

Frank Hoffmann  
Klaus Kriegel  
Romain Grunert  
Ludmilla Scharf  
André Schulz

Ws 2007/08

23. November 2007  
Abgabe: 3. Dezember 2007  
vor der Vorlesung

## 6. Übung zur Vorlesung Höhere Algorithmik

### 1. Aufgabe (5 Punkte).

Entwerfen Sie für die folgenden Probleme effiziente Algorithmen, die das Prinzip der dynamischen Programmierung nutzen.

#### (a) Shortest Supersequence:

Gegeben sind zwei Wörter  $S$  und  $T$  über dem Alphabet  $\Sigma$ . Gesucht ist das kleinste Oberwort  $O$ , das sowohl  $S$  als auch  $T$  als Teilsequenz enthält. (Beispiel  $S=GT$ ,  $T=ATT$ ,  $O=GATT$ ).

#### (b) End-free Alignment:

Gegeben seien zwei Wörter  $S$  und  $T$  über dem Alphabet  $\Sigma$ . Wir wollen ein optimales globales Alignment beider Wörter bestimmen, bei dem äußere Leerzeichen vor bzw. nach den Wörtern im Alignment nicht berücksichtigt, also mit 0 bewertet werden. Bis auf diese Ausnahme erfolgt die Bewertung wie üblich entsprechend einer gegebenen Scoring-Matrix.

Beschreiben Sie, wie man den Standardansatz für ein optimales globales Alignment mittels Dynamischer Programmierung abändern muss, um diese Aufgabe zu lösen.

### 2. Aufgabe (5 Punkte).

Neben der Binärsuche in einem beschränkten linear geordneten Bereich gibt es mit der exponentiellen Suche ein anderes Suchparadigma, was sich an folgendem Beispiel gut illustrieren lässt:

Sie stehen in finsterner Nacht an einem beidseitig unendlichen Zaun (etwa im Ursprung der Zahlengerade), von dem Zaun wissen Sie, dass er irgendwo ein Loch (entspricht einem Punkt  $z \in \mathbb{Z}$ ) hat. Wie finden Sie das Loch??? Sie können nach links bzw. nach rechts laufen. Die Entscheidung sofort (oder nach endlich vielen Richtungswechseln) nur in eine Richtung zu laufen, ist natürlich nicht ratsam. Das Loch könnte ja genau auf der anderen Seite sein. Jede erfolgreiche Strategie muss immer wieder die Richtung ändern und ist charakterisiert durch eine unendliche Folge  $f_0, f_1, \dots, f_n, \dots$  von "Wendepunkten", die abwechselnd auf der rechten bzw. linken Seite des Startpunktes liegen, wobei natürlich  $|f_i| > |f_{i-2}|$  vorausgesetzt werden kann. Nehmen wir an, das Loch befindet sich auf der rechten Seite. Machen Sie sich zunächst die folgende Aufgabenstellung an einer Skizze deutlich.

(a) Sei  $S_1$  die Strategie, die zuerst nach rechts geht und die Wendepunkte bei  $|f_i| = i+1$

hat. Berechnen Sie die Gesamtlänge des Weges bis zum Erreichen des Zieles bei  $z = 1025$ .

- (b) Ersetzen Sie  $S_1$  durch  $S_{exp}$  mit  $|f_i| = 2^i$ , bei dieser Strategie wird die Suchtiefe also immer verdoppelt! Berechnen Sie auch hier die Gesamtlänge des Weges bis zum Erreichen des Zieles bei  $z = 1025$ .
- (c) Wir bewerten die beiden Strategien, indem wir den zurückgelegten Wert für die Suche durch den kürzesten Weg zum Loch teilen. Dieser Quotient gibt uns die Güte einer Strategie an. Bestimmen Sie die Quotienten für beide Strategien wenn sich das Loch in Entfernung  $2^{2k} + 1$  (für  $k \in \mathbb{N}$ ) vom Ausgangspunkt befindet.

**3. Aufgabe** (5 Punkte).

Gegeben sind  $n$  Orte  $V = \{1, \dots, n\}$  und eine Abstandsfunktion  $w : V \times V \rightarrow \mathbb{R}$ . Wir suchen eine kürzeste Tour (Weg der jeden Knoten genau einmal besucht, wobei Startknoten=Zielknoten), bezüglich der Summe der Teilwege.

Dieses Problem läßt sich mit  $O(n!)$  Schritten lösen in dem man einfach alle Touren ausprobiert. Zeigen Sie, dass dies mit dynamischer Programmierung besser geht (wenn auch immer noch nicht polynomiell). Betrachten Sie hierzu die Teilprobleme der Art: Ich befinde mich im Ort  $i$ , muss noch die Orte  $S \subseteq \{2, \dots, n\}$  besuchen und danach zurück zum Startpunkt 1.

Stellen Sie eine geeignete Rekursion auf, die dieses Problem modelliert. Analysieren Sie die Laufzeit des induzierten Algorithmus und argumentieren Sie, warum dieser schneller ist als  $O(n!)$ .

**4. Aufgabe** (5 Punkte).

Gegeben ist ein Text  $\mathcal{T}$  als eine Sequenz von  $n$  Worten  $(w_i)_{i \leq n}$  der Länge  $|w_i|$ . Wir wollen diesen Text linksbündig auf einer Seite anordnen, auf der  $L$  Zeichen pro Zeile Platz haben. Wörter innerhalb einer Zeile sollen durch (genau ein) Leerzeichen getrennt sein. Sei  $s(i)$  der Index des ersten Wortes auf der  $i$ -ten Zeile und  $e(i)$  der Index des letzten Wortes auf der  $i$ -ten Zeile. Ferner sei  $R_i$  der rechte Rand bei der linksbündigen Anordnung des Textes in Zeile  $i$ , also  $R_i = L - (\sum_{j=s(i)}^{e(i)} |w_j|) + s(i) - e(i)$ .

Wir wollen den Text möglichst gleichmäßig anordnen und auch einen kleinen rechten Rand benutzen. Deshalb versuchen wir folgende *Straffunktion*  $\mathcal{P}$  zu minimieren:

$$\mathcal{P} := \sum_{i \text{ ist Zeile}} R_i^2.$$

Bestimmen sie die optimale Anordnung des Textes bezüglich  $\mathcal{P}$ . Nutzen Sie dynamisches Programmieren.