dann *nicht* erfüllt, wenn die linke Seite den Wert 1 und die rechte Seite den Wert 0 annimmt.

a) Geben Sie für die 2-SAT-Formel $F_a := (a \vee b) \wedge (\overline{a} \vee \overline{b}) \wedge c$ alle erfüllenden Belegungen der Variablen $\mathbb{X}_F = \{a, b, c\}$ an! **(4 Punkte)**

b) Der Implikationsgraph $G_F = (V, E)$ einer 2-SAT-Formel F wird wie folgt erzeugt: Für jede Variable $v \in \mathbb{X}_F$ gibt es zwei Knoten v und $\overline{v}$. Analog zu den Äquivalenzen $(a \vee b) \equiv (\overline{a} \rightarrow b) \equiv (\overline{b} \rightarrow a)$, werden für jede Disjunktion $(a \vee b)$ aus F zwei Kanten $(\overline{a}, b)$ und $(\overline{b}, a)$ eingeführt. Konkret:

$$V := \mathbb{X}_F \cup \{\overline{x} \mid x \in \mathbb{X}_F\}$$
$$E := \{(\overline{a}, b), (\overline{b}, a) \mid (a \vee b) \in F\}$$

Zeichnen Sie den Implikationsgraphen der Formel F_a aus Aufgabenteil a)! Beachten Sie dabei, dass einstellige Formelglieder durch zweistellige Disjunktionen ersetzt werden können, also z.B. $c \equiv c \vee c$! **(6 Punkte)**

Matrikelnummer: _____

c) Schiefsymmetrie

Sei G_F Implikationsgraph einer beliebigen 2-SAT-Formel F. Zeigen Sie: Gibt es in einen Pfad $P := (k_0, \ldots, k_n)$ in G_F, dann gibt es auch einen Pfad $(\overline{k}_n, \ldots, \overline{k}_0)$ in G_F. **(8 Punkte)**

d) Transitivität

Sei G_F Implikationsgraph einer beliebigen 2-SAT-Formel F. Zeigen Sie: Wenn es einen Pfad $(k_0, \ldots, k_n)$ in G_F gibt, dann sind alle erfüllenden Belegungen von F auch erfüllende Belegungen der Formel $K := k_0 \to k_n$, d.h. K ist eine logische Konsequenz von F. **(12 Punkte)**

Aufgabe 8: Sortierte Listen (34 Punkte)

a) Welche Invariante ermöglicht die effiziente Suche in einem binären Suchbaum? **(4 Punkte)**

b) Sei h die Höhe eines (a, b)-Baums mit $n \geq 2$ Elementen und $2 \leq a < b$. Zeigen Sie: $h \in \Theta(\log n)$! Betrachten Sie dabei a und b als Konstanten! **(10 Punkte)**

c) Nennen Sie *je eine* Operation aus der Vorlesung, die für ein sortiertes Array oder einen (a, b)-Baum (ohne Augmentierungen) eine bessere asymptotische Laufzeit hat! **(4 Punkte)**

effizient für sortiertes Array:

effizient für (a, b)-Baum:

Matrikelnummer: _____

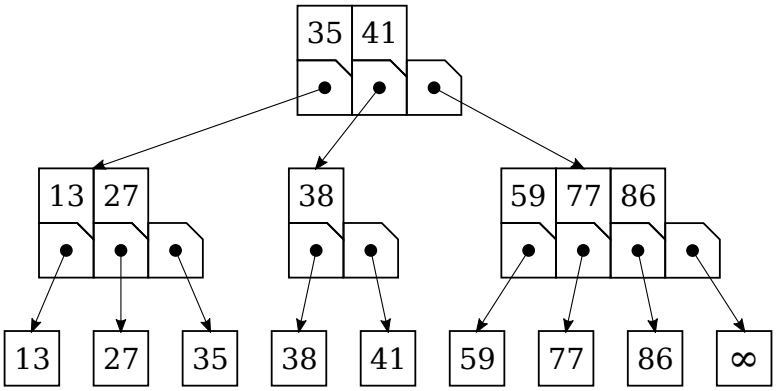
d) Nennen Sie *je einen* Vorteil von (a, b)-Bäumen und Hashtabellen gegenüber der jeweils anderen Datenstruktur! Gehen Sie davon aus, dass nur von beiden unterstützte Operationen benötigt werden und nehmen Sie eine gut gewählte Hashfunktion an! **(4 Punkte)**

Vorteil (a, b)-Baum:

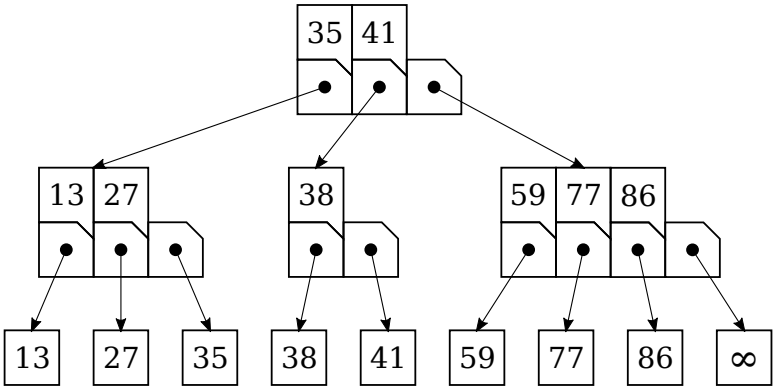
Vorteil Hashtabelle:

e) Führen Sie im folgenden $(2, 5)$-Baum die Operation *locate* mit den jeweils gegebenen Schlüsseln aus! Umkreisen Sie dazu alle betrachteten Spalt-Schlüssel, kreuzen Sie die verwendeten Kanten im Baum an und unterstreichen Sie das Ergebnis! **(4 Punkte)**

`locate(77):`

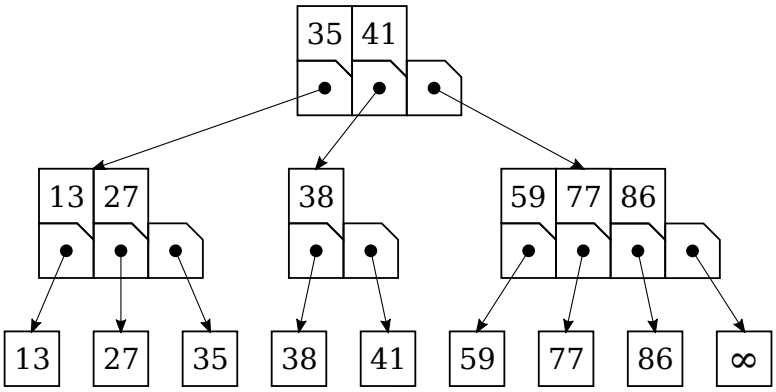


`locate(37):`

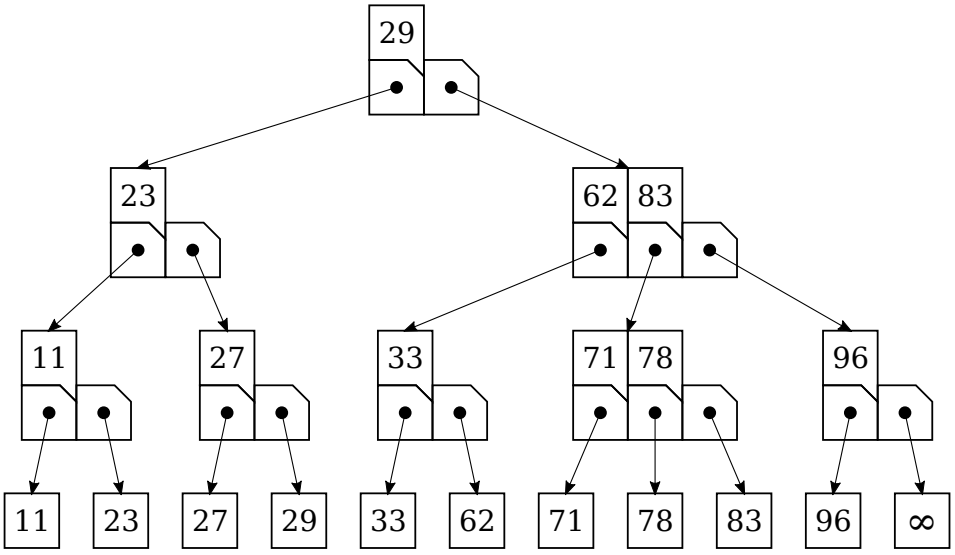


Weiterer Platz, falls Sie Ihre Antwort korrigieren wollen. Falls Sie diesen nutzen, kennzeichnen Sie eindeutig, welche Lösung gewertet werden soll:

`locate( ):`



f) Führen Sie auf dem folgenden $(2, 5)$-Baum die Operation `remove(27)` aus und geben Sie den resultierenden Baum an! **(8 Punkte)**



Matrikelnummer: _____

Aufgabe 9: Edit-Distance (44 Punkte)

Die Levenshtein-Distanz $L_{a,b}$ der beiden Strings a und b mit Länge $|a|$ und $|b|$ kann verwendet werden, um die Ähnlichkeit von a und b zu bestimmen. Sie quantifiziert, wie viele Operationen (z.B. Einfügen von Zeichen) nötig sind, um a in b zu überführen und ist wie folgt rekursiv definiert:

$$L_{a,b} = l_{a,b}(|a|, |b|)$$

$$l_{a,b}(i,j) = \begin{cases} max(i,j) & \text{wenn } min(i,j) = 0, \\ min \begin{cases} l_{a,b}(i-1,j)+1 \\ l_{a,b}(i,j-1)+1 \\ l_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \textit{sonst} \end{cases}$$

wobei

$$1_{(x \neq y)} = \begin{cases} 0 & \text{wenn } x = y, \\ 1 & \textit{sonst}. \end{cases}$$

a) Die Levenshtein-Distanz kann effizient mit Dynamischer Programmierung (DP) berechnet werden. Vervollständigen Sie hierzu die vorgegebene DP-Matrix für $a =$ "passiv" und $b =$ "aktiv"! **(6 Punkte)**

a \ b		a	k	t	i	v
	0	1	2	3	4	5
p	1					
a	2					
s	3	2	2	3	4	5
s	4	3	3	3	4	5
i	5					
v	6					

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

a \ b		a	k	t	i	v
	0	1	2	3	4	5
p	1					
a	2					
s	3	2	2	3	4	5
s	4	3	3	3	4	5
i	5					
v	6					

a \ b		a	k	t	i	v
	0	1	2	3	4	5
p	1					
a	2					
s	3	2	2	3	4	5
s	4	3	3	3	4	5
i	5					
v	6					

Matrikelnummer: _____

b) Die Levenshtein-Distanz definiert drei Operationen, um a in b zu überführen.

1.	$R(i)$	Löschen von Buchstabe a_i
2.	$I(i,j)$	Einfügen von Buchstabe b_j an der Position i in a
3.	$X(i,j)$	Ersetzen von Buchstabe a_i durch b_j

Nun soll "passiv" mit einer minimalen Anzahl Operationen in "aktiv" umgewandelt werden. Bestimmen Sie mithilfe der ausgefüllten Tabelle aus Teilaufgabe a):

i) Wie kann die Anzahl der mindestens benötigten Operationen mithilfe der ausgefüllten DP-Matrix effizient (in konstanter Zeit) ermittelt werden? **(3 Punkte)** ☐

ii) Wie ermittelt man effizient die benötigten Operationen mithilfe der ausgefüllten DP-Matrix? Geben Sie eine Folge von Operationen gemäß der DP-Matrix in der Kurznotation ($R(i)$, $I(i,j)$, $X(i,j)$) an, sodass "passiv" in "aktiv" umgewandelt wird! **(6 Punkte)** ☐

iii) Ist die Folge an Operationen, welche den einen String in den Anderen umwandelt, eindeutig? Begründen Sie Ihre Antwort!**(3 Punkte)** ☐

c) Bei Smartphone-Tastaturen werden oft für Korrekturvorschläge die n besten Worte aus einem Wörterbuch bestimmt. Die Anzahl der Worte im Wörterbuch ist dabei sehr groß. In dieser Aufgabe sollen Sie einen Algorithmus in Pseudocode entwerfen, welcher beliebige Eingabestrings entgegennimmt und das Wort mit der geringsten Levenshtein-Distanz aus dem Wörterbuch findet. Nutzen Sie die Sortiertheit aus!

Im folgenden Pseudocode soll `a:[char]` der String `a` mit beliebiger Länge sein. Sie können mit `|a|` die Länge des Strings auslesen und mit `a[i]` für $i \in [0, |a| - 1]$ auf das i-te Zeichen im String zugreifen.

Gehen Sie beim Entwurf wie folgt vor:

i) Entwerfen Sie eine Funktion in Pseudocode, um die Länge des längsten gemeinsamen Präfixes zweier Strings a und b zu bestimmen! **(6 Punkte)**

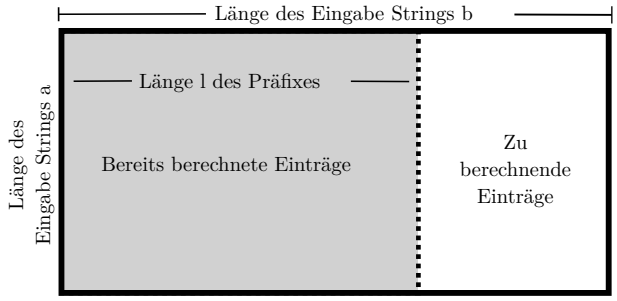
Function longestPrefixLength(a : [char], b : [char]) : int
{

Matrikelnummer: _____

ii) Entwerfen Sie eine Funktion in Pseudocode, um die DP-Matrix zur Bestimmung der Levenshtein-Distanz zu berechnen! Dafür erhalten Sie als Eingabe eine DP-Matrix, die bereits teilweise gefüllt ist und sollen die fehlenden Werte berechnen. Die Funktion erhält folgende Parameter:

- Strings a und b
- Länge l eines Präfixes von b
- 2D-Array *mat*, um das Ergebnis der DP-Matrix zu speichern

Die übergebene DP-Matrix *mat* enthält schon den Teil des Ergebnisses bis zur Spalte l (siehe Abb.). **(8 Punkte)**



Es gilt $M \geq |b|, N \geq |a|$ und $l \in [0..|b|]$. Wird für die Länge des Präfixes $l = 0$ übergeben, dann sind keine Einträge vorberechnet.

Function computeDP(a : [char], b : [char], l : int, mat : [[int; M]; N])
{

iii) Entwerfen Sie eine Funktion in Pseudocode für die Suche nach einem Wort mit der geringsten Levenshtein-Distanz im sortierten Wörterbuch `dict` zum Eingabewort `input`, und geben Sie das gefundene Wort zurück! Nutzen Sie die Funktionen aus i) und ii)! Berechnen Sie die Levenshtein-Distanz zwischen der Eingabe und allen Wörtern im Wörterbuch! Machen Sie sich zu Nutze, wenn zwei aufeinanderfolgende Worte im Wörterbuch ein gemeinsames Präfix besitzen! **(12 Punkte)**

```
// l_max: Länge des längsten Wortes im Wörterbuch
Function bestWord(dict : [[char]; N], input : [char], l_max : int) : [char]
{
```