9. Aufgabenblatt zur Vorlesung

Höhere Algorithmik

WiSe 2016/17

Wolfgang Mulzer, Boris Klemz

Abgabe bis zum 6. Januar 2017, 12 Uhr, in den Briefkasten neben Raum 111

Dieser Zettel enthält nur Zusatzaufgaben. Das HA-Team wünscht frohe Weihnachten und ein gutes neues Jahr.

Aufgabe 1 Verständnisfragen *10 Zusatzpunkte*

(a) Definieren Sie die folgenden Begriffe: worst-case Laufzeit, average-case Laufzeit, amortisierte Laufzeit, erwartete Laufzeit. Finden Sie Gemeinsamkeiten und Unterschiede.

(b) Nennen Sie Gemeinsamkeiten und Unterschiede zwischen Divide-and-Conquer und dynamischem Programmieren.

(c) Angenommen, wir haben einen ungerichteten Graphen $G = (V, E)$ mit Knotenmenge $V = \{a, b, c, d, e, f, g, h, i, j, k\}$ gegeben. Wir möchten die Zusammenhangskomponenten von G bestimmen. Dazu verwenden wir die Union-Find basierte Strategie aus der Vorlesung. Nehmen Sie an, dass die Kanten aus E in folgender Reihenfolge bearbeitet werden: (d, i), (f, k), (g, i), (b, g), (a, h), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e), (i, d).

Zeigen Sie den Status der Union-Find Datenstruktur nach der Bearbeitung jeder Kante. Verwenden Sie Union-By-Rank und Pfadkompression. Geben Sie auch die Ränge der einzelnen Knoten an.

Aufgabe 2 Blocksatz *10 Zusatzpunkte*

Gegeben sei ein Text, also eine Folge $(w_1, \ldots, w_n)$ von n Wörtern. Dabei habe Wort w_i die Länge l_i. Wir möchten diesen Text möglichst gut als Blocksatz mit gegebener Zeilenlänge M setzen (alle Wörter haben höchstens M Buchstaben). Das Kriterium für "möglichst gut" wird im Folgenden beschrieben:

Angenommen eine Zeile z besteht aus den Wörtern $w_i, \ldots, w_j$. Dann enthält diese Zeile

$$M - \sum_{k=i}^{j} l_k$$

Leerzeichen. Sei nun die *Strafe* p_z für eine Zeile z definiert als

$$p_z := \begin{cases} \left(\frac{M - \sum_{k=i}^{j} l_k}{j-i}\right)^3 \cdot (j-i), & \text{falls } j > i \\ \left(M - l_i\right)^3, & \text{falls } i = j. \end{cases}$$

Ein Blocksatz gilt als "gut", wenn die Summe der Strafen p_z über alle Zeilen z außer der letzten minimal ist.

Beispiel:

a	a	a		a	a		a	$(2/2)^3 \cdot 2$	$=$	2
b	b				b	b	b	$(3/1)^3 \cdot 1$	$=$	27
c	c	c	c					4^3	$=$	64
d	d	d	d	d						
										93

(a) Lösen Sie dieses Problem mittels Dynamischer Programmierung. Analysieren Sie die Laufzeit und den Speicherbedarf Ihres Algorithmus und begründen Sie Ihr Vorgehen.
Anmerkung: Natürlich sollen zwei aufeinander folgende Wörter einer Zeile durch mindestens ein Leerzeichen getrennt sein.

(b) Implementieren Sie Ihren Algorithmus in Java. Testen Sie Ihr Programm mit verschiedenen Texten und vergleichen Sie "Ihren" Blocksatz mit dem einer Textverarbeitungssoftware (z.B. MS Word, LaTeX, LibreOffice). Zeigen Sie ein Beispiel, bei dem das Programm eine bessere Ausgabe erzeugt als das triviale Greedy-Verfahren (jede Zeile wird so weit wie möglich gefüllt und dann ausgerichtet).

Aufgabe 3 Viterbi-Algorithmus *10 Zusatzpunkte*

Auf dem 4. Aufgabenblatt haben Sie sich in Aufgabe 3(c) eine Möglichkeit überlegt, wie man mit dem Viterbi-Algorithmus Rechtschreibfehler korrigieren kann. Setzen Sie nun Ihre Idee in die Tat um. Modellieren und implementieren Sie versteckte Markov-Modelle in Java und implementieren Sie sodann den Viterbi-Algorithmus. Entwickeln Sie auch eine Klasse, welche die Übergangswahrscheinlichkeiten für Ihr Modell durch Abzählen ermittelt (Beachten Sie auch den Trick aus Aufgabe 3(b)).

Nehmen Sie dann einen repräsentativen Text mittlerer Länge (etwa 200 KB). Verwenden Sie die ersten 50 KB zum Lernen und verändern Sie den Rest, indem Sie jedes Zeichen mit Wahrscheinlichkeit 10% durch ein zufälliges Zeichen ersetzen. Wie gut schneidet Ihr Algorithmus auf diesen Testdaten ab? Experimentieren Sie mit verschiedenen Trainingslängen und Fehlerraten.

Aufgabe 4 Dynamisches Programmieren und Divide-and-Conquer *10 Zusatzpunkte*

In dieser Aufgabe wollen wir uns überlegen, wie man beim dynamischen Programmieren Speicherplatz einsparen kann. Dabei wird die Laufzeit nur wenig schlechter. Als Beispiel soll das Einkaufsproblem (Rucksackproblem) dienen.

(a) In der Vorlesung haben Sie gesehen, wie man das Einkaufproblem mit n Artikeln und Budget B in Zeit und Platz $O(nB)$ lösen kann. Argumentieren Sie, dass man mit $O(n+B)$ Platz auskommen kann, wenn man nur den *Wert* einer optimalen Lösung sucht.

(b) Erinnern Sie sich, dass ein optimaler Einkauf einem Weg in der Tabelle E von $E[n, B]$ zu $E[0, 0]$ entspricht, wobei die einzelnen Schritte von dem Array `kaufen` abhängen (siehe Notizen auf der Vorlesungsseite). Zeigen Sie: Man kann die Stelle, an der ein optimaler Weg die Zeile $n/2$ kreuzt, in $O(B)$ Platz berechnen.

(c) Benutzen Sie das Divide-and-Conquer Prinzip, um einen optimalen Weg von $E[n, B]$ nach $E[0, 0]$ in $O(n + B)$ Platz zu berechnen. Was ist die resultierende Laufzeit?

Hinweis: Verwenden Sie (b), um die Stelle zu finden, an der ein optimaler Weg die Zeile $n/2$ kreuzt. Berechnen Sie dann rekursiv einen optimalen Weg von Zeile n zu Zeile $n/2$ und von Zeile $n/2$ zu Zeile 0. Beachten Sie, dass sich Speicherplatz wiederverwenden lässt.