

7. Übungsblatt zu Algorithmen II im WS 2018/2019

<http://algo2.iti.kit.edu/AlgorithmenII-WS18.php>
{sanders, lamm, hespe}@kit.edu

Aufgabe 1 (Rechnen: Kompression)

- Führen Sie eine Kompression nach *Lempel-Ziv* auf der Zeichenkette $s = \text{abababcabcc}$ aus.
- Führen Sie eine Dekompression auf der codierten Zeichenkette $c = 1, 2, 3, 5, 4, 2$ aus. Sie wissen, dass die ursprüngliche Zeichenkette über dem Alphabet $\{a, b\}$ aufgebaut ist.
- Durch einen Fehler in der Datenübertragung wurde nur jedes zweite Zeichen einer mit *Lempel-Ziv* komprimierten Zeichenkette übertragen. Glücklicherweise wurde auch das verwendete Wörterbuch mitübertragen. Leider fehlt auch hier jeder zweite Eintrag. Rekonstruieren Sie die codierte Zeichenkette und das Wörterbuch und geben Sie die unkomprimierte Zeichenkette an.

Die übermittelte Zeichenkette lautet $c = 1, ?, 2, ?, 5$. Das übermittelte Wörterbuch enthält die Einträge $D = \{(1, ?), (2, b), (3, ?), (4, \text{aab}), (5, ?), (6, \text{aabb})\}$. Sie wissen außerdem, dass die ursprüngliche Zeichenkette über dem Alphabet $\{a, b\}$ aufgebaut wurde.

Aufgabe 2 (Rechnen+Analyse: Suche in Strings)

- Zur Ausführung des KMP-Algorithmus muss zunächst ein sogenanntes *border-array* berechnet werden. Geben Sie das *border-array* für das Suchmuster $p = \text{abacababc}$ an.
- Führen Sie den KMP-Algorithmus auf dem Text $t = \text{abacababbabacababc}$ mit obigem Suchmuster durch.
- 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.
- 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 3 (Rechnen: Burrows-Wheeler-Transformation)

- (*) Bestimmen Sie die Entropie von Zeichenkette $s = \text{ababababab}$.
- Führen Sie die *Burrows-Wheeler-Transformation* auf Zeichenkette s aus. Wie groß ist jetzt die Entropie der Zeichenkette?
- Führen Sie eine *Move-to-Front* Kodierung auf dem Ergebnis durch.
(das Abschlusszeichen $\$$ aus der BWT muss bei der Kompression nicht berücksichtigt werden)
- Vergleichen Sie das Ergebnis mit einer direkten *Move-to-Front* Kodierung von s . Wie groß ist jeweils die Entropie der beiden kodierten Zeichenketten?
- Führen Sie eine inverse *Burrows-Wheeler-Transformation* auf Zeichenkette $s^{BWT} = \text{bc}\$ \text{aab}$ aus.

Aufgabe 4 (Rechnen: Suffixarrays und DC3-Algorithmus)

Gegeben sei die Zeichenkette $s = \text{aberakadabera}$.

- Geben Sie den Suffixbaum für s an.
- 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 $C^k = \langle s_i \mid (i \bmod 3) = k \rangle$ für $k \in \{0, 1, 2\}$ aufgeteilt. Danach müssen die Sequenzen C^0 und $C^{12} = C^1 \cup C^2$ lexikographisch sortiert werden.

Sortierung von C^{12} :

- Geben Sie die Tripelsequenzen $R^k = \langle s[i..i+2] \mid (i \bmod 3) = k \rangle$ für $k \in \{1, 2\}$ an. Für $i \geq |s|$ gelte $s[i] = \$$ (Auffüllen mit zusätzlichen Abschlusszeichen).
- Bestimmen Sie den Rang der Tripel von $R^{12} = R^1 \circ R^2$. Sortieren Sie dazu die Tripel und entfernen mehrfache Vorkommnisse. Die Position eines Tripels in dieser Sortierung gibt seinen Rang an.
- Die berechneten Ränge definieren eine eindeutige Bezeichnung für jedes Tripel in R^{12} . Drücken Sie R^{12} mit Hilfe dieser Ränge aus. Diese Darstellung ergibt die Zeichenkette s^{12} . Muss der DC3-Algorithmus eine Rekursion ausführen?
- Geben Sie das Suffixarray SA^{12} für s^{12} von Hand an (unabhängig, ob der DC3-Algorithmus eine Rekursion durchführt). Vergewissern Sie sich, dass SA^{12} eine Sortierung von C^{12} beschreibt.

Sortierung von C^0 :

- Erstellen Sie eine Zuordnung $rank$, die jedem i mit $s_i \in C^{12}$ den Index von s_i in der sortierten Sequenz C^{12} zuweist. Für alle anderen i sei $rank(i) = 0$.

Formale Berechnung von $rank$ mit Hilfe des Suffixarray SA^{12} nach:

$$\begin{aligned} rank[3 \cdot (SA^{12}[i]) + 1] &= i & SA^{12}[i] < |C^1| \\ rank[3 \cdot (SA^{12}[i] - |C^1|) + 2] &= i & SA^{12}[i] \geq |C^1| \end{aligned}$$

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

- Erstellen Sie Tupel $(s[i], rank[i+1])$ f.a. $s_i \in C^0$ und sortieren diese lexikographisch. Vergewissern Sie sich, dass diese Sortierung einer Sortierung von C^0 entspricht.

Nachdem C^0 und C^{12} sortiert worden sind, kann das Suffixarray von s bestimmt werden:

- Führen Sie eine Mischen-Operation auf C^0 und C^{12} aus. Die resultierende Sequenz wird mit C bezeichnet. Es gelten folgende Sortierkriterien:

$$s_i \leq s_j \iff \begin{cases} (s[i], rank[i+1]) \leq (s[j], rank[j+1]) & s_j \in C^1 \\ (s[i..i+1], rank[i+2]) \leq (s[j..j+1], rank[j+2]) & s_j \in C^2 \end{cases}$$

Vergewissern Sie sich, dass C 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.
- $s[i \dots j]$: Zeichen an Stelle i (bis j) in s (z.B. $s[1..3] = \text{ber}$)
- s_i : Suffix von s ab Stelle i (z.B. $s_2 = \text{erakadabera}$)

Aufgabe 5 (*Rechnen+Analyse: LCP-Array*)

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

- a) Geben Sie das Suffixarray für s an.
- b) Geben Sie das LCP-Array für s an.

Im Folgenden sei ein String t sowie dessen Suffixarray $\text{SA}[\cdot]$ und dessen LCP-Array $\text{LCP}[\cdot]$ gegeben.

- c) Wie kann der längste sich wiederholende Substring in t effizient bestimmt werden?
(der Substring darf sich dabei selbst überlappen)
- d) 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?
- e) 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 6 (*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.

Ausgabe: 29.1.2019

Abgabe: keine Abgabe, keine Korrektur