Datenstrukturen und Algorithmen
Übung 9, Frühling 2011

21. April 2011

Diese Übung muss zu Beginn der Übungsstunde bis spätestens um 16 Uhr 15 am 5. Mai abgegeben werden. Die Abgabe der DA Übungen erfolgt immer in schriftlicher Form auf Papier. Programme müssen zusammen mit der von ihnen erzeugten Ausgabe abgegeben werden. Drucken Sie wenn möglich platzsparend 2 Seiten auf eine A4-Seite aus. Falls Sie mehrere Blätter abgeben heften Sie diese bitte zusammen (Büroklammer, Bostitch, Mäppchen). Alle Teilnehmenden dieser Vorlesung müssen Ihre eigene Übung abgeben (einzeln oder in Zweiergruppen). Vergessen Sie nicht, Ihren Namen und Ihre Matrikelnummer auf Ihrer Abgabe zu vermerken.

Theoretische Aufgaben

1. In der Vorlesung wurde behauptet, dass ein rekursiver Algorithmus zur optimalen Ablaufkoordination von Montagebändern exponentielle Laufzeit hat.

   Verwenden Sie zuerst die Gleichungen (15.8) und (15.9) aus dem Buch, um zu zeigen, dass die Anzahl $r_i(j)$ der in einem rekursiven Algorithmus notwendigen Zugriffe auf $f_i[j]$ genau 2^{n-j} beträgt. Hinweis: Benutzen Sie die Substitutionsmethode und Induktion über j, wobei der Induktionsanfang bei $j = n$ ist.

   Zeigen Sie dann mit Hilfe dieses Resultats, dass die Gesamtzahl der Zugriffe auf alle $f_i[j]$, das heisst $\sum_{i=1}^{2}\sum_{j=1}^{n} r_i(j)$, genau $2^{n+1} - 2$ ist. **1 Punkt**

2. Beschreiben Sie in zwei oder drei Sätzen, welche überlappenden Teilprobleme bei der Ablaufkoordination von Montagebändern auftreten. **1 Punkt**

3. Bestimmen Sie eine längste gemeinsame Teilsequenz (LCS) von $\langle 0, 1, 0, 1, 0, 1, 1, 0\rangle$ und $\langle 1, 0, 0, 1, 0, 0, 1, 0, 1\rangle$. **1 Punkt**

4. Ein Taxifahrer in New York hat sich in ein gefährliches Quartier verfahren. Er befindet sich in der linken oberen Ecke des Strassengitters, das unten dargestellt ist. Das Ziel des Taxifahrers ist es, in die untere rechte Ecke zu gelangen, wo er das gefährliche Quartier verlässt. Jeder Strassenabschnitt hat ein gewisses Risiko für einen Überfall, das mit ganzen Zahlen angegeben ist. Alle Strassen sind Einbahnstrassen, welche von links nach rechts oder von oben nach unten verlaufen. Gesucht ist ein Weg, der das Gesamtrisiko für einen Überfall, also die Summe der Risiken aller passierten Strassenabschnitte, minimiert.

Nehmen Sie an, die Risiken seien in einem zweidimensionalen Feld r gespeichert. Das heisst $r[i, j]$ ist das Risiko für den Strassenabschnitt in Zeile i und Spalte j des Strassengitters. Die Startposition ist also $i = 1, j = 1$, und die Zielposition $i = 5, j = 7$.

	8	1	2	3	5	7
8	3	3	11	2	9	8
12	9	4	9	2	14	1
7	10	8	16	3	5	9
1	2	2	12	15	8	3

a) Entwerfen Sie einen Algorithmus nach dem Muster der dynamischen Programmierung, der das Gesamtrisiko eines sichersten Weges berechnet. Beschreiben Sie den Algorithmus mit Pseudocode und skizzieren Sie die Lösungstabelle. Geben Sie die Laufzeit Ihres Algorithmus an. **1 Punkt**

b) Beschreiben Sie in Worten, wie durch Rückverfolgung in der Lösungstabelle, die Ihr Algorithmus aus a) berechnet, die traversierten Zellen eines sichersten Weges gefunden werden können. Zeichnen Sie den Weg in der Lösungstabelle ein. Geben Sie die Laufzeit für die Rückverfolgung an. **1 Punkte**

Praktische Aufgaben

In dieser Aufgabe verwenden Sie dynamische Programminerung, um die Grösse von Rasterbildern zu verändern. Die Grösse von Bildern kann am einfachsten durch lineare Skalierung oder durch Abschneiden (cropping) verändert werden. Skalierung führt aber zu Verzerrungen, und Abschneiden kann zum Verlust von wichtigen Bildteilen führen. Sie werden in dieser Übung ein Verfahren implementieren, das versucht, den Inhalt des Bildes so wenig wie möglich zu verändern. Der Algorithmus wurde unter der Bezeichnung *Seam Carving* bekannt[1]. Wir stellen auf der Vorlesungs Webpage Java Code zur Verfügung, auf dem Sie aufbauen sollen.

Die Idee von Seam Carving ist, im Bild Schritt für Schritt *ein Pixel breite Nähte (seams)* zu entfernen, bis die gewünschte Bildgrösse erreicht ist. Eine Naht ist entweder *vertikal*, um die Breite des Bildes zu verändern, oder *horizontal*, um die Höhe zu verändern. Die Idee ist nun, jeweils die "beste" Naht zu bestimmen, die möglichst unauffällig aus dem Bild entfernt werden kann. Dies ist in der Abbildung unten illustriert.

Um die "Güte" für eine Naht zu bestimmen, definieren wir zuerst eine Energiefunktion $e(x, y)$ für einzelne Pixel (x, y). Diese Energiefunktion beschreibt die Kosten, die entstehen, wenn der Pixel aus dem Bild entfernt wird. Intuitiv sollen die Kosten hoch sein, wenn sich das Bild an diesem Pixel stark ändert. In dem Fall führt das Entfernen des Pixels zu starken Kanten. Deshalb macht es Sinn, die Funktion e über die Ableitung des Bildes zu berechnen. Wir stellen eine Implementation dieser Funktion zur Verfügung (die Methode *energy*). Die Kosten für eine Naht sind nun einfach die Summe der Kosten über alle Pixel in der Naht.

[1] siehe zum Beispiel *http://en.wikipedia.org/wiki/Seam_carving*

Abbildung 1: Beispiel für Seam Carving mit einer vertikalen Naht. Links: Eingabebild. Mitte: Visualisierung der optimalen vertikalen Naht. Rechts: Ausgabe nach sukzessiver Entfernung von 100 Nähten.

In dieser Übung beschränken wir uns darauf, optimale *vertikale* Nähte zu finden. Der Algorithmus um die optimale Naht zu finden basiert auf dynamischer Programmierung und läuft wie folgt ab: Wir berechnen zuerst eine Tabelle M, welche für jeden Pixel die Kosten der optimalen Naht enthält, die durch diesen Pixel geht. Im zweiten Schritt wird aus der Tabelle die Naht selbst rekonstruiert. Im letzten Schritt wird das Ausgabebild ohne die Pixel der Naht erzeugt. Die ersten zwei Schritte sollen Sie implementieren, der dritte ist bereits im Code vorhanden (die Methode *removeSeam*).

Um das Berechnen der Kosten zu vereinfachen definieren wir eine vertikale Naht als eine Reihe von Pixeln, je ein Pixel pro Bildzeile, welche von Zeile zu Zeile um *höchstens einen* Pixel horizontal verschoben sind. Damit lassen sich die Kosten $M(x, y)$ für die otpimale Naht durch Pixel (x, y) leicht durch folgende Rekursionsformel bestimmen:

$$M(x, y) = e(x, y) + \min(M(x - 1, y - 1), M(x, y - 1), M(x + 1, y - 1).$$

Die Tabelle wird in der Zeile $y = 0$ mit den Werten $e(x, 0)$ initialisiert.

1. Implementieren Sie im Java Code die Methode *computeCosts*, mit der die Tabelle M berechnet wird. Hinweis: Initialisieren Sie zuerst die Zeile $y = 0$, und berechnen Sie dann die Tabelle Zeile für Zeile indem Sie bei $y = 1$ anfangen. **2 Punkte**

2. Implementieren Sie im Java Code die Methode *computeSeam*, mit der aus der Tabelle M eine optimale Naht rekonstruiert wird. Hinweis: Finden Sie in der Tabelle M zuerst denjenigen Pixel auf Zeile $y = height - 1$ mit den kleinsten Kosten. Dies ist der Endpunkt der Naht. Verfolgen Sie dann die Naht zurück zur Zeile $y = 0$. **2 Punkte**

3. Demonstrieren Sie Ihre Implementation mit zwei oder drei Beispielbildern. **1 Punkt**