Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Moritz Laupichler
Nikolai Maas

5. Übungsblatt zu Algorithmen II im WS 2023/2024

https://algo2.iti.kit.edu/AlgorithmenII_WS23.php
{sanders, moritz.laupichler, nikolai.maas}@kit.edu

Aufgabe 1 (*Rechnen+Analyse: Suche in Strings*)

a) Zur Ausführung des KMP-Algorithmus muss zunächst ein sogenanntes *border-array* berechnet werden. Geben Sie das *border-array* für das Suchmuster $P =$ `abacababc` an.

b) Führen Sie den KMP-Algorithmus auf dem Text $T =$ `abacababbabacababc` mit obigem Suchmuster durch.

c) Wie oft muss der KMP-Algorithmus ein Muster P der Länge $|P|$ maximal an einen Text T der Länge $|T|$ anlegen, falls P nicht in T vorkommt? Wie oft minimal? Geben Sie das Ergebnis in Abhängigkeit von $|P|$ und $|T|$ an. Geben Sie außerdem jeweils ein Beispiel für P und T an.

d) Zeigen oder widerlegen Sie:
Das *border-array* kann keine drei aufeinanderfolgenden Einträge enthalten, die jeweils um eins kleiner sind als ihr Vorgänger, falls der erste von diesen drei Einträgen auf ein Suffix der Größe $k \geq 3$ verweist, welches gleichzeitig echtes Präfix ist.

Beispiel: $\text{border}[10] = 3,\ \text{border}[11] = 2,\ \text{border}[12] = 1$

Kein Beispiel: $\text{border}[10] = 2,\ \text{border}[11] = 1,\ \text{border}[12] = 0$

Aufgabe 2 (*Rechnen: Burrows-Wheeler-Transformation*)

Die Entropie $H(T)$ einer Zeichenkette T gibt eine untere Schranke für die Anzahl Bits pro Zeichen an, die zur Encodierung von T nötig wären, falls T einer Zufallsquelle entspränge. Sie ist definiert als

$$H(T) = \sum_{c \in \Sigma} p(c) \log_2(1/p(c))$$

wobei $p(c) := |\{s_i : s_i = c\}|/|T|$ die relative Häufigkeit von c in T ist.

a) Bestimmen Sie die Entropie von Zeichenkette $s =$ `abababababab`.

b) Führen Sie die *Burrows-Wheeler-Transformation* auf Zeichenkette s aus. Wie groß ist jetzt die Entropie der Zeichenkette?

c) Führen Sie eine *Move-to-Front* Kodierung auf dem Ergebnis durch.
(das Abschlusszeichen `$` aus der BWT muss bei der Kompression nicht berücksichtigt werden)

d) Vergleichen Sie das Ergebnis mit einer direkten *Move-to-Front* Kodierung von s. Wie groß ist jeweils die Entropie der beiden kodierten Zeichenketten?

e) Führen Sie eine inverse *Burrows-Wheeler-Transformation* auf Zeichenkette $s^{BWT} =$ `bc$aab` aus. Erzeugen und verwenden Sie hierfür das LF-Array der Zeichenkette (siehe Übung).

Aufgabe 3 (*Rechnen: Suffixarrays und DC3-Algorithmus*)

Gegeben sei die Zeichenkette $s =$ `aberakadabera$`.

a) Geben Sie den Suffixbaum für s an.

b) Geben Sie das Suffixarray für s an.

In der Vorlesung haben Sie einen Linearzeitalgorithmus zur Konstruktion von Suffixarrays kennengelernt. Dieser ist unter dem Namen *DC3-Algorithmus* bekannt. Im Folgenden soll der Algorithmus Schritt für Schritt per Hand ausgeführt werden.
Die Suffixe von s werden zunächst in drei Sequenzen $B_i = \langle s_k \mid (k \bmod 3) = i \rangle$ für $i \in \{0, 1, 2\}$ aufgeteilt. Danach müssen die Sequenzen $C = B_0 \cup B_1$ und B_2 lexikographisch sortiert werden.

Sortierung von C:

c) Geben Sie die Tripelsequenzen $R_k = \langle s[i..i+2] \mid (i \bmod 3) = k \rangle$ für $k \in \{0, 1\}$ an. Für $i \geq |s|$ gelte $s[i] =$ \$ (Auffüllen mit zusätzlichen Abschlusszeichen).

d) Bestimmen Sie den *Rang* der Tripel von $R = R_0 \circ R_1$. Sortieren Sie dazu die Tripel und entfernen mehrfache Vorkommnisse. Die Position eines Tripels in dieser Sortierung gibt seinen Rang an.

e) Die berechneten Ränge definieren eine eindeutige Bezeichnung für jedes Tripel in R. Drücken Sie R mit Hilfe dieser Ränge aus. Diese Darstellung ergibt die Zeichenkette s^{01}. Muss der DC3-Algorithmus eine Rekursion ausführen?

f) Geben Sie das Suffixarray SA^{01} für s^{01} von Hand an (unabhängig, ob der DC3-Algorithmus eine Rekursion durchführt). Vergewissern Sie sich, dass SA^{01} eine Sortierung von C beschreibt.

Sortierung von C_2:

g) Erstellen Sie eine Zuordnung *rank*, die jedem i mit $s_i \in C$ den Index von s_i in der sortierten Sequenz C zuweist. Für alle anderen i sei $rank(i) = 0$.

Formale Berechnung von *rank* mit Hilfe des Suffixarray SA^{01} nach:

$$\begin{array}{lcll} rank[3 \cdot (\mathrm{SA}^{01}[i])] & = & i & \text{falls} \quad \mathrm{SA}^{01}[i] < |B_0| \\ rank[3 \cdot (\mathrm{SA}^{01}[i] - |B_0|) + 1] & = & i & \text{falls} \quad \mathrm{SA}^{01}[i] \geq |B_0| \end{array}$$

Alle anderen Werte von $rank[i]$ können gleich 0 gesetzt werden.

h) Erstellen Sie Tupel $(s[i], rank[i{+}1])$ f.a. $s_i \in B_2$ und sortieren diese lexikographisch. Vergewissern Sie sich, dass diese Sortierung einer Sortierung von B_2 entspricht.

Nachdem C und B_2 sortiert worden sind, kann das Suffixarray von s bestimmt werden:

i) Führen Sie eine Mischen-Operation auf C und B_2 aus. Die resultierende Sequenz wird mit D bezeichnet. Sei $s_i \in C$ und $s_j \in B_2$ Es gelten folgende Sortierkriterien:

$$s_i \leq s_j \iff \begin{cases} (s[i], \quad rank[i+1]) & \leq & (s[j], \quad rank[j+1]) & \text{falls} \quad s_i \in B_0 \\ (s[i], s[i+1], rank[i+2]) & \leq & (s[j], s[j+1], rank[j+2]) & \text{falls} \quad s_i \in B_1 \end{cases}$$

Vergewissern Sie sich, dass D lexikographisch sortiert ist und damit das Suffixarray induziert.

Notation:
- Alle Indizes fangen bei 0 an – analog zu Kapitel 9.3.6, auf dem die Aufgabe basiert.
- $s[i\ (..j)\ ]$: Zeichen an Stelle i (bis j) in s (z.B. $s[1..3] =$ `ber`)
- s_i: Suffix von s ab Stelle i (z.B. $s_2 =$ `erakadabera`)

Aufgabe 4 (*Rechnen+Analyse: LCP-Array*)

Gegeben sei die Zeichenkette $s = \texttt{salsadipp\$}$.

a) Geben Sie den Suffixbaum für s an.

b) Geben Sie das Suffixarray für s an.

c) Geben Sie das LCP-Array für s an.

Im Folgenden sei ein String T sowie dessen Suffixarray SA[·] und dessen LCP-Array LCP[·] gegeben.

d) Wie kann der längste sich wiederholende Substring in T effizient bestimmt werden?
(der Substring darf sich dabei selbst überlappen)

e) Wie viele paarweise unterschiedliche Substrings kann ein String der Länge n maximal besitzen? Wie kann die tatsächliche Anzahl für einen konkreten String T bestimmt werden?

f) Ein String lässt sich schlecht komprimieren, wenn er wenig Redundanz besitzt. Ein Maß dafür ist die Anzahl paarweise unterschiedlicher Substrings normiert auf die mögliche Gesamtanzahl unterschiedlicher Substrings. Geben Sie an, wie dieses Maß für T berechnet werden kann.

Aufgabe 5 (*RMQ in Wavelet Trees*)

Gegeben sei ein Universum von Zahlen $\mathcal{U}$ und ein Feld A von Zahlen aus $\mathcal{U}$. Geben sie einen Algorithmus an, mit dem sich unter Benutzung eines Wavelet Trees in $\log |\mathcal{U}|$ Zeit $\arg\min_i\{A[i] \mid i \in [a, b]\}$ für Parameter a, b berechnen lässt.

Aufgabe 6 (*Rechnen: Kompression*)

a) Bestimmen Sie die *LZ77-Faktoren* $f_i = (l_i, p_i)$ des Textes $T_1 = \texttt{abbbababbbb\$}$.

b) Gegeben seien folgende *LZ77-Faktoren*. Bestimmen Sie den Text T_2, aus dem sie entstanden sind.

$$(0, \texttt{a}), (5, 1), (0, \texttt{b}), (6, 1), (7, 7), (0, \texttt{\$})$$

c) Angenommen, jedes Zeichen kann in 1 byte, sowie jeder *LZ77-Faktor* in 2 byte repräsentiert werden. Wie viel Prozent des Speicherplatzes von T_2 konnte somit eingespart werden?

Ausgabe: 19.01.2024
Besprechung: 30.01.2024