DAP2 – Heimübung 9

Ausgabedatum: 9.6.17 — Abgabedatum: Montag 19.6. für alle Gruppen, 12 Uhr

Schreiben Sie unbedingt immer Ihren **vollständigen Namen**, **Ihre Matrikelnummer** und **Ihre Gruppennummer** auf Ihre Abgaben!

Aufgabe 9.1 (5 Punkte): (Dynamische Programmierung)

Es seien $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ und $b = (b_1, \ldots, b_m) \in \{0,1\}^m$ zwei Bitstrings der Längen n und m. Um den Abstand der beiden Bitstrings zu bestimmen, werden Einfüge-, Lösch- und Ersetzungsoperationen betrachtet, die einen String in den anderen umwandeln. Diese Operationen haben nun unterschiedlichen Kostenaufwand: Eine 0 hinzuzufügen oder zu entfernen ist kostenlos, eine 1 durch eine 0 zu ersetzen oder umgekehrt kostet 2 Einheiten. Beispielsweise kostet es also 2 Einheiten, $a = (1,1,1,0)$ in $b = (1,0,1)$ umzuwandeln, indem man a_2 durch 0 ersetzt (Kosten 2) und $a_4 = 0$ löscht (Kosten 0).

a) Welches ist der minimale Kostenaufwand, um $c = (0,1,1,0)$ in $b = (1,0,1)$ umzuwandeln?

b) Geben Sie eine rekursive Form $K(i,j)$ zur Berechnung des minimalen Kostenaufwands an, um einen Bitstring $a = (a_1, \ldots, a_i)$ in einen Bitstring $b = (b_1, \ldots, b_j)$ umzuwandeln.

c) Zeigen Sie die Korrektheit der in Teilaufgabe b) angegebenen rekursiven Form für einen beliebigen String $a = (a_1, \ldots, a_i)$ und den Fall, dass $b = (b_1, \ldots, b_j)$ mit $b_k = 0$ für alle k, $1 \leq k \leq j$, ist.

Lösung:

a) Der String c lässt sich kostenlos in b umzuwandeln, indem man $c_4 = 0$ löscht, zwischen c_2 und c_3 eine 0 hinzufügt und c_1 löscht.

b) Wir beobachten zunächt, dass es 2 Einheiten kostet, eine 1 zu löschen oder hinzuzufügen, indem man sie in eine 0 umwandelt und dann entfernt bzw. eine 0 hinzufügt und dann durch 1 ersetzt.

Die jeweils aktuellen Elemente a_i und b_j werden verglichen und abhängig davon, was für die übrigen Teilstrings optimal ist, beide Elemente vereinheitlicht (ggf. durch Ersetzen), a_i entfernt oder b_j an a angehängt, so dass die Teilstrings $(a_1, \ldots, a_{i-1})$ und $(b_1, \ldots, b_{j-1})$ bzw. $(a_1, \ldots, a_{i-1})$ und $(b_1, \ldots, b_j)$ bzw. $(a_1, \ldots, a_i)$ und $(b_1, \ldots, b_{j-1})$ betrachtet werden können. Beachte, dass alle zusätzlichen Operationen noch teurer sind. Falls einer der beiden Strings leer ist, kann nur noch in dem anderen String ein Element gelöscht werden. Die Rekursion bricht ab, sobald beide Strings leer sind.

Es ergibt sich folgende rekursive Form

$$K(i,j) = \begin{cases} 0 & \text{falls } i = j = 0 \\ K(i, j-1) & \text{falls } i = 0, j > 0, b_j = 0 \\ K(i, j-1) + 2 & \text{falls } i = 0, j > 0, b_j = 1 \\ K(i-1, j) & \text{falls } i > 0, j = 0, a_i = 0 \\ K(i-1, j) + 2 & \text{falls } i > 0, j = 0, a_i = 1 \\ \min\{K(i-1, j-1), K(i-1, j), & \\ \quad K(i, j-1)\} & \text{falls } i > 0, j > 0, a_i = 0, b_j = 0 \\ \min\{K(i-1, j-1) + 2, K(i-1, j), & \\ \quad K(i, j-1) + 2\} & \text{falls } i > 0, j > 0, a_i = 0, b_j = 1 \\ \min\{K(i-1, j-1) + 2, K(i-1, j) + 2, & \\ \quad K(i, j-1)\} & \text{falls } i > 0, j > 0, a_i = 1, b_j = 0 \\ \min\{K(i-1, j-1), K(i-1, j) + 2, & \\ \quad K(i, j-1) + 2\} & \text{falls } i > 0, j > 0, a_i = 1, b_j = 1. \end{cases}$$

Als zusätzliche Übung kann man sich natürlich auch hierzu ein dynamisches Programm überlegen, das sich die rekursive Form oben zu Nutze macht, und eine Laufzeit in $\mathcal{O}(n \cdot m)$ hat.

c) *Behauptung:* Die rekursive Form $K(i, j)$ berechnet den minimalen Kostenaufwand, um einen Bitstring $a = (a_1, \ldots, a_i)$ in $b = (0, \ldots, 0)$ umzuwandeln.

Beweis: Wir zeigen dies durch eine verschachtelte Induktion über i und $j = |b|$.

Induktionsanfang Falls $i = j = 0$ ist, sind beide (leeren) Strings gleich, der Kostenaufwand ist also $K(0, 0) = 0$.

Induktionsvoraussetzung Betrachte feste i und j. Es gelte die Behauptung für alle $i_0 < i$ und $j_0 < j$ sowie für $i_0 = i$ und $j_0 < j$ (falls $j \geq 1$) sowie für $i_0 < i$ und $j_0 = j$ (falls $i \geq 1$).

Induktionsschritt Falls $i = 0$ und $j > 0$ gelten, lässt sich der String bestehend aus $j - 1$ Nullen durch Hinzufügen einer 0 in b umwandeln. Alle anderen Operationen sind teurer (0 ist minimal) und beziehen sich ebenfalls auf das vorhergie Optimum um a in den String bestehend aus $j - 1$ Nullen umzuwandeln. Da dafür $K(0, j-1)$ nach Induktionsvoraussetzung optimal ist, ist $K(0, j) = K(0, j-1)$, was für $b_j = 0$ richtig berechnet wird.

Falls $i > 0$ und $j = 0$ gelten, kann analog nur ein Element aus a gelöscht werden. Abhängig davon, ob $a_i = 0$ oder $a_i = 1$ ist, ist das Optimum $K(i-1, 0)$ (nach I.V.) plus 0 bzw. 2.

Falls nun $i > 0$ und $j > 0$ gelten, gibt es zwei Fälle: $a_i = 0$ oder $a_i = 1$. Angenommen, $a_i = 0$ und $K(i, j)$ nicht optimal. Dann gibt es Kosten $v < K(i, j)$, mit denen sich a in b umwandeln lässt. Es gibt drei Möglichkeiten, die während dieser Umwandlung auftreten können:

- Das Zeichen a_i wurde nicht gelöscht und entspricht $b_j = 0$. Dann gab es keinen Kostenaufwand, sodass $v - 0 < K(i-1, j-1)$ die minimalen Kosten sind,

um $(a_1, \ldots, a_{i-1})$ in $(b_1, \ldots, b_{j-1})$ umzuwandeln. Das widerspricht allerdings der Induktionsvoraussetzung, dass $K(i-1, j-1)$ optimal ist.

- Das Zeichen a_i wurde gelöscht. Dann gab es ebenfalls keinen Kostenaufwand und v entspricht dem Optimum, um $(a_1, \ldots, a_{i-1})$ in b umzuwandeln. Das bedeutet $v - 0 < K(i-1, j)$, was wieder der Induktionsvoraussetzung widerspricht.
- Zu a wurde eine 0 hinzugefügt, die $b_j = 0$ entspricht. Hier gab es ebenfalls keinen Kostenaufwand und v entspricht dem Optimum, um a in $(b_1, \ldots, b_{j-1})$ umzuwandeln. Auch hier ist $v - 0 < K(i, j-1)$ ein Widerspruch.

Also ist $K(i, j)$ optimal.

Sei nun $a_i = 1$ und angenommen, $K(i, j)$ nicht optimal. Dann gibt es Kosten $v < K(i, j)$, mit denen sich a in b umwandeln lässt. Es gibt drei Möglichkeiten:

- Das Zeichen a_i soll b_j entsprechen, wurde also durch eine 0 ersetzt. Dann gab es einen Kostenaufwand von 2, sodass $v - 2 < K(i-1, j-1)$ die minimalen Kosten sind, um $(a_1, \ldots, a_{i-1})$ in $(b_1, \ldots, b_{j-1})$ umzuwandeln. Das widerspricht allerdings der Induktionsvoraussetzung, dass $K(i-1, j-1)$ optimal ist.
- Das Zeichen a_i wurde gelöscht. Das kostet (wie oben beobachtet) 2 Einheiten. Dann entspricht $v - 2$ dem Optimum, um $(a_1, \ldots, a_{i-1})$ in b umzuwandeln. Das bedeutet $v - 2 < K(i-1, j)$, was wieder der Induktionsvoraussetzung widerspricht.
- Zu a wurde eine 0 hinzugefügt, die $b_j = 0$ entspricht. Hier gab es wie oben keinen Kostenaufwand und v entspricht dem Optimum, um a in $(b_1, \ldots, b_{j-1})$ umzuwandeln. Auch hier ist $v - 0 < K(i, j-1)$ ein Widerspruch.

Also ist $K(i, j)$ optimal.

Aufgabe 9.2 (5 Punkte): (Binäre Bäume und Suchbäume)

Neben dem kennengelernten *Inorder*-Durchlauf, betrachtet ein *Preorder*-Durchlauf zuerst die Wurzel und danach den linken und rechten Teilbaum jedes Teilbaums. Umgekehrt betrachtet ein *Postorder*-Durchlauf zuerst den linken, dann den rechten Teilbaum und danach die Wurzel jedes Teilbaums.

Wir nennen einen Binärbaum der Höhe h *vollständig*, falls alle Knoten zwei oder keine Kinder besitzen und alle Blätter den gleichen Abstand h zur Wurzel haben. Ein Knoten w heißt (direkter) Nachfolger eines Knotens v, falls der Schlüssel $S(w)$ von w größer als der Schlüssel $S(v)$ von v ist und kein Knoten x existiert mit Schlüssel $S(x)$, $S(w) > S(x) > S(v)$. Ein Knoten heißt *innerer Knoten*, falls er kein Blatt is.

a) Geben Sie einen binären Baum an, sodass ein *Inorder*-Durchlauf die gleiche Knotenfolge wie ein *Postorder*-Durchlauf liefert. Geben Sie außerdem einen binären Baum an, sodass ein *Inorder*-Durchlauf die gleiche Knotenfolge wie ein *Preorder*-Durchlauf liefert.

b) Zeigen Sie mit struktureller vollständiger Induktion, dass ein vollständiger binärer Baum 2^h Blätter besitzt.

c) Zeigen Sie, dass in einem vollständigen binären Suchbaum der direkte Nachfolger eines inneren Knotens immer ein Blatt ist.

Lösung:

a) Der *Inorder*-Durchlauf ist gleich dem *Preorder*-Durchlauf, falls kein Knoten einen linken Kindsknoten hat. Jeder Baum dieser Form ist eine gültige Lösung.

Der *Inorder*-Durchlauf ist gleich dem *Postorder*-Durchlauf, falls kein Knoten einen rechten Kindsknoten hat. Jeder Baum dieser Form ist eine gültige Lösung.

b) Wir zeigen die Aussage mittels vollständiger Induktion über die Höhe des vollständigen binären Baums.

Induktionsanfang Für Höhe $h = 0$ gibt es nur einen Knoten, das heißt die Wurzel ist ein Blatt. Es gibt also $1 = 2^0$ Blätter.

Induktionsvoraussetzung Jeder vollständige binäre Baum der Höhe h_0 habe 2^{h_0} Blätter.

Induktionsschritt Bestrachte einen vollständigen binären Baum der Höhe $h = h_0 + 1$. Dieser besteht aus einer Wurzel und deren linken und rechten Teilbaum, die beide vollständige binäre Bäume der Höhe h_0 sind. Die Anzahl der Blätter ist die Summe der Anzahlen der Blätter in beiden Teilbäumen. Diese sind jeweils 2^{h_0} nach Induktionsvoraussetzung. Also haben wir hier $2 \cdot 2^{h_0} = 2^{h_0+1} = 2^h$ Blätter.

c) Wir zeigen folgende Hilfsaussage.

Hilfsaussage: Der direkte Nachfolger eines Knotens, der zwei Kinder hat, kann kein linkes Kind haben.

Daraus lässt sich schließen, dass der Nachfolger eines inneren Knotens ein Blatt sein muss, da in einem vollständigen binären Baum, jeder Knoten entweder zwei Kinder (innerer Knoten) oder keins (Blatt) hat.

Beweis der Hilfsaussage: Sei v ein Knoten in diesem Baum, der zwei Kinderknoten hat. Sei w der direkte Nachfolger von v. Wir nehmen an, dass $x = \text{left}[w]$ der linke Kinderknoten von w ist. Seien $S(v)$, $S(w)$ und $S(x)$ die entsprechende Schlüssel von v, w und x. Es gilt, dass $S(w) > S(x)$ und $S(w) > S(v)$, da x linkes Kind von w und der Nachfolger von v ist.

Da v zwei Kinderknoten hat, muss w im rechten Teilbaum von v liegen: Angenommen w ist nicht im rechten Teilbaum, dann muss w oberhalb von v in dem Baum liegen, weil $S(w) > S(v)$. Da der rechte Teilbaum von v existiert, gibt es einen Knoten mit Schlüssel s, so dass $S(v) < s$. Es gilt aber auch $s > S(w)$, denn w ist der direkte Nachfolger von v in der sortierten Reihenfolge. Dies steht aber im Widerspruch zur Suchbaumeigenschaft, dass alle Schlüssel im linken Teilbaum von w kleiner sind als $S(w)$.

Somit liegt aber auch x im rechten Teilbaum von v, d. h. $S(x) > S(v)$. Damit haben wir insgesamt $S(v) < S(x) < S(w)$, was im Widerspruch zur Nachfolgereigenschaft von w steht. Somit kann es keinen solchen Knoten x geben.

Alternativ lässt sich auch diese Aussage mit struktureller Induktion zeigen.