Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn

Klausur Algorithmen und Datenstrukturen

25. Februar 2021, 9:00 -12:00

Name: .....

Matrikel Nr.: .....

Unterschrift: .....

Erst öffnen, wenn die Klausuraufsicht die Erlaubnis erteilt!

- Schreiben Sie Ihren **Namen** und Ihre **Matrikelnummer** an die vorgesehenen Stellen.
- **Unterschreiben Sie** diese Seite um zu bestätigen, dass Sie die Fragen ohne unerlaubte Hilfsmittel beantwortet haben und die Klausuraufsicht über (gesundheitliche) Probleme informiert haben.
- Dies ist eine **Open Book** Klausur, weshalb alle gedruckten und handgeschriebenen Materialien erlaubt sind. **Elektronische Hilfsmittel** sind **nicht** erlaubt.
- Schreiben Sie **lesbar** und nur mit **dokumentenechten** Stiften. Benutzen Sie **keine rote** Farbe und **keinen Bleistift**!
- Es wird **nur eine Lösung pro Aufgabe** gewertet. Vergewissern Sie sich, dass Sie zusätzliche Lösungen durchstreichen, andernfalls wird die schlechteste Lösung gewertet.
- Detaillierte Schritte können Ihnen zu **Teilpunkten** verhelfen, falls das Endergebnis falsch ist.
- Die Schlüsselwörter **Zeigen Sie...**, **Beweisen Sie...**, **Begründen Sie...** oder **Leiten Sie ... her** zeigen an, dass Sie Ihre Antwort sorgfältig und gegebenenfalls formal begründen müssen.
- Die Schlüsselwörter **Geben Sie ... an, Zeichnen Sie ...** zeigen an, dass Sie lediglich die geforderte Antwort oder Zeichnung und keine Begründung liefern müssen (falls nicht explizit verlangt).
- Die folgenden Regeln gelten **überall**, außer sie werden explizit außer Kraft gesetzt.
- Bei Laufzeitfragen ist nur die **asymptotische Laufzeit** notwendig.
- Wenn Sie einen Algorithmus angeben sollen, so können Sie Pseudocode angeben. Eine **ausreichend detaillierte** Beschreibung der Funktionsweise Ihres Algorithmus genügt jedoch.
- Algorithmen aus der Vorlesung **können als Blackbox** verwendet werden.
- **Lesen Sie jede Aufgabe sorgfältig** durch und stellen Sie sicher, dass Sie diese verstanden haben!
- Falls Sie eine Frage zur Aufgabenstellung haben, geben Sie der Klausuraufsicht ein **Handzeichen**.
- Schreiben Sie Ihren Namen auf **alle Blätter**!
- Füllen Sie die **Erklärung über Ihren gesundheitlichen Status** aus.

Aufgabe	1	2	3	4	5	6	7	8	**Total**
Maximum	21	17	18	15	10	14	16	9	120
Punkte									

Aufgabe 1: Kurze Fragen *(21 Punkte)*

(a) Sei $h(x, i) := x + i \mod 7$ eine Hashfunktion mit linearem Sondieren zur Kollisionsauflösung. Fügen Sie die Schlüssel $44, 45, 79, 55, 91, 18$ mittels h in die Tabelle ein. *(3 Punkte)*

0	1	2	3	4	5	6

(b) Geben Sie einen *gewichteten, ungerichteten* Graphen G mit *positiven* Gewichten an und markieren Sie einen Knoten v und einen Shortest Path Tree von v in G, so dass *(3 Punkte)*

- G *kein* Baum ist,
- der minimale Spannbaum von G eindeutig ist und
- ihr markierter Shortest Path Tree dem minimalen Spannbaum von G entspricht.

(c) Gegeben seien zwei Arrays A und B mit $|A| = m$ und $|B| = n$ und $m \leq n$. Die Einträge der Arrays seien natürliche Zahlen. Man möchte herausfinden, ob es eine Zahl gibt, die sowohl in A als auch in B vorkommt. Geben Sie einen möglichst effizienten Algorithmus für dieses Problem an ...

(i) unter der Annahme, dass finden und einfügen in einer Hashtabelle $O(1)$ Zeitschritte benötigt, solange der Load der Hashtabelle $O(1)$ ist. *(4 Punkte)*

(ii) ohne Nutzung von Hashing. *(5 Punkte)*

Analysieren Sie jeweils die Laufzeit als Funktion von m und n.

(d) Gegeben seien zwei Rot-Schwarz Bäume T_1 und T_2 mit gleicher *Schwarztiefe* h. Zudem seien alle Schlüssel in T_1 *echt kleiner* als alle Schlüssel in T_2.

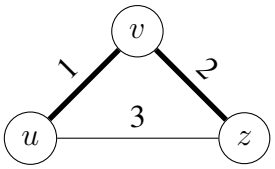
Geben Sie einen Algorithmus an welcher T_1 und T_2 in $\mathcal{O}(h)$ Zeitschritten in einen *gültigen* Rot-Schwarz Baum überführt der *genau* die Schlüssel aus T_1 und T_2 enthält. Erklären Sie kurz die Laufzeit und warum der Algorithmus einen gültigen Rot-Schwarz Baum zurück gibt. *(6 Punkte)*

Musterlösung

(a) keys $i = \{44, 45, 79, 55, 91, 18\}$

$h(44) = 44 \mod 7 = 2$, $h(45) = 45 \mod 7 = 3$, $h(79) = 79 \mod 7 = 2$, aber 2 ist schon mit 44 belegt, nun wird Feld 3 geprüft, welches ebenfalls belegt ist. Entsprechend wird 79 in Feld 4 gespeichert. $h(55) = 55 \mod 7 = 6$, $h(91) = 91 \mod 7 = 0$, $h(18) = 18 \mod 7 = 4$, da 4 besetzt ist gehen wir weiter zu 5.

91	-	44	45	79	18	55
0	1	2	3	4	5	6

(b) 

Die fett gedruckten Kanten zeigen den MST und den Shortest Path Tree von v.

(c) (i) Man alloziiert eine Hashtabelle der Größe n und hasht die Werte von A in diese Tabelle. Danach testet man für jede Zahl in B, ob sie in der Hashtabelle ist. Die Gesamtkosten betragen $O(m+n)$.

(ii) Man sortiert A in Zeit $O(m \log m)$. Nun testet man mittels binärer Suche für jede Zahl in B, ob sie in A liegt. Dies kostet $O(n \log m)$. Insgesamt beträgt die Laufzeit also $O(m \log m + n \log m) = O(n \log m)$.

(d) Zuerst konstruieren wir uns einen Rot-Schwarz Baum T', welcher einen neu erzeugten (schwarzen) Wurzelknoten v bekomtmt. Dieser Wurzelknoten führt T_1 als linken und T_2 als rechten Teilbaum. Also, $\mathsf{root}(T') = v$, $\mathsf{LeftChild}(v) = \mathsf{root}(T_1)$ und $\mathsf{RightChild}(v) = \mathsf{root}(T_2)$. Der Schlüssel von v spielt hierbei keine Rolle. Somit haben wir einen Rot-Schwarz Baum mit folgenden Eigenschaften erstellt (wir ignorieren den Schlüssel von v hier):

- Der Wurzelkoten v ist schwarz.
- Alle Schlüssel links vom Wurzelknoten sind kleiner als die Schlüssel rechts vom Wurzelknoten.
- Da T_1 und T_2 valide Rot-Schwarz Bäume mit Schwarztiefe h sind, hat T' die Schwarztiefe $h+1$.

Nachem wir T' erstellt haben, löschen wir den Wurzelknoten v um nur noch Schlüssel aus T_1 und T_2 zu haben. Daraus ensteht ein Rot-Schwarz Baum T mit den gewünschten eigenschaften.

Da die Löschoperation in $O(h)$ Schritten ausgeführt werden kann, ergibt sich eine Gesamtlaufzeit von $O(h)$.

Aufgabe 2: O-Notation *(17 Punkte)*

(a) Gegeben sei der folgende Pseudocode.

Algorithm 1 `myst-div(`n`)`

1: **while** $n > 1$ **do**
2: $\quad n = n/3$ ▷ *Dieser Schritt kostet eine Zeiteinheit*
3: **return** n

Nehmen Sie an, dass die Laufzeit $T(n)$ der Anzahl der Divisionen in Zeile 2 entspricht. Geben Sie die *exakte* Laufzeit $T(n)$ der Funktion `myst-div(`n`)` an. Zeigen Sie dann anhand der Definition der O-Notation (d.h. ohne Grenzwerte), dass $T(n) \in O(\log_{100} n)$. *(5 Punkte)*

Hinweis: Sie dürfen den Definitionsbereich von `myst-div` *auf die Menge* $\{3^k \mid k \in \mathbb{N}\}$ *beschränken.*

(b) Geben Sie an, ob die folgenden Aussagen wahr oder falsch sind. Beweisen oder widerlegen Sie die Aussagen anhand der Definition der O-Notation oder anhand der Grenzwert-Charakterisierung.

(i) $2\sqrt{n} + \log(n) \in o(n)$ *(4 Punkte)*

(ii) $2^{2n} \in \Theta(2^n)$ *(3 Punkte)*

(iii) $a^n \in \omega(n^k)$, for every real $a > 1$ and integer $k \geq 1$ *(5 Punkte)*

Musterlösung

1. Für $n = 3^k$ führt der Algorithmus genau $k = \log_3 n$ Divisionen aus, d.h. $T(n) = \log_3 n$. Es gilt $\log_3 n = \frac{\log_{100} n}{\log_{100} 3}$, d.h. mit $c = (\log_{100} 3)^{-1}$ und $n_0 = 1$ gilt $\log_3 n \leq c \log_{100} n$ für alle $n \geq n_0$. Also ist $\log_3 n \in O(\log_{100} n)$.

2. (a) Wahr. Mit L'Hospital erhält man

$$\lim_{n\to\infty} \frac{2\sqrt{n} + \log(n)}{n} = \lim_{n\to\infty} \frac{(2\sqrt{n} + \log(n))'}{(n)'} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} + \frac{1}{n \ln 2} = 0 \,.$$

(b) Falsch. Es gilt

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} 2^n = \infty \,.$$

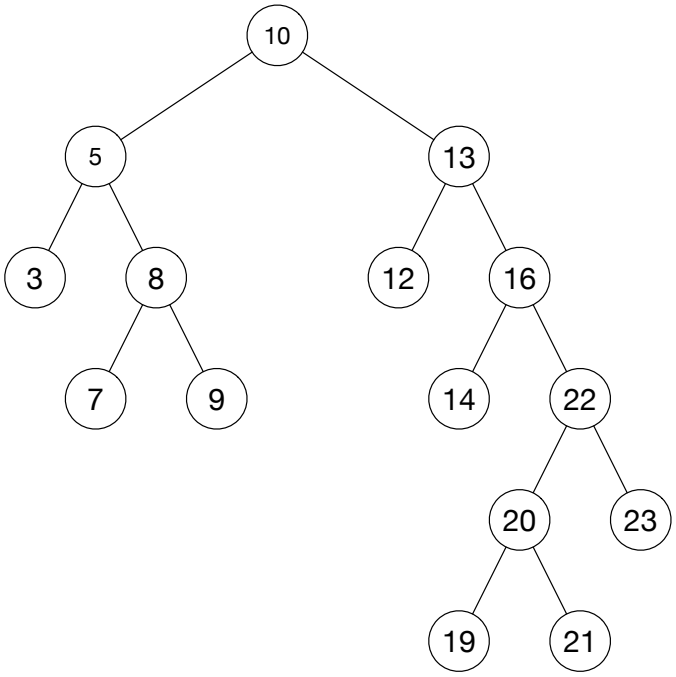
Daraus folgt $2^{2n} \notin O(2^n)$ und deshalb $2^{2n} \notin \Theta(2^n)$.

(c) Wahr. Bezeichne $f^{(k)}$ die k-te Ableitung einer Funktion f. Mit L'Hospital erhält man

$$\lim_{n\to\infty} \frac{a^n}{n^k} = \lim_{n\to\infty} \frac{(a^n)^{(k)}}{(n^k)^{(k)}} = \lim_{n\to\infty} \frac{(\ln a)^k a^n}{k!} = \infty \,.$$

Aufgabe 3: Traversierung Binärer Suchbäume *(18 Punkte)*

(a) Betrachten Sie den folgenden binären Suchbaum. Nummerieren Sie die Knoten entsprechend der Besuchsreihenfolge einer *Pre-Order*, *In-Order* und *Post-Order DFS-Traversierung*, analog zur Vorlesung. *(6 Punkte)*



(b) Zeigen Sie, dass es (wenn man die Suchschlüssel ignoriert) *genau einen* nicht-leeren binären Suchbaum gibt bei dem die Besuchsreihenfolgen der *Pre-Order*, *In-Order* und *Post-Order DFS-Traversierungen* gleich sind. *(5 Punkte)*

Hinweis: Teilpunkte falls Sie diesen Baum angeben, aber nicht die Eindeutigkeit beweisen.

(c) Sei T ein Binärbaum in dem jeder Knoten *entweder* 2 Kindknoten *oder* keine Kindknoten hat. Zudem speichern wir zu jedem *jedem* Knoten v von T einen Wert $\mathsf{Post}(v)$, der die Position von v in der Besuchsreihenfolge einer *Post-Order* DFS-Traversierung enthält.

Geben Sie einen Algorithmus an welcher für einen Knoten v die Größe des *rechten Teilbaumes* von v in *konstant* vielen Zeitschritten bestimmt. Erklären Sie Ihren Algorithmus.

Beispiel: 5 ist die richtige Ausgabe für den Knoten mit Schlüssel 16. *(7 Punkte)*

Musterlösung

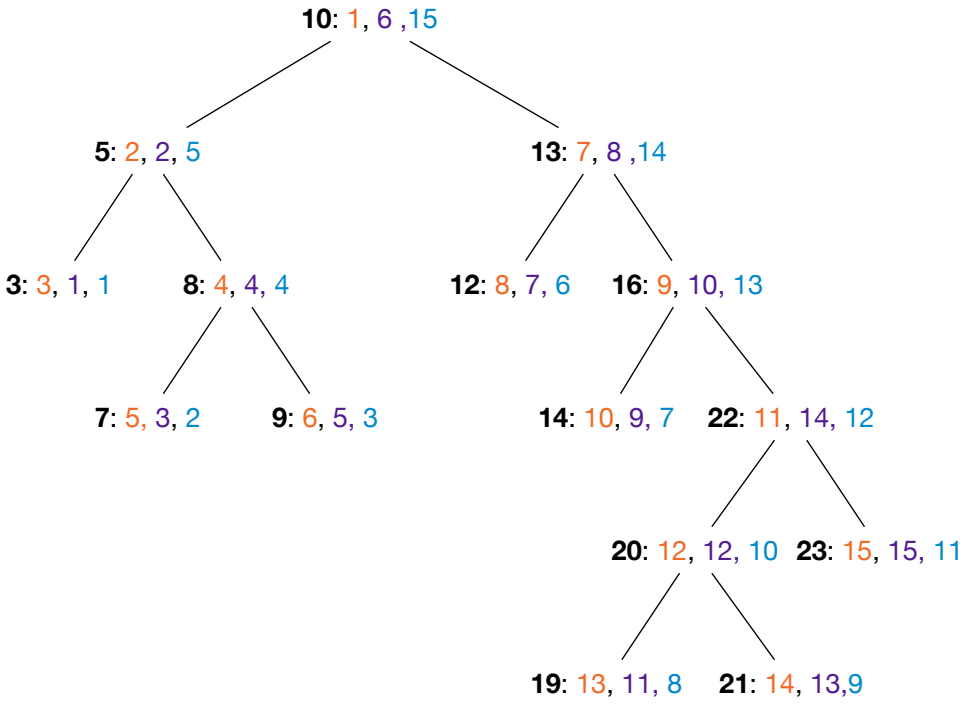


Abbildung 1: Lösungsbaum Teil (a)

(a) Abbildung 1 gibt die Lösung an. Die pre-order, in-order, and post-order Besuchsreihenfolgen sind jeweils in Orange, Lila , und Blau.

(b) Ein binärer Suchbaum mit einem einzigen Knoten erfüllt die Anforderung. Ein binärer Suchbaum mit $n > 1$ Knoten kann diese Anforderung nicht erfüllen da für die Wurzel r gilt $\mathsf{Pre}(r) = 1 \neq n = \mathsf{Post}(r)$.

(c) Wenn v keine Kinder hat, gib 0 aus. Sonst sei u das linke Kind von v. Gib $\mathsf{Post}(v) - \mathsf{Post}(u) - 1$ aus.

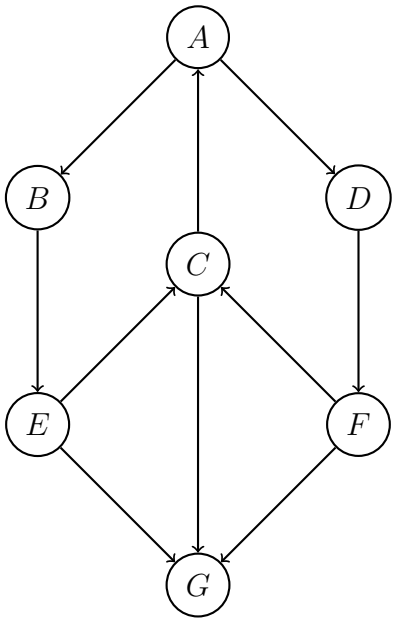
Der Algorithmus ist offensichtlich korrekt wenn v keine Kindknoten hat. Ansonsten basiert die Korrektheit auf der Post-order einer DFS-Traversierung. Das linke Kind u von v bekommt seine Post-order Nummer nachdem alle Knoten im linken Teilbaum von der DFS abgearbeitet sind. Der Knoten v erhält seine Post-order Nummer nachdem der linke *und* der rechte Teilbaum von v abgearbeitet sind. Wenn r die Größe des rechten Teilbaumes ist erhält v seine Nummer $r + 1$ Schritte nachdem u seine Nummer erhält, nämlich nachdem zusätzlich alle r Knoten des rechten Teilbaumes und v selbst abgearbeitet sind. Damit ist

$$\begin{aligned} & \mathsf{Post}(v) = \mathsf{Post}(u) + r + 1 \\ \iff \quad & r = \mathsf{Post}(v) - \mathsf{Post}(u) - 1. \end{aligned}$$

Aufgabe 4: Graph-Traversierung *(15 Punkte)*

(a) Betrachten Sie den unten stehenden Graph. Führen Sie eine *Tiefensuche* auf diesen Graph mit dem Startknoten A aus. Beschriften Sie die Kanten jeweils mit B, V, R beziehungsweise Q wenn es sich um eine Baum-, Vorwärts-, Rückwärts- beziehungsweise Querkante handelt.

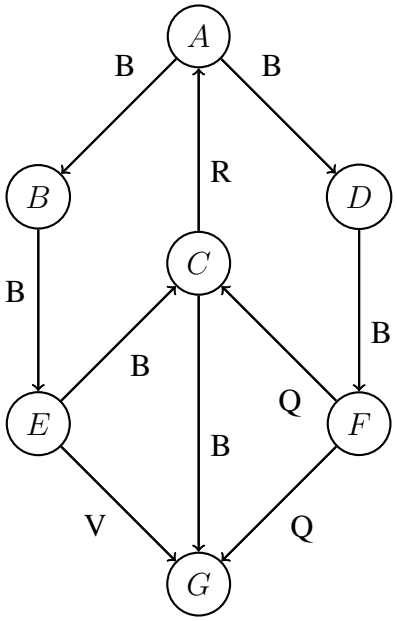
Wichtiger Hinweis: Falls Sie die Wahl zwischen mehreren Knoten haben, wählen Sie als Nachfolgeknoten jeweils den alphabetisch kleinsten. *(6 Punkte)*



(b) Sei $G = (V, E)$ ein *zusammenhängender, ungerichteter* Graph. Sei $u \in V$. Wenn wir eine Tiefensuche mit Startknoten u ausführen erhalten wir einen Baum T. Angenommen eine Breitensuche mit gleichem Startknoten u produziert exakt den gleichen Baum T. Beweisen Sie, dass dann $G = T$ gilt. *(9 Punkte)*

Musterlösung

(a)



(b) Sei T der Baum welcher sowohl von der Breitensuche (BFS) als auch der Tiefensuche (DFS) zurückgegebenen wird. T ist also *sowohl* BFS-Baum *als auch* DFS-Baum, was wir im folgenden ausnutzen.

Formulierungsmöglichkeit A: Für einen Widerspruchsbeweis nehmen wir an, dass G eine Kante $\{x, y\}$ hat die nicht Teil von T ist. Eine Eigenschaft der DFS in ungerichteten Graphen ist, dass entweder x ein Vorgänger von y in T sein muss oder umgekehrt, denn die DFS hat in ungerichteten Graphen nur Baum-kanten oder Rückwärts-kanten. Sei o.B.d.A. x der Vorgänger von y in T (1).

Betrachten wir nun die Eigenschaften von T als BFS-Baum. Aufgrund von (1) können x und y nicht in der gleichen BFS Schicht sein. Da x und y aber eine Kante teilen, sind diese auf jeden Fall in *aufeinanderfolgenden* BFS-Schichten (2). Aus den beiden Eigenschaften (1),(2) folgt, dass die Kante $\{x, y\}$ in T sein müsste, ein Widerspruch.

Formulierungsmöglichkeit B: Sei $\{x, y\} \in E$ eine beliebige Kante von G. In *ungerichteten* Graphen gibt es bzgl. der DFS keine Querkanten. Damit ist entweder x ein Vorgänger von y in T oder umgekehrt (1). Nun ist T aber insbesondere ein BFS-Baum, d.h., x und y sind in aufeinanderfolgenden Schichten in T (2). Aufgrund von (1) und (2) muss $\{x, y\}$ eine Baumkante sein. Damit sind alle Kanten von G Baumkanten und damit muss $G = T$ sein.

Formulierungsmöglichkeit C: Angenommen G ist kein Baum. Damit hat G mindestens einen Zyklus. Sei o.B.d.A. v_1 der Knoten auf einem solchen Zyklus in G der am nähesten zur Wurzel von T ist. Sei $v_1, \ldots, v_k$ für $k \geq 3$ die Folge von Knoten des Zyklus, d.h., $\{v_i, v_{i+1}\} \in E$ und $\{v_k, v_1\} \in E$. Da v_k und v_2 jeweils voneinander erreichbar sind, selbst ohne die Kanten $\{v_1, v_2\}$ und $\{v_1, v_k\}$, enthält T als DFS-Baum nur *entweder* $\{v_1, v_2\}$ *oder* $\{v_1, v_k\}$, der jeweils andere Knoten wird über den Kreis exploriert.

Der Baum T gesehen als BFS-Baum müsste sowohl $\{v_1, v_2\}$ *als auch* $\{v_1, v_k\}$ enthalten (andernfalls gäbe es einen Zyklusknoten der näher an der Wurzel von T ist). Damit kann T

nicht sowohl BFS Baum als auch DFS Baum sein, ein Widerspruch zur Vorraussetzung.

Aufgabe 5: Kürzeste Pfade *(10 Punkte)*

(a) Geben Sie einen *gewichteten, gerichteten* Graphen G und markieren Sie einen Knoten v von G, so dass

- G kein Baum ist,
- es mindestens eine Kante mit negativem Gewicht gibt,
- G keine negativen Kreise hat,
- man durch Anwendung von Dijkstra auf G mit v als Wurzel einen Shortest Path Tree von v in G erhält.

Markieren Sie den Shortest-Path-Tree in G, den Dijkstras' Algorithmus ausgibt. *(4 Punkte)*

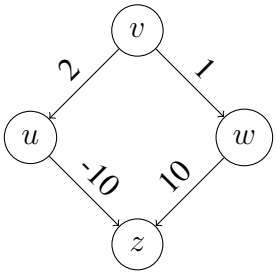
(b) Geben Sie einen *gewichteten, gerichteten* Graphen G und markieren Sie einen Knoten v von G, so dass

- G kein Baum ist,
- es mindestens eine Kante mit negativem Gewicht gibt,
- G keine negativen Kreise hat,
- man durch Anwendung von Dijkstra auf G mit v als Wurzel *keinen* Shortest Path Tree von v in G erhält.

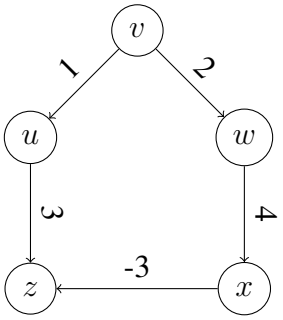
Markieren Sie den Baum in G, den Dijkstras' Algorithmus ausgibt. Markieren Sie den Knoten, für den die berechnete Distanz nicht minimal ist. Geben Sie die kürzeste Distanz von v zu diesem Knoten sowie die von Dijkstra berechnete Distanz. *(6 Punkte)*

Musterlösung

(a)



(b)



Dijkstra would start by putting the edge (v, u) in the shortest path tree. Then (v, w). Then (u, z). But (u, z) should not be in the shortest path tree, since the path (v, w, x, z) is shorter than (v, u, z).

Aufgabe 6: Mystischer Algorithmus *(14 Punkte)*

Betrachten Sie den folgenden Algorithmus, welcher als Input ein Array A der Länge n sowie eine natürliche Zahl m erhält mit der Eigenschaft, dass die Einträge in A natürliche Zahlen zwischen 0 und $m-1$ sind.

Algorithm 2 `myst`

```
1: B = [0] · m                          ▷ create an array of length m with each entry 0.
2: for i = 0 to n − 1 do
3:     B[A[i]] = B[A[i]] + 1
4: ℓ = 0
5: for j = 0 to m − 1 do
6:     if B[j] > 0 then
7:         for k = 0 to B[j] − 1 do
8:             A[ℓ + k] = j
9:     ℓ = ℓ + B[j]
```

(a) Welche Bedeutung hat der Wert $B[j]$ (nach Zeile 3) für ein $j \in \{0, \ldots, m-1\}$? *(3 Punkte)*

(b) Beschreiben Sie in einem kurzen Satz, was `myst` tut. *(3 Punkte)*

(c) Geben Sie die asymptotische Laufzeit von `myst` als Funktion von n und m an und begründen Sie diese. *(4 Punkte)*

(d) Beschreiben Sie die Vor- und Nachteile von `myst` gegenüber anderen Ihnen bekannten Algorithmen, welche die gleiche Funktion haben. *(4 Punkte)*

Musterlösung

(a) Der Wert $B[j]$ entspricht der Anzahl an Vorkommen des Wertes j in A.

(b) `myst` ist ein Sortieralgorithmus (entspricht Counting Sort).

(c) Zeile 1 benötigt Laufzeit $O(m)$, Zeilen 2-3 Laufzeit $O(n)$. Die Schleife in Zeile 5 wird m mal durchlaufen und die innere Schleife (Zeile 7) *insgesamt* n mal. Zeilen 5-9 benötigen also Laufzeit $O(m+n)$. Die Gesamtlaufzeit ist also $O(m+n)$.

(d) Die beste Sortierlaufzeit für allgemeine Zahlbereiche ist $O(n \log n)$ (z.B. Mergesort). Für $m = o(n \log n)$ ist Counting Sort daher besser, für $m = \omega(n \log n)$ (bzw. wenn nichts über eine obere Schranke von m bekannt ist) ist Mergesort besser. Counting Sort hat die gleiche Laufzeit wie Bucket Sort. Allerdings ist Counting Sort in dieser Form nur für das Sortieren von Zahlen geeignet, nicht für Sortieren von allgemeinen Dateien nach Sortierschlüssel.

Aufgabe 7: Dynamische Programmierung *(16 Punkte)*

Gegeben sei eine Folge ganzer Zahlen $S = (s_1, \ldots, s_n)$.

(a) Geben Sie einen Algorithmus an (nach dem Prinzip des Dynamischen Programmierens) der *die Länge* einer *maximalen aufsteigenden Teilfolge* von S in $\mathcal{O}(n^2)$ Zeitschritten ausgibt. Dass heißt, die Länge k einer längsten Folge $(s_{i_1}, \ldots, s_{i_k})$ sodass $s_{i_1} \leq \cdots \leq s_{i_k}$ und $i_1 < \cdots < i_k$. Begründen Sie die Laufzeit. *(10 Punkte)*

Beispiel: Eine max. aufsteigende Teilfolge von $S = (3, 6, 9, 4, 2, 1, 5, 7, 8)$ *ist* $(3, 4, 5, 7, 8)$ *und die richtige Ausgabe für* S *ist somit 5.*

(b) Geben Sie einen Algorithmus an, der *die Länge* einer *maximalen auf- und wieder absteigenden Teilfolge* von S in $\mathcal{O}(n^2)$ Zeit ausgibt. Dass heißt, die Länge k einer längsten Folge $(s_{j_1}, \ldots, s_{j_k})$ sodass $j_1 < \cdots < j_k$ und für ein $\ell \in \{1, \ldots, k\}$ gilt $s_{j_1} \leq \cdots \leq s_{j_\ell} \geq \cdots \geq s_{j_k}$. Begründen Sie die Laufzeit. *(6 Punkte)*

Beispiel: Eine maximale auf- und absteigende Teilfolge von $S = (3, 6, 9, 4, 2, 1, 5, 7, 8)$ *ist* $(3, 6, 9, 4, 2, 1)$ *und die richtige Ausgabe für* S *ist somit 6.*

Musterlösung

(a) Sei $k(j)$ die Länge eine längsten Teilfolge von $S_j := (s_1, \ldots, s_j)$ die mit dem Wert s_j endet. Wir können $k(j)$ rekursiv wie folgt bestimmen. Im Basisfall haben wir $k(1) = 1$. Für $j > 1$ setzen wir

$$k(j) := 1 + \max_{0 \leq \ell < j \text{ und } s_\ell \leq s_j} k(\ell).$$

(Formal müssten wir noch $k(0) = 0$ und $s_0 := -\infty$ setzen aber im Pseudocode unten lösen wir den Randfall anders). Um die längste Teilfolge von $S_j := (s_1, \ldots, s_j)$ die auf den Wert s_j endet zu finden, prüft obige Rekursion die Längen $1+k(\ell)$ für alle passenden $s_\ell \leq s_j$ und wählt diejenige Option die $k(j)$ maximiert. Wir wenden unser Standardvorgehen für diese Rekursion an. Sei dazu `memo`$[j]$ ein Dictionary welches die Werte von $k(j)$ speichert.

Algorithm 3 `subseq(`j`)` ▷ *`memo` sei global gegeben*

```
if j = 1 then return 1                                   ▷ Basisfall
if memo[j] = Null then                                   ▷ Ergebnis liegt noch nicht vor
    k ← 1                                                ▷ Mindestwert von 1
    for ℓ ← 1 to j − 1 do
        if s_ℓ ≤ s_j then
            k ← max(k, 1 + subseq(ℓ))                    ▷ Behalte den besseren Wert
    memo[j] = k                                          ▷ Ergebnis speichern
return memo[j]
```

Das gewünschte Ergebnis kann dann schlussendlich mit folgender Formel bestimmt werden

$$\max_{1 \leq j \leq n} \texttt{subseq}(j).$$

Wenn man die Dauer der rekursiven Unteraufrufe vernachlässigt benötigt die Berechnung von `subseq(`j`)` höchstens $\mathcal{O}(n)$ Zeitschritte. Die *Gesamtanzahl* rekursiver Aufrufe von `subseq(`j`)` ist höchstens n, denn danach liegen alle Werte `subseq(`j`)` für $j = 1, \ldots, n$ in $\texttt{memo}[j]$ vor. Insgesamt also $\mathcal{O}(n^2)$ Zeitschritte.

(b) Wie in Teil (a) berechnen wir die Werte $k(j)$ welcher die Länge einer längsten Teilfolge $(s_{i_1}, \ldots, s_{i_k})$ von S ist, sodass $s_{i_1} \leq \cdots \leq s_{i_k}$ und $i_1 < \cdots < i_k = j$.

Symmetrisch zu Teil (a) berechnen wir die Werte $k'(j)$ welcher die Länge einer längsten Teilfolge $(s_{i_1}, \ldots, s_{i_{k'}})$ von S ist, sodass $s_{i_1} \geq \cdots \geq s_{i_{k'}}$ und $j = i_1 < \cdots < i_{k'}$.

Dies lässt sich einfach bewerkstelligen indem man die Folge S umdreht und den gleichen Algorithmus aus (a) benutzt.

Die Länge der geforderten Sequenz ist dann gegeben durch

$$\max_{j=1 \ldots n} k(j) + k'(j).$$

Die Berechnung der Werte $k(j)$ und $k'(j)$ für alle $j = 1, \ldots, n$ dauert jeweils $\mathcal{O}(n^2)$ wie in (a). Die Berechnung des obigen Maximums dauert lediglich $\mathcal{O}(n)$. Insgesamt also $\mathcal{O}(n^2)$.

Aufgabe 8: Rabin-Karp Algorithmus *(9 Punkte)*

Sei $T = 334710367$ der Text und $P = 103$ das zu suchende Muster der Länge $m = 3$ jeweils gegeben zur Basis $b = 10$. Die Hashfunktion ist der Modulus mit $M = 11$. Sei

$$t_s := T\big[s \ldots (s+m-1)\big] \mod 11$$

der vom Rabin-Karp Algorithmus in Iteration s genutzte Hashwert eines Teilstrings von T.

Geben Sie die fehlenden Werte von t_s in der gegebenen Tabelle an. Setzen Sie eine Markierung ($\times$) wenn die Hashwerte von P und $T\big[s \ldots (s+m-1)\big]$ in Iteration s übereinstimmen. Setzen Sie außerdem eine Markierung wenn in Iteration s das Muster P in T erkannt wird.

Iteration s	0	1	2	3	4	5	6
Hash value t_s	4						
Hash match?	$\times$						
Pattern match?							

Musterlösung

Iteration s	0	1	2	3	4	5	6
Hash value t_s	4	6	9	6	4	3	4
Hash match?	$\times$				$\times$		$\times$
Pattern match?					$\times$		