Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

10. November 2016

Datenstrukturen & Algorithmen Lösungen zu Blatt 7 HS 16

Lösung 7.1 *Palindrome aufzählen.*

a) *Definition der DP-Tabelle:* Wir verwenden eine $n \times n$-Tabelle T mit Einträgen, die entweder 0 oder 1 sind. Für $1 \leq i \leq j \leq n$ sei genau dann $T[i,j] = 1$, wenn $\langle A[i], \ldots, A[j] \rangle$ ein Palindrom ist.

Berechnung eines Eintrags: Wir unterscheiden drei Fälle.

1. Fall: $1 \leq i = j \leq n$. $A[i]$ ist ein Palindrom der Länge 1, also setzen wir $T[i,i] = 1$ für alle i, $1 \leq i \leq n$.

2. Fall: $1 \leq i \leq n$, $j = i + 1 \leq n$. Dann betrachten wir Palindrome der Länge 2, und wir setzen $T[i, i+1] = 1$ genau dann, wenn $A[i] = A[i+1]$ gilt.

3. Fall: $1 \leq i \leq n$, $i + 1 < j \leq n$. Sei $\langle A[i], \ldots, A[j] \rangle$ die betrachtete Zeichenkette. Diese ist nach Definition genau dann ein Palindrom, wenn $A[i] = A[j]$ gilt und zusätzlich $\langle A[i+1], \ldots, A[j-1] \rangle$ ein Palindrom ist. Wir setzen also genau dann $T[i,j] = 1$, wenn $A[i] = A[j]$ gilt und $T[i+1, j-1] = 1$ ist.

Berechnungsreihenfolge: Wir berechnen die Einträge $T[i,j]$ mit wachsender Differenz von $j - i$, starten also bei $T[i,i]$ für $1 \leq i \leq n$. Danach fahren wir fort mit der Berechnung von $T[i, i+1]$ für $1 \leq i \leq n-1$, danach mit $T[i, i+2]$ für $1 \leq i \leq n-2$, usw. Schlussendlich wird $T[1,n]$ berechnet.

Auslesen der Lösung: Wir iterieren über alle Einträge $T[i,j]$, $1 \leq i \leq j \leq n$, und geben genau dann (i,j) aus, wenn $T[i,j] = 1$ ist.

Laufzeit: Die Tabelle hat n^2 Einträge. Die Berechnung jedes Eintrags geht in Zeit $\mathcal{O}(1)$. Zum Auslesen der Lösung wird jeder Eintrag der Tabelle erneut betrachtet, und auch dort fällt nur Zeit $\mathcal{O}(1)$ pro Eintrag an. Die Gesamtlaufzeit beträgt daher $\mathcal{O}(n^2)$.

b) Wir betrachten die Einträge der Tabelle in umgekehrter Berechnungsreihenfolge, d.h. wir starten mit $T[1,n]$, betrachten danach $T[1,n-1]$ und $T[2,n]$, usw. Sobald wir einen Eintrag $T[i,j]$ mit Wert 1 finden, terminiert unser Verfahren, und wir geben $\langle A[i], \ldots, A[j] \rangle$ aus.

Wie vorher fällt bei der Betrachtung eines Eintrags nur Zeit $\mathcal{O}(1)$ an. Da es n^2 Einträge gibt und das ausgegebene Palindrom maximal n Zeichen lang ist, ergibt sich wie vorher eine Gesamtlaufzeit von $\mathcal{O}(n^2)$.

Lösung 7.2 *Aufsteigende Sequenzen.*

Hier gibt es zwei verschieden gute Lösungen. Wir beginnen mit der direkteren und passen sie später an.

Lösung 1:

Definition der DP-Tabelle: Wir verwenden eine Tabelle T der Grösse $n \times m$. Der Eintrag $T[x][y]$ enthält die Länge der längsten aufsteigenden Sequenz $S_{x,y}$, die im Feld $A[x][y]$ endet. Ausserdem enthält $T[x][y]$ die Koordinaten des Vorgängers von (x, y) in $S_{x,y}$, falls vorhanden.

Berechnung eines Eintrags: Die Sequenz $S_{x,y}$ (und damit der Eintrag an Stelle $T[x][y]$) lässt sich aus den Sequenzen $S_{x-1,y}, S_{x+1,y}, S_{x,y-1}, S_{x,y+1}$ berechnen, sofern diese existieren. Dazu nehmen wir die längste Sequenz eines Nachbarn, der einen kleineren Wert als $A[x][y]$ hat, und hängen an diese Sequenz einfach $A[x][y]$ an. Zur Berechnung eines Eintrags $T[x][y]$ wählen wir den Nachbarn (x', y'), der $A[x'][y'] < A[x][y]$ erfüllt und dessen in $T[x'][y']$ gespeicherte Länge maximal ist (unter allen Nachbarn, deren Wert in A kleiner als $A[x][y]$ ist). Wir setzen dann die in $T[x][y]$ gespeicherte Länge um Eins höher als die in $T[x'][y']$ gespeicherte Länge und setzen als Vorgänger des Eintrags $T[x][y]$ einen Zeiger auf (x', y'). Gibt es keinen Nachbarn mit kleinerem Wert in A als $A[x][y]$, dann setzen wir die in $T[x][y]$ gespeicherte Länge auf 1 und den Vorgänger auf **null**, um zu vermerken, dass hier eine neue Sequenz beginnt.

Berechnungsreihenfolge: Da wir für jeden Eintrag nur die Einträge zu kleineren Werten in der Matrix brauchen, können wir die Einträge in aufsteigender Reihenfolge ihres Werts in der Matrix berechnen.

Auslesen der Lösung: Es müssen alle Einträge betrachtet werden, um den Eintrag zu finden, in dem eine längste Sequenz endet. Von dort aus können wir die Lösung rekonstruieren, indem wir dem entsprechenden Vorgänger folgen (dieser ist in $T[x][y]$ vermerkt).

Laufzeit: Insgesamt füllen wir $n \cdot m$ Einträge aus, und für jeden müssen wir vier Nachbarn betrachten. Allerdings müssen wir die Einträge zunächst sortieren, um sie in aufsteigender Reihenfolge durchgehen zu können. Um die Lösung zu finden, müssen wir noch einmal alle Einträge anschauen und dann die Sequenz rekonstruieren – beides benötigt $\mathcal{O}(nm)$ Schritte. Die Laufzeit wird also durch das Sortieren dominiert und beträgt $\mathcal{O}(nm \log(nm))$.

Lösung 2: Wir belassen das dynamische Programm genau wir zuvor, modifizieren aber die Berechnungsreihenfolge, so dass wir um das Sortieren herum kommen.

Berechnungsreihenfolge: Statt die Werte ihrer Grösse nach zu bearbeiten, gehen wir die Matrix A in beliebiger Reihenfolge durch. Wenn wir auf einen Eintrag stossen, der bereits berechnet wurde (wie genau, wird gleich klar), dann überspringen wir ihn. Ansonsten benötigen wir wie vorher die Einträge aller kleineren Nachbarn. Sind diese schon berechnet, haben wir Glück und können unseren Eintrag wie zuvor ausfüllen. Ansonsten müssen wir rekursiv erst die Einträge der Nachbarn bestimmen. Insgesamt starten wir also eine Art Tiefensuche und füllen die tiefsten Einträge in der Suche zuerst aus.

Laufzeit: Das Verfahren kann als eine Mischung aus dynamischer Programmierung und Memoization aufgefasst werden. Wichtig ist, dass wir ganz auf das Sortieren verzichten können. Um effizient zu sein, müssen wir noch sicher gehen, dass wir Einträge nicht zu oft besuchen. Bei jedem Eintrag starten wir einmal eine Tiefensuche. Das bedeutet, dass jeder Eintrag selbst nur vier Mal von einer Tiefensuche besucht werden kann, denn jeder Nachbar startet ja auch nur ein einziges Mal selbst eine Tiefensuche. Insgesamt benötigt die wiederholte Tiefensuche also $\mathcal{O}(nm)$ Schritte. Unsere Laufzeit ist daher ebenfalls $\mathcal{O}(nm)$, also linear in der Eingabegrösse.