Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithmen und Datenstrukturen
Sommersemester 2026
Musterlösung Übungsblatt 13

Abgabe: Dienstag, 28. Juli, 2026, 10:00 Uhr

Aufgabe 1: Minimale Spannbäume *(5 Bonus Punkte)*

Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph mit paarweise verschiedenen Kantengewichten. Sei T ein minimaler Spannbaum von G.

(a) Sei e eine Kante mit minimalem Gewicht in G. Zeigen Sie, dass es einen minimalen Spannbaum von G gibt, der e enthält. (1 Punkte)

(b) Sei e eine Kante mit maximalem Gewicht auf einem Kreis C in G. Zeigen Sie, dass kein minimaler Spannbaum von G die Kante e enthält. (3 Punkte)

(c) Erklären Sie kurz, welche der beiden Aussagen aus den Teilen (a) und (b) die Korrektheit des Kruskal-Algorithmus unterstützt. (1 Punkt)

Musterlösung

(a) Sei $e = \{u, v\}$ eine Kante mit minimalem Gewicht in G. Sei T ein beliebiger minimaler Spannbaum von G. Falls $e \in T$ gilt, ist die Aussage bereits gezeigt. Nehmen wir also an, dass $e \notin T$ gilt. Wenn wir e zu T hinzufügen, entsteht genau ein Kreis. Dieser Kreis besteht aus e und dem eindeutigen Pfad von u nach v in T. Sei f eine beliebige andere Kante auf diesem Kreis. Da e die Kante mit minimalem Gewicht in G ist und alle Kantengewichte paarweise verschieden sind, gilt

$$w(e) < w(f).$$

Wir entfernen f aus dem Kreis und erhalten den Spannbaum

$$T' = T \cup \{e\} \setminus \{f\}.$$

Für dessen Gesamtgewicht gilt

$$w(T') = w(T) + w(e) - w(f) < w(T).$$

Dies widerspricht der Annahme, dass T ein minimaler Spannbaum ist. Daher muss e in jedem minimalen Spannbaum enthalten sein. Insbesondere gibt es einen minimalen Spannbaum, der e enthält.

(b) Sei e die Kante mit maximalem Gewicht auf einem Kreis C. Sei T ein minimaler Spannbaum. Falls $e \notin T$ gilt, ist die Aussage bereits gezeigt. Nehmen wir also an, dass $e \in T$ gilt. Durch das Entfernen von e zerfällt T in zwei Zusammenhangskomponenten mit Knotenmengen A und B. Da e auf dem Kreis C liegt, enthält C noch mindestens eine weitere Kante f, die einen Knoten aus A mit einem Knoten aus B verbindet. Da e die eindeutig schwerste Kante auf C ist, gilt

$$w(f) < w(e).$$

Der Graph

$$T' = T \setminus \{e\} \cup \{f\}$$

ist wieder ein Spannbaum. Für sein Gesamtgewicht gilt

$$w(T') = w(T) - w(e) + w(f) < w(T).$$

Dies widerspricht erneut der Minimalität von T. Daher kann e in keinem minimalen Spannbaum enthalten sein. Insbesondere gibt es einen minimalen Spannbaum, der e nicht enthält.

(c) Beide Aussagen unterstützen die Korrektheit des Kruskal-Algorithmus. Aussage (a) erklärt, weshalb eine leichteste geeignete Kante sicher zu einer optimalen Lösung hinzugefügt werden kann. Nach dem Zusammenfassen der bereits aufgebauten Zusammenhangskomponenten kann die Aussage auf die jeweils nächste von Kruskal gewählte Kante angewendet werden. Aussage (b) erklärt, weshalb eine Kante verworfen werden kann, wenn sie beim Einfügen einen Kreis erzeugen würde. Da Kruskal die Kanten nach aufsteigendem Gewicht betrachtet, ist diese Kante die schwerste Kante des entstehenden Kreises und muss daher nicht in einem minimalen Spannbaum enthalten sein.

Aufgabe 2: Insertion Sort und Inversionen *(7 Bonus Punkte)*

Sei $A[1, \ldots, n]$ ein Array mit paarweise verschiedenen Zahlen. Eine *Inversion* ist ein Paar von Indizes (i, j) mit

$$i < j \qquad \text{und} \qquad A[i] > A[j].$$

Mit $I(A)$ bezeichnen wir die Anzahl der Inversionen in A. Betrachten Sie eine Variante von Insertion Sort, bei der ein Element so lange mit seinem linken Nachbarn vertauscht wird, bis es seine korrekte Position erreicht hat.

(a) Zeigen Sie, dass das Vertauschen zweier benachbarter Elemente, die eine Inversion bilden, die Anzahl der Inversionen um genau eins reduziert. (2 Punkte)

(b) Zeigen Sie, dass jeder von Insertion Sort ausgeführte Tausch zwei Elemente vertauscht, die eine Inversion bilden. (2 Punkte)

(c) Folgern Sie daraus, dass Insertion Sort genau $I(A)$ Vertauschungen ausführt. (1 Punkt)

(d) Geben Sie eine asymptotische Schranke für die Laufzeit von Insertion Sort in Abhängigkeit von n und $I(A)$ an. Begründen Sie Ihre Antwort kurz. (2 Punkte)

Musterlösung

(a) Seien $A[i]$ und $A[i+1]$ zwei benachbarte Elemente, die eine Inversion bilden. Es gilt also

$$A[i] > A[i+1].$$

Durch das Vertauschen dieser beiden Elemente verschwindet genau die Inversion $(i, i+1)$. Für jedes andere Element $A[k]$ mit $k < i$ liegt $A[k]$ sowohl vor dem Vertauschen als auch danach links von beiden Elementen. Die Inversionsbeziehungen zwischen $A[k]$ und den beiden vertauschten Elementen ändern sich daher nicht. Analog liegt jedes Element $A[k]$ mit $k > i+1$ sowohl vorher als auch nachher rechts von beiden Elementen. Auch hier ändern sich keine Inversionsbeziehungen. Somit wird genau eine Inversion entfernt und keine andere Inversion erzeugt oder entfernt. Daher sinkt die Anzahl der Inversionen um genau eins.

(b) In der betrachteten Variante von Insertion Sort wird ein Element genau dann mit seinem linken Nachbarn vertauscht, wenn es kleiner als dieser Nachbar ist. Unmittelbar vor dem Tausch gilt daher für die benachbarten Positionen $j-1$ und j:

$$A[j-1] > A[j].$$

Somit bilden die beiden Elemente eine Inversion. Jeder von Insertion Sort ausgeführte Tausch vertauscht also zwei Elemente, die eine Inversion bilden.

(c) Zu Beginn besitzt das Array $I(A)$ Inversionen. Nach Teil (a) reduziert jeder von Insertion Sort ausgeführte Tausch die Anzahl der Inversionen um genau eins. Am Ende ist das Array sortiert und besitzt daher keine Inversionen mehr. Somit müssen genau $I(A)$ Inversionen entfernt worden sein. Folglich führt Insertion Sort genau

$$\boxed{I(A)}$$

Vertauschungen aus.

(d) Die äußere Schleife verarbeitet jedes der n Elemente höchstens einmal und verursacht daher einen Aufwand von $\Theta(n)$. Zusätzlich führt der Algorithmus nach Teil (c) genau $I(A)$ Vertauschungen aus. Jede Vertauschung sowie der zugehörige Vergleich benötigen konstante Zeit. Daher beträgt die Laufzeit

$$\boxed{\Theta(n + I(A))}.$$

Insbesondere ergibt sich für ein bereits sortiertes Array mit $I(A) = 0$ eine Laufzeit von $\Theta(n)$. Für ein umgekehrt sortiertes Array gilt

$$I(A) = \binom{n}{2} = \Theta(n^2),$$

sodass die Laufzeit in diesem Fall $\Theta(n^2)$ beträgt.

Aufgabe 3: Asymptotisches Wachstum *(6 Punkte)*

Nehmen Sie an, dass alle Logarithmen zur Basis 2 sind.

(a) Sortieren Sie die folgenden Funktionen nach ihrem asymptotischen Wachstum. Falls zwei Funktionen asymptotisch gleich schnell wachsen, kennzeichnen Sie dies ausdrücklich. Beweise sind nicht erforderlich.

$$f_1(n) = n^{3/2}, \quad f_2(n) = 2^{\log n + \sqrt{\log n}}, \quad f_3(n) = (\log n)^{10},$$
$$f_4(n) = n \log n, \quad f_5(n) = 4^{\log n}.$$

(3 Punkte)

(b) Beweisen oder widerlegen Sie mithilfe der Definition der Landau-Notation:

$$n(\log n)^2 \in O\left(n^{4/3}\right).$$

(3 Punkte)

Musterlösung

(a) Die gegebenen Funktionen lauten

$$f_1(n) = n^{3/2}, \quad f_2(n) = 2^{\log n + \sqrt{\log n}}, \quad f_3(n) = (\log n)^{10},$$
$$f_4(n) = n \log n, \quad f_5(n) = 4^{\log n}.$$

Zunächst vereinfachen wir f_2 und f_5:

$$f_2(n) = 2^{\log n} \cdot 2^{\sqrt{\log n}} = n \cdot 2^{\sqrt{\log n}},$$

und

$$f_5(n) = 4^{\log n} = \left(2^2\right)^{\log n} = n^2.$$

Der Faktor $2^{\sqrt{\log n}}$ wächst schneller als jede feste Potenz von $\log n$, aber langsamer als jede feste positive Potenz von n. Daher gilt

$$(\log n)^{10} \in o(n \log n),$$
$$n \log n \in o\left(n \cdot 2^{\sqrt{\log n}}\right),$$
$$n \cdot 2^{\sqrt{\log n}} \in o\left(n^{3/2}\right),$$

und

$$n^{3/2} \in o(n^2).$$

Die gesuchte Reihenfolge ist somit

$$\boxed{f_3 \ < \ f_4 \ < \ f_2 \ < \ f_1 \ < \ f_5}.$$

Hier bezeichnet $<$ jeweils ein echt langsameres asymptotisches Wachstum.

(b) Zu zeigen ist

$$n(\log n)^2 \in O\left(n^{4/3}\right).$$

Nach Definition der O-Notation müssen Konstanten $c > 0$ und n_0 existieren, sodass für alle $n \geq n_0$ gilt:

$$n(\log n)^2 \leq cn^{4/3}.$$

Dies ist äquivalent zu

$$(\log n)^2 \leq cn^{1/3}.$$

Wir wählen

$$c = 1 \qquad \text{und} \qquad n_0 = 2^{36}.$$

daraus ergibt sich

$$(\log n)^2 = 36^2 \ \texttt{und}\ n^{1/3} = 2^{36/3} = 2^{12} = 2^{6\cdot 2} = 64^2$$

Nun gilt noch zu argumentieren, dass $n^{1/3}$ auch für alle größeren n größer ist, dafür substituieren wir

$$x = \log n.$$

Für $n \geq 2^{36}$ gilt $x \geq 36$. Außerdem gilt für $x \geq 36$

$$x^2 \leq 2^{x/3}.$$

Somit folgt

$$(\log n)^2 = x^2 \leq 2^{x/3} = 2^{(\log n)/3} = n^{1/3}.$$

Durch Multiplikation mit n erhalten wir

$$n(\log n)^2 \leq n \cdot n^{1/3} = n^{4/3}.$$

Mit $c = 1$ und $n_0 = 2^{36}$ ist die Definition erfüllt. Daher gilt

$$\boxed{n(\log n)^2 \in O\left(n^{4/3}\right)}.$$

Aufgabe 4: Hashing *(7 Punkte)*

Wir speichern Daten in einer Hashtabelle der Größe $m = 11$. Kollisionen werden durch offene Adressierung mit Double Hashing aufgelöst. Die Sondierungsfunktion ist

$$h(x, j) = \big(h_1(x) + j \cdot h_2(x)\big) \bmod 11,$$

wobei

$$h_1(x) = x \bmod 11 \qquad \text{und} \qquad h_2(x) = 1 + (x \bmod 10).$$

Die Schlüssel

$$22,\ 41,\ 53,\ 46,\ 30,\ 13$$

werden in dieser Reihenfolge in eine anfangs leere Hashtabelle eingefügt.

(a) Fügen Sie alle Schlüssel in die Hashtabelle ein. Geben Sie für jeden Schlüssel alle Positionen an, die überprüft werden, bevor der Schlüssel eingefügt wird. (3 Punkte)

(b) Geben Sie den endgültigen Inhalt der Hashtabelle an. (1 Punkt)

(c) Erklären Sie, weshalb die Sondierungsfolge jedes Schlüssels jede Tabellenposition besucht, bevor sich eine Position wiederholt. (2 Punkte)

(d) Wie groß ist die Worst-Case-Laufzeit einer erfolglosen Suche in dieser Hashtabelle? Geben Sie Ihre Antwort für eine allgemeine Tabellengröße m in asymptotischer Notation an. (1 Punkt)

Musterlösung

Wir beginnen bei jeder Einfügung mit $j = 0$.

(a) Die Schlüssel werden in der gegebenen Reihenfolge eingefügt. **Schlüssel** 22**:**

$$h_1(22) = 0, \qquad h_2(22) = 3.$$

Die Position 0 ist frei. Die überprüfte Position ist

$$\boxed{0}.$$

Der Schlüssel 22 wird an Position 0 eingefügt. **Schlüssel** 41**:**

$$h_1(41) = 8, \qquad h_2(41) = 2.$$

Die Position 8 ist frei. Die überprüfte Position ist

$$\boxed{8}.$$

Der Schlüssel 41 wird an Position 8 eingefügt. **Schlüssel** 53**:**

$$h_1(53) = 9, \qquad h_2(53) = 4.$$

Die Position 9 ist frei. Die überprüfte Position ist

$$\boxed{9}.$$

Der Schlüssel 53 wird an Position 9 eingefügt. **Schlüssel** 46**:**

$$h_1(46) = 2, \qquad h_2(46) = 7.$$

Die Position 2 ist frei. Die überprüfte Position ist

$$\boxed{2}.$$

Der Schlüssel 46 wird an Position 2 eingefügt. **Schlüssel** 30**:**

$$h_1(30) = 8, \qquad h_2(30) = 1.$$

Für $j = 0$ erhalten wir

$$h(30, 0) = 8.$$

Diese Position ist durch 41 belegt. Für $j = 1$ erhalten wir

$$h(30, 1) = 9.$$

Diese Position ist durch 53 belegt. Für $j = 2$ erhalten wir

$$h(30, 2) = 10.$$

Diese Position ist frei. Die überprüften Positionen sind daher

$$\boxed{8,\ 9,\ 10}.$$

Der Schlüssel 30 wird an Position 10 eingefügt. **Schlüssel** 13**:**

$$h_1(13) = 2, \qquad h_2(13) = 4.$$

Für $j = 0$ erhalten wir

$$h(13, 0) = 2.$$

Diese Position ist durch 46 belegt. Für $j = 1$ erhalten wir

$$h(13, 1) = (2 + 4) \bmod 11 = 6.$$

Diese Position ist frei. Die überprüften Positionen sind daher

$$\boxed{2,\ 6}.$$

Der Schlüssel 13 wird an Position 6 eingefügt.

(b) Der endgültige Inhalt der Hashtabelle ist

Position	0	1	2	3	4	5	6	7	8	9	10
Inhalt	22	–	46	–	–	–	13	–	41	53	30

Dabei bezeichnet "–" eine leere Tabellenposition.

(c) Für jeden Schlüssel gilt

$$h_2(x) \in \{1, 2, \ldots, 10\}.$$

Da 11 eine Primzahl ist, gilt für jeden solchen Wert

$$\mathrm{ggT}(h_2(x), 11) = 1.$$

Angenommen, zwei Sondierungsschritte $j_1 < j_2$ führten zur gleichen Tabellenposition. Dann wäre

$$h_1(x) + j_1 h_2(x) \equiv h_1(x) + j_2 h_2(x) \pmod{11}.$$

Daraus folgt

$$(j_2 - j_1) h_2(x) \equiv 0 \pmod{11}.$$

Da $h_2(x)$ modulo 11 invertierbar ist, folgt

$$j_2 - j_1 \equiv 0 \pmod{11}.$$

Innerhalb der ersten 11 Sondierungsschritte ist dies nur möglich, wenn

$$j_2 - j_1 = 11.$$

Somit wiederholt sich innerhalb der ersten 11 Schritte keine Position. Da die Tabelle genau 11 Positionen besitzt, werden alle Positionen genau einmal besucht.

(d) Bei einer erfolglosen Suche werden die Positionen der Sondierungsfolge überprüft, bis entweder eine leere Position gefunden wurde oder alle m Tabellenpositionen untersucht wurden. Im schlechtesten Fall müssen daher m Positionen überprüft werden. Die Worst-Case-Laufzeit beträgt somit

$$\boxed{\Theta(m)}.$$

Für die konkrete Tabelle mit $m = 11$ sind dies höchstens 11 Sondierungsschritte.