Algorithmische Mathematik I

Ausgewählte Aufgaben

1. Ersetzen Sie bei der Berechnung der Collatz-Folge die Anweisung `n = 3 * n + 1;` durch

   (a) `n = n + 1;` Zeigen Sie, dass das Programm dann stets terminiert, und geben Sie (mit Hilfe der O-Notation) eine möglichst gute Schranke für die Zahl der Rechenschritte an.

   Die Anweisung `n = n + 1` wird nur auf ungerade Zahlen ausgeführt also muss das Ergebnis gerade sein, also kann die Anweisung maximal jedes zweite mal vorkommen. Beide Anweisungen hintereinander, also `n = (n + 1)/2` oder `n = n/2 + 1`, schrumpfen jedes $n > 2$. Also muss `2` erreicht werden, und `2` führt zu `1` also terminiert das Programm stets.

   Da mindestens jede zweite Anweisung `n = n/2` ist fällt `n` nach zwei Operationen mindestens mit `n = n/2 + 1`. Da man jedes `n` nur $\log_2(n)$ mal durch 2 teilen muss bis es 1 ist, wird also `n = n+1` nicht viel mehr als $\log_2(n)$ mal ausgeführt, also ist der additive Term für große `n` irrelevant. Also ist die Laufzeit in $O(\log n)$.

   (b) `n = n + 3;` Für welche Startwerte terminiert das Verfahren dann? Beweisen Sie die Korrektheit Ihrer Antwort.

   Notation: Sei ein $n \in M_3$ falls es durch 3 teilbar ist.

   Alle $n \in M_3$ werden immer in M_3 bleiben. Ist n ungerade und durch 3 teilbar muss $n + 3$ auch durch 3 teilbar sein.

   Ist n gerade und durch 3 teilbar so muss n durch 6 teilbar sein. Ist ein n durch 6 teilbar gibt es eine Natürliche Zahl m so, dass $n = 3 \cdot 2 \cdot m$, also gilt $\frac{n}{2} = m \cdot 3$. Also ist $\frac{n}{2}$ auch durch 3 teilbar.

   Also wird für ein `n` in M_3 nach jeder Operationen das Ergebnis immernoch in M_3 sein. Insbesondere führt `n = 3` zu `n = 6` und `n = 6` zu `n = 3`.

   Also terminiert das Programm für kein `n` aus M_3.

   Für alle anderen `n` terminiert das Programm. Da kein `n` nicht in M_3 nach einer Operation in M_3 ist. Für `n = n + 3` ist dies offensichtlich.

   Angenommen $\frac{n}{2}$ ist in M_3. Also gibt es ein $k \in \mathbb{N}$ so, das $\frac{n}{2} = 3k$. Natürlich muss $n = 2 \cdot 3k$ gelten also muss n auch in M_3 sein. Also kann die Anweisung `n = n/2` nie eine Zahl nicht in M_3 in eine Zahl in M_3 umwandeln.

Da wieder mindestens jede zweite Operation `n = n/2` ist fällt `n` in zwei Operationen mindestens mit `n = n/2 + 3`, also fällt jedes $n > 6$ nach zwei Operationen. Insebesondere gilt `1 <- 2 <- 4 <- 8 <- 5`. Also werden genau alle Eingaben nicht in M_3 terminieren.

2. Seien n_1 und n_2 zwei natürliche Zahlen mit identischer Ziffernfolge $z_{l-1}z_{l-2}...z_0$ bezüglich unterschiedlicher Basen b_1 und b_2. Welche der folgenden Aussagen sind wahr? Begründen Sie Ihre Antwort!

   (a) Falls $b_1 > b_2$, so ist $n_1 > n_2$.

   Die Aussage stimmt nicht. Zum beispiel mit der Ziffernfolge $z_0 = 1$, $b_1 = 4$ und $b_2 = 3$, dann ist $n_1 = 1 \cdot 4^0 = 1 = 1 \cdot 3^0 = n_2$.

   (b) Falls $n_1 > n_2$, so ist $b_1 > b_2$.

   Die Aussage stimmt. Es gilt ja, mit $j \in \{1, 2\}$, $n_j = \sum_{i=0}^{l-1} z_i b_j^i$. Da alle in der Summe vorkommenden Zahlen nichtnegativ sind und $z_{l-1}z_{l-2}...z_0$ für beide Zahlen gleich sind muss $b_1 > b_2$ gelten damit $n_1 = \sum_{i=0}^{l-1} z_i b_1^i > \sum_{i=0}^{l-1} z_i b_2^i = n_2$ gilt. Der erste Term der Summe ist immer identisch, und jeder folgende Term unterscheidet sich nur in der basis b_j mit der multipliziert wird, falls b_1 nicht größer als b_2 ist kann kein Term von n_1 jemals größer als n_2 sein.

   (c) Falls b_1 Teiler von b_2 ist, so ist n_1 Teiler von n_2.

   Nein, zum Beispiel: mit der Ziffernfolge 11 und $b_1 = 2, b_2 = 10$. b_1 ist ein Teiler von b_2. Jedoch sind $n_1 = 3$ und $n_2 = 11$ und somit teilt n_1 nicht n_2.

   (d) Falls n_1 Teiler von n_2 ist, so ist b_1 Teiler von b_2.

   Nein, zum Beispiel: mit der Ziffernfolge 11 und $b_1 = 7$ und $b_2 = 23$. b_1 ist kein Teiler von b_2. Jedoch sind $n_1 = 7 + 1 = 8$ und $n_2 = 23 + 1 = 24$, also ist n_1 ein Teiler von n_2.

3. Es sei $(a_{l-1}a_{l-2}...a_0)_{-10} := \sum_{i=0}^{l-1} a_i(-10)^i$, wobei $a_i \in \{0, ..., 9\}$ sei für $i \in \{0, ..., l-1\}$. Man nennt $a_{l-1}...a_0$ Darstellung von $\sum_{i=0}^{l-1} a_i(-10)^i$ zur Basis -10, falls $a_{l-1} \neq 0$ gilt oder $l = 1$ und $a_0 = 0$ gelten.

(a) Zeigen Sie, dass es für jede ganze Zahl x eine Darstellung zur Basis -10 gibt.

Erstmal für positive Zahlen. Per induktion, der Fall für 1 ist offensichtlich mit der Zahlenfolge die nur aus a_0 besteht und $a_0 = 1$. Zu zeigen: angenommen n ist darstellbar und positiv, so ist auch $n + 1$ darstellbar.

Ist $a_0 \neq 9$ kann man einfach a_0 um eins incrementieren. Ist $a_0 = 9$ kann man stattdessen a_1 um eins verringern und $a_1 = 0$ setzen, damit ist x um eins incrementiert. Dies geht aber nur falls $a_1 \neq 0$ ist, in diesem fall müsste man a_2 incrementieren und a_1 gleich 9 setzen. Dieser Vorgang wiederholt sich solange alle geraden stellen 9 sind und alle ungeraden 0. Generell falls eine stelle $a_j = 9$ und j gerade ist, ist kann man a_{j+1} um eins verringern und a_j gleich 0 setzen anstatt dass man a_j incrementiert. Mit j ungerade falls $a_j = 0$ ist kann man a_{j+1} incrementieren und $a_j = 9$ setzen, anstatt a_j um eins zu verringern.

Dieser vorgang endet irgendwann außer die Zahl hat die form $(90909...0909)_{-10}$ in diesem Fall wird a_{l-1} zu 0, a_l kommt hinzu und wird 9 und a_{l+1} kommt hinzu und wird 1. So wird z.B. $(90909)_{-10}$ zu $(1909090)_{-10}$ wenn man es um eins incrementiert. Also kann man alle zahlen incrementieren. Das schließt die Induktion, also lassen sich alle positiven Zahlen darstellen.

Jede negative Zahl x mit f stellen (zur basis 10) lässt sich als $x = k - 10^{f+1}$ darstellen, mit $k \in \mathbb{N}$. Stellt man k zur Basis -10 dar, falls $f + 1$ ungerade ist incrementiert man a_{f+1} um eins um x zu erhalten, falls $a_{f+1} = 9$ ist geht man wie eben vor und verringert a_{f+2} um eins etc.

Ist a_{f+1} gerade setzt man es gleich 9 und a_{f+2} gleich 1 um um x zu erhalten. Also lassen sich alle negativen Zahlen erreichen.

0 erreicht man mit $l = 1$ und $a_0 = 0$.

(b) Ist die Darstellung aus Aufgabenteil (a) immer eindeutig?

Angenommen die Darstellung ist nicht eindeutig es müsste unterschiedliche Folgen $a_0 ... a_{l-1}$ und $b_0 ... b_{l-1}$ geben, mit

$$\sum_{i=0}^{l-1} a_i(-10)^i = x = \sum_{i=0}^{l-1} b_i(-10)^i$$

Also muss gelten $\sum_{i=0}^{l-1}(a_i - b_i)(-10)^i = 0$. Angenommen der Term $(a_k - b_k)(-10)^k$ sei der höchste Term ungleich 0. So muss gelten $\sum_{i=0}^{k-1}(a_i - b_i)(-10)^i = -(a_k - b_k)(-10)^k$. Der größtmöglichste absolute Wert für die linke Seite dieser Gleichung ist jedoch kleiner als der kleinstmögliche absolute Wert für die rechte Seite, also kann diese Gleichung niemals erfüllt sein. Somit kann es keine zwei Darstellungen für die Gleiche Zahl geben, und die Darstellung ist immer eindeutig.

4. Zeigen Sie, dass man mit Hilfe von Karatsubas Algorithmus zwei natürliche Zahlen mit k- bzw. l-stelliger Binärdarstellung (mit $k \leq l$) in $O\left(k^{\log_2(3)-1} l\right)$ Zeit multiplizieren kann.

   Wir können schreiben:

$$T(l,k) \leq c \qquad \text{für } l,k \leq 3$$
$$T(l,k) \leq c \cdot l + 3T\left(\left\lceil \frac{l}{2} \right\rceil + 1, \left\lceil \frac{k}{2} \right\rceil + 1\right) \text{für } l,k \geq 4.$$

   Dies muss gelten da es drei rekursive Aufrufe gibt und die zu multipliziereden Zahlen jeweils maximal $\left\lceil \frac{l}{2} \right\rceil + 1$ beziehungsweise $\left\lceil \frac{k}{2} \right\rceil + 1$ Stellen haben.

   Die Rekursion lösen wir wie folgt. Erstmal behaupten wir, dass für alle $m, n \in \mathbb{N} \cup \{0\}$ gilt:

$$T(2^m + 2, 2^n + 2) \leq c \cdot \left(\left(\frac{3}{2}\right)^n \cdot 2^m - 2^{m+1} - 1\right)$$

   Dies kann man per induktion beweisen. Für $m = n = 0$ ist $T(3,3) \leq c$. Für $m, n \in \mathbb{N}$ ist, wieder wegen der obigen Gleichung, $T(2^m + 2, 2^n + 2) \leq c \cdot (2^m + 2) + 3T\left(2^{m-1} + 2, 2^{n-1} + 2\right)$, was man nach der Induktionsvoraussetzung ersetzen kann durch

$$T(2^m + 2, 2^n + 2) \leq c \cdot \left(2^m + 2 + 3\left(\left(\frac{3}{2}\right)^{n-1} \cdot 2^{m-1} - 2^m - 1\right)\right)$$

   und sich vereinfachen lässt zu

$$c \cdot \left(3 \cdot \left(\frac{3}{2}\right)^{n-1} \cdot 2^{m-1} - 2 \cdot 2^m - 1\right).$$

   Jetzt bleibt zu zeigen, dass gilt:

$$c \cdot \left(3 \cdot \left(\frac{3}{2}\right)^{n-1} \cdot 2^{m-1} - 2 \cdot 2^m - 1\right) \leq c\left(\cdot \left(\frac{3}{2}\right)^n \cdot 2^m - 2^{m+1} - 1\right)$$
$$3 \cdot \left(\frac{3}{2}\right)^{n-1} \cdot 2^{m-1} - 2 \cdot 2^m \leq \left(\frac{3}{2}\right)^n \cdot 2^m - 2^{m+1}$$
$$3 \cdot \left(\frac{3}{2}\right)^{n-1} - 2 \cdot 2 \leq \left(\frac{3}{2}\right)^n \cdot 2 - 4$$
$$3 \cdot \left(\frac{3}{2}\right)^{n-1} \leq \left(\frac{3}{2}\right)^n \cdot 2$$
$$3 \leq \left(\frac{3}{2}\right) \cdot 2$$
$$3 \leq 3$$

das schließt die induktion.

Für zwei beliebige $l, k \in \mathbb{N}$ mit $2 < k \leq l$ sei nun $m := \lceil \log_2(l-2) \rceil < 1 + \log_2 l$ und $n := \lceil \log_2(k-2) \rceil < 1 + \log_2 k$. Dann gilt

$$
\begin{aligned}
T(l,k) &\leq T(2^m + 2, 2^n + 2) \\
&\leq c \cdot \left(\left(\frac{3}{2} \right)^n \cdot 2^m - 2^{m+1} - 1 \right) \\
&< c \cdot \left(\left(\frac{3}{2} \right)^n \cdot 2^m \right) \\
&= c \cdot \left(3^n \cdot 2^m \cdot \frac{1}{2^n} \right) \\
&\leq c \cdot \left(3^{\log_2(k)} \cdot l \cdot \frac{1}{k} \right) \\
&= c \cdot \left(k^{\log_2(3)-1} \cdot l \right)
\end{aligned}
$$

5. Berechnen Sie die Kondition der folgenden Funktionen und geben Sie an, wo die Funktions auswertung qualitativ gut bzw. schlecht konditioniert ist.

   (a) $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ mit $f(x) = \sqrt[3]{x}$,

   Die Kondition ist $\frac{\left| \frac{1}{3} x^{-\frac{2}{3}} \cdot x \right|}{\left| x^{\frac{1}{3}} \right|} = \frac{1}{3}$ Also ist sie überall gut konditioniert.

   (b) $f : \mathbb{R} \to \mathbb{R}$ mit $f(x) = a^x = e^{x \ln a}$ für festes $a > 0$,

   Die Kondition ist $\frac{\left| \ln(a) e^{x \ln(a)} \cdot x \right|}{\left| e^{x \ln a} \right|} = |\ln(a) \cdot x|$ Also ist sie nur für absolut kleine Werte für a und x gut konditioniert.

   (c) $f : \mathbb{R} \setminus \{0\} \to \mathbb{R}$ mit $f(x) = \frac{1}{x}$

   Die Kondition ist $\frac{|-x^{-2} \cdot x|}{|x^{-1}|} = \frac{x^{-1}}{x^{-1}} = 1$ Also ist sie überall gut konditioniert.

   (d) $f : \mathbb{R} \to \mathbb{R}$ mit $f(x) = \ln(1 + |x-1|)$

Lassen wir $g(x) = |x-1|$ sein so ist für $x \geq 1$, $g(x) = x-1$ und für $x \leq 1$, $g(x) = 1-x$.

Für $x \geq 1$ ist die Kondition $\frac{\left|\frac{1}{x} \cdot x\right|}{|\ln(x)|} = \frac{1}{\ln(x)}$. Für $x \geq e$ ist die Funktion gut konditioniert. Für $1 \leq x \leq e$ schlecht.

Für $x \leq 1$ ist die kondition $\frac{|x|}{|(2-x)\ln(2-x)|}$. Um 1 dominiert der logarithmus und wird sehr klein deshalb gilt, $\lim_{x \to 1^-} \frac{|x|}{|(2-x)\ln(2-x)|} = \infty$, und die Funktion ist schlecht konditioniert um 1. Für kleinere x werte wird die Funktion wieder gut konditioniert, da $\ln(2-x)$ größer wird. Sehr grob ist die Funktion in etwa für $x < 0.54$ gut konditioniert.

Also it sie überall gut konditioniert außer zwischen ungefähr 0.54 und e.

6. Man kann die Quadratwurzel einer Zahl $a \geq 0$ auch mit dem Newtonverfahren angewandt auf $f(x) = 1 - \frac{a}{x^2}$ berechnen. Dabei können wir uns auf Eingaben a mit $1 \leq a < 4$ und den Startwert $x_0 = 1$ beschränken.

   (a) Wie sieht eine Iteration dieses Verfahrens aus? Berechnen Sie x_1, x_2, x_3 für $a = 3$ und $x_0 = 1$.

   Eine Iteration besteht aus

$$x_{n+1} \leftarrow \frac{x_n}{2}\left(3 - \frac{x_n^2}{a}\right).$$

   Die geforderten Werte sind:
   $x_0 = 1$,
   $x_1 = 0.5\left(3 - \frac{1}{3}\right) = \frac{4}{3}$,
   $x_2 = \frac{\frac{4}{3}}{2}\left(3 - \frac{\left(\frac{4}{3}\right)^2}{3}\right) = \frac{130}{81}$,
   $x_3 = \frac{130}{162}\left(3 - \frac{\left(\frac{130}{81}\right)^2}{3}\right) = 1.7290735\ldots$

   (b) Beweisen Sie, dass auch diese Variante quadratisch konvergiert.

   Zuerst zeige ich, dass sie gegen $\sqrt{a}$ konvergiert. Für $x_n = \sqrt{a}$ gilt $x_n = x_{n+1}$ also divergiert die Folge nicht sobald sie $\sqrt{a}$ einmal erreicht hat.

   Für $x_n < \sqrt{a}$ gilt $\frac{x_n^2}{a} < 1$ also wird nach einer Iteration

$$x_{n+1} \leftarrow x_n + \frac{1}{2}\left(x_n - x_n \cdot \frac{x_n^2}{a}\right).$$

dann $x_{n+1} > x_n$ gelten. Also wird die Folge immer wachsen wenn sie kleiner als $\sqrt{a}$ ist.

Damit die Folge nicht konvergiert müsste $x_n < \sqrt{a} < x_{n+1}$ gelten. Andernfalls muss die Folge für ein $x_0 = 1$ zu $\sqrt{a}$ konvergieren.

Angenommen $x_n < \sqrt{a}$ so können wir schreiben $x_n + \varepsilon = \sqrt{a}$.

Damit dann gelten kann

$$\sqrt{a} < x_{n+1} = \frac{x_n}{2}\left(3 - \frac{x_n^2}{a}\right)$$

$$2\sqrt{a} < 3\left(\sqrt{a} - \varepsilon\right) - \frac{x_n^3}{a}$$

$$0 < -\left(3\varepsilon + \frac{\varepsilon^3}{a} + \frac{3\varepsilon\left(\sqrt{a} + \varepsilon\right)}{\sqrt{a}}\right)$$

dies ist ein Wiederspruch also kann nie $x_n < \sqrt{a} < x_{n+1}$ gelten, also konvergiert die Funktion.

Nun zu zeigen, dass sie quadratisch konvergiert. Sei $x_n = \sqrt{a} + \varepsilon_n$ so gilt, nach etwas umformung $x_{n+1} = \sqrt{a} - \frac{3\varepsilon^3}{2\sqrt{a}} - \frac{\varepsilon^3}{2a}$. Dies kann man in

$$x_{n+1} - \sqrt{a} \le c\left(x_n - \sqrt{a}\right)^2$$

einsetzen und erhält

$$\frac{3}{2\sqrt{a}} + \frac{x - \sqrt{a}}{2a} \le c$$

also gibt es ein c.

7. Beweisen Sie, dass eine Folge natürlicher Zahlen $d_1, \dots, d_n$ mit $d_1 \ge d_2 \ge \cdots \ge d_n$ genau dann die Gradfolge eines einfachen ungerichteten Graphen ist (d.h. es gibt einen Graph G mit $V(G) = \{v_1, \dots, v_n\}$ und $|\delta(v_i)| = d_i$ für alle $i \in \{1, \dots, n\}$), wenn $d_2 - 1, d_3 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n$ dies ist.

   (In der Lösung dieser Aufgabe ist mit Graph ist immer ein einfacher und ungerichteter Graph gemeint).

   Ich zeige zuerst die **Hinrichtung**.

   Falls in einem Graphen die Knoten mit dem größten Grad alle mit v_1 adjazent sind (das heißt es gibt keine zwei Knoten wo der mit dem kleineren Grad mit v_1 adjazent ist, der größere aber nicht), dann kann man v_1, und alle mit v_1 verbundenen Kanten entfernen und erhält einen neuen Graphen mit genau der verlangten Gradfolge. Dies ist der Fall da v_1 entfernt wurden und genau die d_1 Knoten mit dem größten Grad jetzt mit einem Knoten weniger adjazent sind.

Bleibt zu zeigen, dass man jeden Graphen in einen solchen Graphen umwandeln kann ohne die Gradfolge zu verändern. Sei v_s ein mit v_1 adjazenter Knoten, und v_l ein nicht mit v_1 adjazenter Knoten. Desweiteren sei $d_s < d_l$.

- Entferne die Kante zwischen v_s und v_1.
- Entferne eine Kante zwischen v_l und einem Knoten v_r.
- Füge eine Kante zwischen v_l und v_1 hinzu.
- Füge eine Kante zwischen v_s und v_r hinzu.

Man muss beachten, dass v_r nicht gleich v_s oder mit v_s adjazent sein darf, dies ist aber möglich da der Grad von v_l größer als der von v_s ist, es muss also ein v_r geben welches nicht v_s ist oder mit v_s adjazent ist, aber mit v_l adjazent ist.

Dieser Vorgang lässt die Gradfolge unverändert, aber wir sind jetzt ein Schritt näher dran, dass genau die Knoten mit dem großten Grad mit v_1 adjazent sind. Dieser Vorgang kann wiederholt werden, bis dies erreicht ist. Dann kann man v_1 entfernen und erhält einen neuen Graph mit der geforderten Gradfolge.

Die **Rückrichtung** lässt sich deutlich schneller beweisen.

Startet man mit einem Graphen, kann man einen Knoten hinzufügen der mit so vielen der Knoten mit dem größten Grad adjazent ist, dass er jetzt den größten Grad hat. Dieser neue Graph wird auch einfach sein. Und seine Gradfolge entspricht der des ursprünglichen Graphens, nur mit einem neuen Knoten mit größtem Grad v_1, und dass der Grad der d_1 Knoten mit dem nächstgroßen Grad jetzt um eins erhöht ist.

8. Es sei G ein Baum und $A, B \subseteq V(G)$, $A \cap B = \emptyset$, so dass $G[A]$ und $G[B]$ Bäume sind. Zeigen Sie, dass $G[A \cap B]$ ein Baum ist.

   G ist ein Baum, enthält also keine Kreise. Ein Teilgraph von G kann natürlich auch keine Kreise enthalten, also ist $G[A \cap B]$ ein Wald.

   Angenommen $G[A \cap B]$ ist nicht zusammenhängend, es gibt also zwei Knoten $x, y \in V(G[A \cap B])$ ohne einen x-y-Weg zwischen ihnen. Da $G[A]$ und $G[B]$ aber Bäume sind muss es in ihnen jeweils einen solchen Weg geben, es kann aber nur einen x-y-Weg geben, da es sonst einen Kreis in G gäbe. Dieser eine x-y-Weg muss also sowohl in $G[A]$ als auch in $G[B]$ liegen, damit ist er aber in $G[A \cap B]$, ein Wiederspruch, also ist $G[A \cap B]$ zusammenhängend und damit ein Baum.

9. Wir zeigen, dass es keine Kante $\{x,y\} \in E(G)$ geben kann ohne, dass gilt $x \in V(P_{ry})$ oder $y \in V(P_{rx})$.

   Gilt für eine Kante $\{x,y\} \in E(T)$ so ist einer der Knoten näher an r, dieser Knoten muss dann im Weg nach r des anderen sein, da T ein Baum ist.

   Lassen wir x den Knoten aus $\{x,y\}$ sein der während der Tiefensuche als erstes erforscht wurde. Da x und y in G adjazent sind aber nicht in T muss es noch eine weitere Kante aus x geben, die als nächstes erforscht wird. Entweder wir suchen weiter und finden irgendwann einen Weg zu y ohne vorher nochmal x betreten zu haben, in diesem fall muss x natürlich im y-r-Weg liegen. Oder wir haben alles erkundet und müssen zu x zurückkehren und einen weiteren an x adjazenten Knoten ansteuern der nicht y ist, wo sich das Prozedere wiederholt. Oder alle anderen Kanten an x wurden vollständig erforscht und jetzt würde $\{x,y\}$ entlanggegangen werden. Also gibt es entweder einen Wiederspruch oder x ist im y-r -Weg.

10. Wir wollen n Binärstrings der Länge m lexikographisch sortieren.

    Wir betrachten die erste Stelle jedes Strings und sortieren alle Strings so, dass alle Strings mit Eintrag 0 vor allen Strings mit Eintrag 1 stehen. Nach der Sortierung nach der ersten Stelle habe wir quasi zwei disjunkte Mengen:

    Innerhalb jeder dieser Mengen betrachten wir die nächste Stelle und sortieren erneut nach dem gleichen Verfahren wie zuvor.

    Dieses Verfahren wird rekursiv für jede der beiden Teilmengen angewendet. Da die Teilmengen disjunkt sind, müssen wir für jede Stelle nur die verbleibenden Strings innerhalb der jeweiligen Menge prüfen.

    Dies tun wir für jede Stelle, also m mal. Dabei besteht eine Iteration aus maximal n Überprüfungen und Vertauschungen. Also hat der Algorithmus eine Laufzeit von $O(mn)$.

11. Der Algorithmus für Mergesort mit b teilungen:

    Eingabe: eine Menge $S = \{s_1, ..., s_n\}$; eine durch Schlüssel induzierte partielle Ordnung $\preceq$ auf S (durch ein Orakel gegeben). Ein Zahl $b \in \mathbb{N}$.

    Ausgabe: eine Bijektion $f : \{1, ..., n\} \to S$ mit $f(j) \not\preceq f(i)$ für alle $1 \le i < j \le n$.

    if $|S| > 1$

    then sei $m = n \bmod b$ und $S = \dot{\bigcup}_{i=1}^{b} S_i$ mit $|S_i| = \left\lceil \frac{n}{b} \right\rceil$ für $i = 1, ..., m$ und $|S_i| = \left\lfloor \frac{n}{b} \right\rfloor$ für $i = m+1, ..., b$. Sortiere S_1 bis S_b (rekursiv mit b-Mergesort) Verschmelze die Sortierungen von S_1 bis S_b zu Sortierung von S.

(a) Man kann ein binär heap mit dem ersten Element aus jedem der b Teilmengen erstellen. Die komplexität hierfür ist $O(b \log b)$, (Buch Satz 8.21). Jetzt kann man das kleinste Element entfernen und ans Ende einer neuen Liste stellen. Und danach das nächste Element der Liste wo das kleinste Element herkommt, dem heap hinzufügen. Diesen Vorgang kann man n mal wiederholen, dann hat man alle Elemente verschmolzen. In einem binär heap benötigen `extract_min` und `insert`, die Operationen die man benutzt, beide eine Zeit von $\log b$. Also ist die Gesamtzeit $b \log b + 2(n \log b)$ und da $b \leq n$ gilt ist die Zeit in $O(n \log b)$.

(b) Wir können schreiben

$$T(n) \leq bT\left(\frac{n}{b}\right) + cn \log b$$

Per induktion zeigen wir das gilt: $T\left(b^k\right) \leq c(k \log b + 1)b^k$

Für $k = 0$ ist es trivial.

$$\begin{aligned} T\left(b^k\right) &\leq bT\left(b^{k-1}\right) + cb^k \log b \leq b(c(k-1)\log b + 1)b^{k-1} + cb^k \log b \\ &= cb^k(k \log b + 1) + cb^k \log b - cb^k \log b = cb^k(k \log b + 1) \end{aligned}$$

Das schließt die Induktion.

Sei nun $k \coloneqq \lceil \log_b n \rceil < 1 + \log_b n$:

$T(n) \leq T\left(b^k\right) \leq c(k \log b + 1)b^k \leq nc(\log b + \log n + 1) = O(n \log n)$

12. Der Algorithmus der dieses Problem löst sieht wie folgt aus:

Eingabe: ein Digraph G mit Kantengewichten $c : E(G) \to \mathbb{R}_{\geq 0}$, ein Knoten $s \in V(G)$.

Ausgabe: eine Arboreszenz $A \coloneqq (R, T)$ in G ,so dass R alle von s aus erreichbaren Knoten enthält und $\mathrm{dist}_{(G,c)}(s, v) = \mathrm{dist}_{(A,c)}(s, v)$ für alle $v \in R$ gilt.

$R \leftarrow \emptyset, Q \leftarrow \{s\}, l(s) \leftarrow 0$
while $Q \neq \emptyset$ **do**
  Wähle ein $v \in Q$ mit $l(v)$ minimal
  $Q \leftarrow Q \setminus \{v\}$
  $R \leftarrow R \cup \{v\}$
  for $e = (v, w) \in \delta_G^+(v)$ mit $w \notin R$ **do**
    if $w \notin Q$ **or** $c(e) < l(w)$ **then** $l(w) \leftarrow c(e), p(w) \leftarrow e, Q \leftarrow Q \cup \{w\}$
$T \leftarrow \{p(v) \mid v \in R \setminus \{s\}\}$

Sei zu jedem Zeitpunkt des Algorithmus $T \coloneqq \{p(v) \mid v \in (R \cup Q) \setminus \{s\}\}$.

Am Ende jeder Iteration der **while** Schleife gilt:

(a) Für den Knoten $v \in Q$ mit $l(v)$ minimal gilt $p(v)$ ist die kürzeste Kante in $\delta_G^+(R)$.

Für alle Kanten $e \in \delta_G^+(R)$ ist $c(e)$ entweder identisch mit $l(w)$ wobei $w \in Q$ der Knoten ist zu dem e führt. Oder es gibt eine andere Kante zum gleichen Knoten mit geringeren Kosten.

(b) Für alle $v \in R$ ist die längste Kante des s-v-Wegs in T genau so lang wie die längste Kante des besten s-v-Wegs in G.

Nehmen wir an dies gilt nicht, es existiert also eine Kante $k = (a, b)$ im s-v-Weg in T die länger ist als alle Kanten im s-v-Weg in G.

Nennen wir t_b den Zeitpunkt bevor b zu R hinzugefügt wurde. Zu t_b muss $l(b)$ gleich oder kleiner als $l(v)$ für alle $v \in Q$ gewesen sein, und damit ist (wegen **(a)**) auch k einer der kleinsten Kanten in $\delta_G^+(R)$. Da v noch nicht in R ist muss der s-v-Weg in G eine der Kanten aus $\delta_G^+(R)$ benutzen, damit ist k aber nicht mehr länger als alle s -v-Weg in G, ein Widerspruch. Dies hält genau so falls $b = v$ ist.

Aus (b) folgt die korrektheit des Algorithmus.

Die Laufzeit von $O(m \log n)$ kann man analog zu Dijkstras Algorithmus zeigen (Satz 9.13 in Algorithmische Mathematik; Hougardy, Vygen).

Nutzt man für die Menge Q einen Binärheap, wobei v den Schlüssel $l(v)$ hat. Dann haben wir jeweils bis zu n `extract_min`- und `insert-Operationen`, und bis zu m `decrease_key-Operationen`. Die Laufzeit folgt dann mit Satz 8.19 aus dem soeben genannten Buch.