1. Übungsblatt zur Übung

Datenstrukturen und Algorithmen, WS 2020

Abzugeben bis: 5. 11. 2020

Durch dieses Aufgabenblatt sollen Sie mit asymptotischen Schranken vertraut werden.

1. (2 Punkte) Die o-Notation ist wie folgt definiert: $f(n) = o(g(n))$ wenn $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Zeigen Sie, dass $n \log n = o(n^2)$.

   Antwort:

   $$\lim_{n\to\infty} = \frac{n \log n}{n^2} = \lim_{n\to\infty} \frac{\log n \frac{d}{dn} n + n \frac{d}{dn} \log n}{\frac{d}{dn} n^2} = \lim_{n\to\infty} \frac{\log n + 1}{2n} = \lim_{n\to\infty} \frac{\frac{d}{dn} \log n}{\frac{d}{dn} 2n} = \lim_{n\to\infty} \frac{\frac{1}{n}}{2} = 0 \quad (1)$$

2. (4 Punkte) Es seien $f(n)$ und $g(n)$ zwei *asymptotisch nichtnegative* Funktionen. Beweisen Sie, dass $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

   Anmerkung: *Asymptotisch nichtnegativ* bedeutet, dass für alle hinreichend große n, $f(n) \geq 0$ ist.

   Antwort: Wir schreiben zuerst formal auf was die Funktion $\max(f(n), g(n))$ ist. Wir definieren $h(n) = \max(f(n), g(n))$ mit

   $$h(n) = \begin{cases} f(n), & \text{falls } f(n) \geq ng(n), \\ g(n), & \text{falls } f(n) < g(n). \end{cases} \quad (1)$$

   Da $f(n)$ und $g(n)$ asymptotisch nichtnegativ sind, gibt es ein n_0, sodass $f(n) \geq 0$ und $g(n) \geq 0$ $\forall n \geq n_0$. Somit ist für $n \geq n_0$ $f(n) + g(n) \geq f(n) \geq 0$ und $f(n) + g(n) \geq g(n) \geq 0$. Da für jedes n, $h(n)$ entweder $f(n)$ oder $g(n)$ ist, folgt $f(n) + g(n) \geq h(n) \geq 0$. Somit haben wir

   $$h(n) = \max(f(n), g(n)) \leq c_2(f(n) + g(n)) \quad (2)$$

   für alle $n > n_0$ (und $c_2 = 1$). Es gilt für alle $n \geq n_0$

   $$0 \leq f(n) \leq h(n) \text{ und } 0 \leq g(n) \leq h(n) \quad (3)$$

   und somit $0 \leq f(n) + g(n) \leq 2h(n)$ bzw. $0 \leq (f(n) + g(n))/2 \leq h(n)$. Damit zeigen wir, dass

   $$h(n) = \max(f(n), g(n)) \geq c_1(f(n) + g(n)) \quad (4)$$

   für alle $n \geq n_0$ und mit $c_1 = 1/2$.

3. (4 Punkte) Zeigen Sie unter der Verwendung der Θ-Notation, dass $\forall a, b \in \mathbb{R}$ mit $b > 0$, n^b eine asymptotisch exakte Schranke für $(n + a)^b$ ist.

   Antwort: Um zu zeigen, dass $(n+a)^b = \Theta(n^b)$ ist, müssen wir die Konstanten $c_1, c_2, n_0 > 0$ finden, sodass

   $$0 \leq c_1 n^b \leq (n + a)^b \leq c_2 n^b \ \forall n \geq n_0. \quad (1)$$

   Es gilt

   $$\begin{aligned} n + a &\leq n + |a| && (2) \\ &\leq 2n \text{ falls } |a| \leq n && (3) \end{aligned}$$

   und

   $$\begin{aligned} n + a &\geq n - |a| && (4) \\ &\geq 1/2n \text{ falls } |a| \leq 1/2n. && (5) \end{aligned}$$

   Da $b > 0$ ist gilt das auch nach potenzieren mit b:

   $$0 \leq (1/2n)^b \leq (n + a)^b \leq (2n)^b \quad (6)$$

   $$0 \leq (1/2)^b n^b \leq (n + a)^b \leq 2^b n^b. \quad (7)$$

   Somit ist $c_1 = (1/2)^b, c_2 = 2^b$ und $n_0 = 2|a|$.

2. Übungsblatt zur Übung

Datenstrukturen und Algorithmen, WS 2020

Abzugeben bis: 19. 11. 2020

In diesem Aufgabenblatt sollen Sie das Auflösen von rekursiven Zeitgleichungen üben. Zeigen Sie eine möglichst scharfe asymptotische obere Schranke für folgende rekursive Zeitgleichungen durch iteriertes Einsetzen (es gilt jeweils $T(n) = O(1)$ für $n \leq 2$):

1. (2 Punkte) $T(n) = aT(\frac{n}{a}) + n^b$ für $a, b \in \mathbb{N}_{\geq 2}$.

   Antwort: Wir setzen einen Rekursionsschritt tiefer, das heißt $T(\frac{n}{a}) = aT(\frac{n}{a^2}) + (\frac{n}{a})^b$, in die Originalgleichung ein und erhalten:

   $$T(n) = a^2 T\Big(\frac{n}{a^2}\Big) + a\Big(\frac{n}{a}\Big)^b + n^b. \tag{1}$$

   Analog erhalten wir durch weiteres Einsetzen eine allgemeine Formel in Abhängigkeit der Rekursionstiefe k:

   $$\begin{aligned} T(n) &= a^2 T\Big(\frac{n}{a^2}\Big) + a\Big(\frac{n}{a}\Big)^b + n^b && (2)\\ &= a^3 T\Big(\frac{n}{a^3}\Big) + a^2\Big(\frac{n}{a^2}\Big)^b + a\Big(\frac{n}{a}\Big)^b + n^b && (3)\\ &\vdots \\ &= a^k T\Big(\frac{n}{a^k}\Big) + \sum_{i=0}^{k-1} a^i \Big(\frac{n}{a^i}\Big)^b. && (4) \end{aligned}$$

   Die Rekursion ist beendet wenn $T(\frac{n}{a^k}) = T(1)$ ist, also wenn $\frac{n}{a^k} = 1$. Dies ist der Fall für $k = \log_a n$. Wir erhalten:

   $$\begin{aligned} T(n) &= nO(1) + \sum_{i=0}^{log_a(n)-1} a^i \Big(\frac{n}{a^i}\Big)^b && (5)\\ &= O(n) + n^b \sum_{i=0}^{log_a(n)-1} \Big(\frac{1}{a^{b-1}}\Big)^i && (6)\\ &\overset{\text{GR}}{\leq} O(n) + n^b \sum_{i=0}^{\infty} \Big(\frac{1}{a^{b-1}}\Big)^i && (7)\\ &= O(n) + n^b \frac{a^{b-1}}{a^{b-1}-1} = O(n^b). && (8) \end{aligned}$$

2. (4 Punkte) $T(n) = T(n-2) + \log n$

   Antwort: Wir setzen einen Rekursionsschritt tiefer, das heißt $T(n-2) = T(n-4) + \log(n-2)$, in die Originalgleichung ein und erhalten:

   $$T(n) = T(n-4) + \log(n-2) + \log(n). \tag{1}$$

   Analog erhalten wir durch weiteres Einsetzen eine allgemeine Formel in Abhängigkeit der Rekursionstiefe k:

   $$\begin{aligned} T(n) &= T(n-4) + \log(n-2) + \log(n) && (2)\\ &= T(n-6) + \log(n-4) + \log(n-2) + \log(n) && (3)\\ &\vdots \\ &= T(n-2k) + \sum_{i=0}^{k-1} \log(n-2i). && (4) \end{aligned}$$

Nun müssen wir eine Fallunterscheidung zwischen geraden und ungeraden Termen durchführen. Für gerade Terme erhalten wir:

$$T(2n) = T(2n-2k) + \sum_{i=0}^{k-1} \log(2n-2i). \tag{5}$$

Mit $k = n$ erhalten wir eine Lösung unabhängig von k:

$$\begin{aligned} T(2n) &= T(0) + \sum_{i=0}^{n-1} [\log(2) + \log(n-i)] && (6)\\ &= O(1) + n\log(2) + \sum_{i=0}^{n-1} \log(n-i) && (7)\\ &= O(1) + n\log(2) + \log\left(\prod_{i=0}^{n-1} \log(n-i)\right) && (8)\\ &= O(1) + n\log(2) + \log(n!) && (9)\\ &\overset{\text{Stirling}}{\sim} O(1) + n\log(2) + \log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) = \Theta(n\log n) && (10) \end{aligned}$$

Analog erhalten wir für ungerade Terme:

$$T(2n+1) = T(2n-2k+1) + \sum_{i=0}^{k-1} \log(2n-2i+1). \tag{11}$$

Mit $k = n$ und $T(1) = O(1)$ erhalten wir:

$$\begin{aligned} T(2n+1) &= O(1) + \sum_{i=0}^{n-1} \log(2n-2i+1) && (12)\\ &= O(1) + \log\left(\prod_{i=0}^{n-1} (2n-2i+1)\right) && (13)\\ &= O(1) + \log\frac{(2n+1)!}{n!2^n} && (14)\\ &= O(1) + \log\frac{(2n+1)(2n)!}{n!2^n} && (15)\\ &\overset{\text{Stirling}}{\sim} O(1) + \log\frac{(2n+1)\sqrt{2\pi(2n)}\left(\frac{(2n)}{e}\right)^{2n}}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n 2^n} && (16)\\ &= O(1) + \log\left(2^{1/2+n} e^{-n} n^n (2n+1)\right) && (17)\\ &\sim \log(n^n) = n\log n = \Theta(n\log n) && (18) \end{aligned}$$

Somit ist $T(n) = \Theta(n \log n)$.

3. (4 Punkte) $T(n) = T(\sqrt{n}) + \Theta(\log_2 \log_2 n)$

Antwort: Manchmal ist es hilfreich Substitutionen durchzuführen, um die weitere Berechnung zu vereinfachen. Mit $m = \log_2 n$ bzw. $n = 2^m$ erhalten wir:

$$T(2^m) = T(2^{\frac{m}{2}}) + \Theta(\log_2 m). \tag{1}$$

Mit $S(m) = T(2^m)$ bekommen wir folgende rekursive Zeitgleichung:

$$S(m) = S\left(\frac{m}{2}\right) + \Theta(\log_2 m). \tag{2}$$

Wir setzen nun einen Rekursionsschritt tiefer, das heißt $S\left(\frac{m}{2}\right) = S\left(\frac{m}{4}\right) + \Theta\left(\log_2 \frac{m}{2}\right)$, in Gleichung (2) ein und erhalten:

$$S(m) = S\left(\frac{m}{4}\right) + \Theta\left(\log_2 \frac{m}{2}\right) + \Theta(\log_2 m). \tag{3}$$

Analog erhalten wir durch weiteres Einsetzen eine allgemeine Formel in Abhängigkeit der Rekursionstiefe k:

$$S(m) = S\Big(\frac{m}{4}\Big) + \Theta\Big(\log_2 \frac{m}{2}\Big) + \Theta(\log_2 m) \tag{4}$$
$$= S\Big(\frac{m}{8}\Big) + \Theta\Big(\log_2 \frac{m}{4}\Big) + \Theta\Big(\log_2 \frac{m}{2}\Big) + \Theta(\log_2 m) \tag{5}$$
$$\vdots$$
$$= S\Big(\frac{m}{2^k}\Big) + \sum_{i=0}^{k-1} \Theta\Big(\log_2 \frac{m}{2^i}\Big). \tag{6}$$

Die Rekursion verläuft bis zur Abbruchbedingung $m/2^k = 1$, das heißt, bis $S(1)$ auftritt. Wir möchten nun eine Lösung unabhängig von k. Mit $S(1) = S\big(\frac{m}{2^k}\big)$ erhalten wir $k = \log_2 m$. Wir setzen diesen Wert für k nun in Gleichung (4) ein und erhalten:

$$S(m) = O(1) + \sum_{i=0}^{\log_2(m)-1} (\log_2(m) - i) \tag{7}$$
$$= O(1) + \log_2 m \sum_{i=0}^{\log_2(m)-1} 1 - \sum_{i=0}^{\log_2(m)-1} i \tag{8}$$
$$= O(1) + \log_2^2 m - \frac{\log_2(m)(\log_2(m)-1)}{2} = \Theta(\log_2(m)\log_2(m)). \tag{9}$$

Für $T(n) = T(2^m) = S(m)$ erhalten wir somit:

$$T(n) = \Theta(\log_2 \log_2 n \log_2 \log_2 n) = \Theta((\log_2 \log_2 n)^2). \tag{10}$$

3. Übungsblatt zur Übung

Datenstrukturen und Algorithmen, WS 2020

Abzugeben bis: 3. 12. 2020

Sortieralgorithmen

1. (10 Punkte) Gegeben sei eine Stichprobe $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ mit $\mathbf{x}_i \in \Sigma = \{0,1\}^M$ einer M-dimensionalen Verteilungsfunktion $F(\mathbf{x})$. Die Verteilungsfunktion die aus den Daten der Stichprobe vom Umfang N gewonnen wird heißt empirische Verteilungsfunktion $F_N(\mathbf{x})$. Sie gibt für jedes $\mathbf{x}$ die relative Häufigkeit an, mit der Werte der Zufallsvariablen $\mathbf{X}$ in der Stichprobe aufgetreten sind, die dieses $\mathbf{x}$ nicht überschreiten. Wir definieren auf dem Alphabet $(\Sigma, \leq)$, dass ein Vektor $\mathbf{x} = [x_1, x_2, \ldots, x_N]^\intercal$ lexikographisch kleiner als ein Vektor $\mathbf{y} = [y_1, y_2, \ldots, y_N]^\intercal$ ist, das heißt $\mathbf{x}$ liegt in der Sortierung vor $\mathbf{y}$, wenn beim komponentenweisen Vergleich Eintrag für Eintrag, der Eintrag x_i von $\mathbf{x}$ mit dem niedrigsten Index i, in dem sich die beiden Vektoren unterscheiden, echt kleiner ist als der entsprechende Eintrag y_i von $\mathbf{y}$, also $x_i \leq y_i \wedge y_i \not\leq x_i$.

   Ihre Aufgabe ist es nun, die Daten der Stichprobe so aufzubereiten, damit die Berechnung von $F_N(\mathbf{x})$ vereinfacht wird (Sie müssen $F_N(\mathbf{x})$ nicht berechnen). Genauer gesagt, gesucht wird die Menge $\{\mathbf{y}_1, \ldots, \mathbf{y}_{N'}\}_{\neq}$ mit $N' \leq N$ und $\mathbf{y}_1 < \mathbf{y}_2 < \cdots < \mathbf{y}_{N'}$ so dass,

$$\forall i \in \{1, \ldots, N\}\ \exists! j \in \{1, \ldots, N'\} \mid \mathbf{x}_i = \mathbf{y}_j \tag{1}$$

   und ein Vektor $\mathbf{c} \in \mathbb{N}^{N'}$ mit

$$c_j = |\{i \in \{1, \ldots, N\} \mid \mathbf{x}_i = \mathbf{y}_j\}|. \tag{2}$$

   In anderen Worten, die Menge $\{\mathbf{y}_1, \ldots, \mathbf{y}_{N'}\}_{\neq}$ enthält jedes Element aus $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ genau einmal und der j-te Eintrag im Vektor $\mathbf{c}$ gibt an, wie oft $\mathbf{y}_j$ in $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ vorkommt.

   Wir nehmen an, dass N und M sehr groß sind. Weiters nehmen wir an, dass sich die Vektoren $\mathbf{x}$, mit einer Größenordnung von $\approx 2^N$ im Dezimalsystem, nicht als Integer darstellen lassen.

   Gehen Sie wie folgt vor:

   (a) Sortieren Sie $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ aufsteigend in geeigneter Weise und berechnen Sie $\{\mathbf{y}_1, \ldots, \mathbf{y}_{N'}\}_{\neq}$.

   (b) Berechnen Sie auf Grundlage der sortierten Vektoren $\mathbf{x}$ die Elemente von $\mathbf{c}$.

   Ihr Algorithmus sollte möglichst effizient sein. Beschreiben Sie Ihren Algorithmus in Worten und in Pseudocode. Analysieren Sie die Laufzeit des Algorithmus und argumentieren Sie die Korrektheit der Lösung.

   Antwort: Wir werden die Vektoren $\mathbf{x}$ lexikographisch sortieren und im Anschluss die Elemente von $\mathbf{c}$ auf Grundlage der sortierten Vektoren $\mathbf{x}$ berechnen. Da sich die Vektoren $\mathbf{x}$, mit einer Größenordnung von $\approx 2^N$, nicht im Dezimalsystem darstellen lassen, gehen wir wie folgt vor: Wir verwenden einen stabilen Sortieralgorithmus der zuerst nach $\mathbf{x}_{i,1}$ sortiert, dann nach $\mathbf{x}_{i,2}$, usw. Dies entspricht Radixsort mit zwei Fächern. Um ein unnötiges verschieben der Vektoren im Speicher zu vermeiden, werden wir nur die Indizes $I_1, \ldots, I_N$ sortieren. Hierbei ist $\langle I_1, \ldots, I_N \rangle$ eine Permutation von $\langle 1, \ldots, N \rangle$ (anfänglich gilt $I_j = j\ \forall j$). Demnach ist in der i-ten Iteration von Radixsort, $I_1, \ldots, I_N$ nach $\mathbf{x}_{I_1,j}, \ldots, \mathbf{x}_{I_N,j}$ sortiert. Nach dem Sortieren mit Radixsort haben wir die Indizes $I_1, \ldots, I_N$, für die gilt: $\mathbf{x}_{I_1}, \ldots, \mathbf{x}_{I_N}$ ist lexikographisch sortiert. Wir können nun die Menge $\{\mathbf{y}_1, \ldots, \mathbf{y}_{N'}\}_{\neq}$ und den Vektor $\mathbf{c}$ mit einer Laufzeit von $O(NM)$ berechnen. Die Laufzeit unseres Algorithmus ist demnach $T(N, M) = O(NM)$ ($O(NM)$ für das Sortieren und $O(NM)$ für die Berechnung von $\mathbf{c}$). *Anmerkung*: Für einen naiven Algorithmus, in dem man jeden Vektor mit jeden anderen Vektor vergleicht, erhält man eine Laufzeit von $\Theta(N^2 M)$.

6. Übungsblatt zur Übung

Datenstrukturen und Algorithmen, WS 2020

Abzugeben bis: 4. 2. 2021

Binärbäume

1. (10 Punkte) Beschreiben Sie einen Algorithmus der aus symmetrischer und Hauptreihenfolge den entsprechenden Binärbaum eindeutig rekonstruiert (Die symmetrische- und Hauptreihenfolge ist jeweils gegeben durch eine Liste der Indizes der Knoten, wobei jeder Knoten einen eindeutigen Index hat). Argumentieren Sie warum der Algorithmus funktioniert.

   Antwort:

   Algorithmus in Worten: Man nimmt die Elemente des Baumes in der Hauptreihenfolge (HR) und fügt sie der Reihe nach in den (anfangs leeren) Baum ein. Als Sortierschlüssel dient dabei die Position in der symmetrischen Reihenfolge (SR).

   Begründung: Für ein beliebiges Element k in der HR hat diese die Struktur k, $\langle L_k \rangle$, $\langle R_k \rangle$. Dabei ist $\langle L_k \rangle$ ($\langle R_k \rangle$) die Liste der Elemente des linken (rechten) Teilbaumes von k. Fügt man k vor allen Elementen aus L_k und R_k ein so ist garantiert dass k als Wurzel von L_k und R_k auftreten kann.

   Aus der HR allein geht jedoch nicht hervor welche Elemente zu L_k bzw. zu R_k gehören. Dies kann man aus der SR ablesen, die ja für k die Struktur $\langle L_k \rangle$, k, $\langle R_k \rangle$ hat. Das heißt, alle Elemente in der SR deren Position kleiner (größer) als die Position von k ist, gehören in den linken (rechten) Teilbaum von k.

   Alternative: Das erste Element der HR, HR[1] ist die Wurzel des Baumes. Wenn $\mathrm{SR}[i] = \mathrm{HR}[1]$ für Index i, dann ist $\mathrm{SR}[1, \ldots, i-1]$ die SR des linken Teilbaums, $\mathrm{SR}[i+1, \ldots, n]$ die SR des rechten Teilbaums, $\mathrm{HR}[2, \ldots, i]$ die HR des linken, und $\mathrm{HR}[i+1, \ldots, n]$ die HR des rechten Teilbaums (n ist die Anzahl der Knoten im Baum).

   Daraus ergibt sich folgender rekursiver Algorithmus: `BUILD_TREE(SR, HR)` gibt die Wurzel des entsprechenden Baumes zurück und verknüpft die Knoten entsprechend.

   `BUILD_TREE(SR, HR)`:

   1. Sei `n` die Länge von `HR`
   2. Sei `i` $\in \mathbb{N}$ sodass `SR[i] = HR[1]`
   3. `l = BUILD_TREE(SR[1, ..., i - 1], HR[2, ..., i])`
   4. Mache `l` zum linken Sohn von `HR[1]`
   5. `r = BUILD_TREE(SR[i + 1, ..., n], HR[i + 1, ..., n])`
   6. Mache `r` zum rechten Sohn von `HR[1]`
   7. `RETURN HR[1]`