Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Sebastian Lamm, Demian Hespe

7. Übungsblatt zu Algorithmen II im WS 2018/2019

http://algo2.iti.kit.edu/AlgorithmenII_WS18.php
{sanders, lamm, hespe}@kit.edu

Musterlösungen

Aufgabe 1 (*Rechnen: Kompression*)

a) Führen Sie eine Kompression nach *Lempel-Ziv* auf der Zeichenkette $s = \texttt{abababcabcc}$ aus.

b) Führen Sie eine Dekompression auf der codierten Zeichenkette $c = \texttt{1,2,3,5,4,2}$ aus. Sie wissen, dass die ursprüngliche Zeichenkette über dem Alphabet $\{\texttt{a}, \texttt{b}\}$ aufgebaut ist.

c) 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 $D = \{(1, ?), (2, \texttt{b}), (3, ?), (4, \texttt{aab}), (5, ?), (6, \texttt{aabb})\}$. Sie wissen außerdem, dass die ursprüngliche Zeichenkette über dem Alphabet $\{\texttt{a}, \texttt{b}\}$ aufgebaut wurde.

Musterlösung:

a) Die komprimierte Zeichenkette lautet: $c = 1, 2, 4, 4, 3, 7, 3$
Das aufgebaute Wörterbuch `D` lautet:
$D = \{(1, \texttt{a}), (2, \texttt{b}), (3, \texttt{c}), (4, \texttt{ab}), (5, \texttt{ba}), (6, \texttt{aba}), (7, \texttt{abc}), (8, \texttt{ca}), (9, \texttt{abcc})\}$

Index	Zeichenfolge
1	a
2	b
3	c
4	ab
5	ba
6	aba
7	abc
8	ca
9	abcc

b) Die dekomprimierte Zeichenkette lautet: $s = \texttt{a}, \texttt{b}, \texttt{ab}, \texttt{aba}, \texttt{ba}, \texttt{b}$
Das aufgebaute Wörterbuch D lautet:
$D = \{(1, \texttt{a}), (2, \texttt{b}), (3, \texttt{ab}), (4, \texttt{ba}), (5, \texttt{aba}), (6, \texttt{abab}), (7, \texttt{bab})\}$

Index	Zeichenfolge
1	a
2	b
3	ab
4	ba
5	aba
6	abab
7	bab

c) Die übermittelte Zeichenkette lautet $c = 1, \underline{3}, 2, \underline{4}, 5$.
Das übermittelte Wörterbuch D lautet:
$D = \{(1, \underline{\texttt{a}}), (2, \texttt{b}), (3, \underline{\texttt{aa}}), (4, \texttt{aab}), (5, \underline{\texttt{ba}}), (6, \texttt{aabb})\}$

Die dekomprimierte Zeichenkette lautet $s = \texttt{a}, \texttt{aa}, \texttt{b}, \texttt{aab}, \texttt{ba}$.

Lösungsweg:
$D[1] = \texttt{a}$, da das Wörterbuch zunächst das komplette Alphabet enthalten muss. $D[3] = \texttt{aa}$, da der vierte Eintrag des Wörterbuchs bereits drei Zeichen enthält und sich jeder neue Eintrag aus einem vorherigen Eintrag (hier: `aa`) und einem weiteren Zeichen (hier: `b`) zusammensetzt. $s[2] = \texttt{aa}$, da $s[3] = \texttt{b}$ und man dies für $D[4]$ benötigt. $s[4]$ in dekodierter Form muss mit einem `a` beginnen, sonst wäre $D[5] = \texttt{bb}$ und $D[6]$ müsste auch mit `b` beginnen. Damit ist $D[4] = \texttt{ba}$. Um schließlich noch $D[6] = \texttt{aabb}$ erzeugen zu können, muss $s[4] = \texttt{aab}$ sein (wie bei $D[4]$).

Aufgabe 2 (*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: $\text{border}[10] = 2,\ \text{border}[11] = 1,\ \text{border}[12] = 0$

Musterlösung:

a) Es ergibt sich folgendes *border-array*:

p	a	b	a	c	a	b	a	b	c
border[]	-1	0	0	1	0	1	2	3	2

b) Das Suchmuster wird insgesamt 4 mal angelegt:

t	a	b	a	c	a	b	a	b	b	a	b	a	c	a	b	a	b	c	
	a	b	a	c	a	b	a	b	c										Stelle 1
							a	b	a	c	a	b	a	b	c				Stelle 7
									a	b	a	c	a	b	a	b	c		Stelle 9
										a	b	a	c	a	b	a	b	c	Stelle 10

c) Das Muster muss maximal $|t| - |p| + 1$ mal angelegt werden. Dieser Fall tritt z.B. auf, falls das erste Zeichen von p nicht in t auftaucht. Das Muster muss minimal $\lceil |t|/(|p| - 1) \rceil$ mal angelegt werden. Dieser Fall tritt z.B. auf, falls p aus einer Folge paarweise unterschiedlicher Zeichen besteht und t aus Konkatenationen von p ohne das letzte Zeichen.

d) Zu zeigen oder widerlegen ist die Existenz von drei Einträgen:

$\text{border}[i] = k,\ \text{border}[i+1] = k-1, \text{border}[i+2] = k-2.$

Einträge dieser Art können nicht existieren, da in diesem Fall gelten würde:

für $\text{border}[i+0] = k-0$: $p_{k-2} = p_{i-3},\ p_{k-1} = p_{i-2},\ p_k = p_{i-1}$,
für $\text{border}[i+1] = k-1$: $p_{k-2} = p_{i-1},\ p_{k-1} = p_i$ und
für $\text{border}[i+2] = k-2$: $p_{k-2} = p_{i+1}$.

Damit, würde sich ergeben:

$p_{k-2} = p_{i-3} = p_{i-1} = p_{i+1} = p_k$,
$p_{k-1} = p_{i-2} = p_i$.

In diesem Fall wäre aber $\text{border}[i+2] = k$, da
$p_{k-2} = p_{i-1} = p_{i-3},\ p_{k-1} = p_i = p_{i-2}$ und $p_k = p_{i+1} = p_{i-1}$.
Andernfalls wäre auch $\text{border}[i+0] \neq k$.

Aufgabe 3 (*Rechnen: Burrows-Wheeler-Transformation*)

a) (*) Bestimmen Sie die Entropie von Zeichenkette $s =$ `abababab`.

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.

Musterlösung:

a) Die Entropie ist definiert als $H(s) = -\sum_{c\in\Sigma} p(c)\log_2 p(c)$ mit $p(c) = |\{s[i] : s[i] = c\}|/n$ gleich der relativen Häufigkeit, dass Zeichen c in Zeichenkette s auftaucht. Es ergibt sich:

$$\begin{aligned} H(s) &= -\left(p(a)\log_2 p(a) + p(b)\log_2 p(b)\right) \\ &= -\left(1/2\log_2 1/2 + 1/2\log_2 1/2\right) \\ &= 1 \end{aligned}$$

b) Vorgehen:

- bilde zyklische Permutationen von s (Endzeichen \$ nicht vergessen!)
- sortiere die Permutationen (\$ ist kleiner als alle anderen Zeichen!)
- lese das Ergebnis s^{BWT} aus der letzten Spalte ab

Ausführung:

```
abababababab$       $abababababab
bababababab$a       ab$abababab
abababab$abab       abab$ababab
bababab$ababa       ababab$abab
ababab$ababab       abababab$ab
babab$ababa   →     abababababab$   → s^BWT = bbbbb$aaaaa
abab$abababab       b$ababababa
bab$abababa         bab$abababa
ab$ababababab       babab$ababa
b$ababababa         bababab$aba
$abababababab       bababababab$a
```

Da die *Burrows-Wheeler-Transformation* lediglich die Zeichen der Zeichenkette permutiert, ändert sich die Entropie nicht.

c) Die kodierte Zeichenkette zu s^{BWT} lautet: $c^{BWT} = 2, 1, 1, 1, 1, 2, 1, 1, 1, 1$
Sie baut sich wie folgt auf:

Index i	Y	$s^{BWT}[i]$	$c^{BWT}[i]$
1	ab	b	2
2	ba	b	1
3	ba	b	1
4	ba	b	1
5	ba	b	1
6	ba	a	2
7	ab	a	1
8	ab	a	1
9	ab	a	1
10	ab	a	1

Entropie von c^{BWT}:

$$\begin{aligned} H(s) &= -\left(p(1)\log_2 p(1) + p(2)\log_2 p(2)\right) \\ &= -\left(2/10\log_2 2/10 + 8/10\log_2 8/10\right) \\ &\approx 0.721 \end{aligned}$$

Musterlösung:

d) Die kodierte Zeichenkette zu s lautet: $c = 1, 2, 2, 2, 2, 2, 2, 2, 2, 2$
Sie baut sich wie folgt auf:

Index i	Y	$s^{BWT}[i]$	$c^{BWT}[i]$
1	ab	a	1
2	ab	b	2
3	ba	a	2
4	ab	b	2
5	ba	a	2
6	ab	b	2
7	ba	a	2
8	ab	b	2
9	ba	a	2
10	ab	b	2

Entropie von c^{BWT}:

$$\begin{aligned} H(s) &= -\left(p(1)\log_2 p(1) + p(2)\log_2 p(2)\right) \\ &= -\left(1/10\log_2 1/10 + 9/10\log_2 9/10\right) \\ &\approx 0.4689 \end{aligned}$$

Erstaunlicherweise ist die *Move-to-Front* Kodierung von die Zeichenkette s erfolgreicher als von Zeichenkette s^{BWT} nach *Burrows-Wheeler-Transformation*. Dies kann bei einigen Spezialfällen auftreten. Eine anschließende Komprimierung von c (z.B. mit *Huffmann-Kodierung*) würde c stärker verkleinern können als c^{BWT}.

Zum Vergleich wird auf der nächsten Seite unter Teilaufgabe x) als weiteres Verfahren eine *Lempl-Ziv* Kompression von s und s^{BWT} ausgeführt und deren Entropie betrachtet.

e) Vorgehen:

- schreibe s^{BWT} in Spaltenform auf
- sortiere zeilenweise
- schreibe s^{BWT} in Spaltenform vor die bisherigen Spalten
- wiederhole bis $|s^{BWT}|$ Spalten sortiert wurden
- die oberste Zeile gibt s an

Ausführung:

```
b      $      b$     $a     b$a    $ab    b$ab
c      a      ca     ab     cab    ab$    cab$
$ sort a add  $a sort ab add  $ab sort abc add  $abc
a      b      ab     b$     ab$    b$a    ab$a
a      b      ab     bc     abc    bca    abca
b      c      bc     ca     bca    cab    bcab
```

```
      $abc       b$abc      $abca      b$abca     $abcab
      ab$a       cab$a      ab$ab      cab$ab     ab$abc
sort  abca  add  $abca sort  abcab  add  $abcab sort abcab$
      b$ab       ab$ab      b$abc      ab$abc     b$abca
      bcab       abcab      bcab$      abcab$     ccab$a
      cab$       bcab$      cab$a      ccab$a     cab$ab
```

Die gesuchte Zeichenfolge lautet $s =$ `abcab`.

Musterlösung:

x) *zusätzliche Betrachtungen: Kompression von s und s^{BWT} mit dem* Lempel-Ziv *Verfahren* (war nicht Teil der Aufgabenstellung)

Die komprimierte Zeichenkette zu s lautet: $c = 1, 2, 3, 5, 4, 2$
Das aufgebaute Wörterbuch D lautet:
$D = \{(1, \texttt{a}), (2, \texttt{b}), (3, \texttt{ab}), (4, \texttt{ba}), (5, \texttt{aba}), (6, \texttt{abab}), (7, \texttt{bab})\}$

Index	Zeichenfolge
1	a
2	b
3	ab
4	ba
5	aba
6	abab
7	bab

Entropie von c:

$$\begin{aligned} H(s) &= -\left(p(1)\log_2 p(1) + p(2)\log_2 p(2) + p(3)\log_2 p(3) + p(4)\log_2 p(4) + p(5)\log_2 p(5)\right) \\ &= -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{2}{5}\log_2\frac{2}{5} + \frac{1}{5}\log_2\frac{1}{5} + \frac{1}{5}\log_2\frac{1}{5} + \frac{1}{5}\log_2\frac{1}{5}\right) \\ &\approx 2.384 \end{aligned}$$

Die komprimierte Zeichenkette zu s^{BWT} lautet: $c^{BWT} = 2, 3, 3, 1, 6, 6$
Das aufgebaute Wörterbuch D^{BWT} lautet:
$D^{BWT} = \{(1, \texttt{a}), (2, \texttt{b}), (3, \texttt{bb}), (4, \texttt{bbb}), (5, \texttt{bba}), (6, \texttt{aa}), (7, \texttt{aaa})\}$

Index	Zeichenfolge
1	a
2	b
3	bb
4	bbb
5	bba
6	aa
7	aaa

Entropie von c^{BWT}:

$$\begin{aligned} H(s) &= -(p(1)\log_2 p(1) + p(2)\log_2 p(2) + p(3)\log_2 p(3) \\ &\quad + p(4)\log_2 p(4) + p(5)\log_2 p(5) + p(6)\log_2 p(6)) \\ &= -\left(\frac{1}{6}\log_2\frac{1}{6} + \frac{1}{6}\log_2\frac{1}{6} + \frac{2}{6}\log_2\frac{2}{6} + \frac{2}{6}\log_2\frac{2}{6}\right) \\ &\approx 1.918 \end{aligned}$$

Wie man sieht, ist die Entropie von c^{BWT} kleiner als die Entropie von c. Damit konnte der String s^{BWT} nach *Burrows-Wheeler-Transformation* besser komprimiert werden als davor. Allerdings ist die die Entropie beider komprimierter Strings größer ist als die des unkomprimierten Strings! Dieser Effekt tritt bei kurzen Strings häufig auf, da sie nicht genug Redundanz bieten.

Aufgabe 4 (*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 $C^k = \langle s_i \mid (i \bmod 3) = k \rangle$ 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}:

c) 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).

d) 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.

e) 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?

f) 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:

g) 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:

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

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

h) Erstellen Sie Tupel $(s[i], rank[i{+}1])$ f.a. $s_i \in C^0$ 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:

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

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

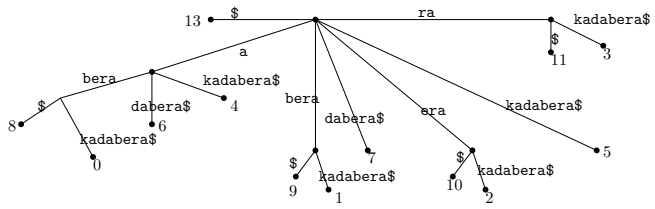
   Vergewissern Sie sich, dass C lexikographisch sortiert ist und damit das Suffixarray induziert.

Notation:
- Alle Indizes fangen bei 0 an – analog zu Kapitel 9.3.6, auf dem die Aufgabe basiert.
- $s[i\ (..j)\ ]$: 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`)

Musterlösung:
Als Referenz zunächst Zeichenkette s mit ihren Indizes (beachten Sie das Abschlusszeichen `$`!):

Index i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$s[i]$	a	b	e	r	a	k	a	d	a	b	e	r	a	$

a) Der Suffixbaum für s als kompaktierter Trie:



Jeder Weg von der Wurzel zu einem Blatt gibt einen Suffix von s an. Der Wert am Blatt gibt den Index des Suffix an (d.h. die Stelle in der Zeichenkette an der der Suffix beginnt).

b) Das Suffixarray SA für s lautet:
SA $= \langle 13, 12, 8, 0, 6, 4, 9, 1, 7, 10, 2, 5, 11, 3 \rangle$.

Index i	SA$[i]$	$s_{\mathrm{SA}[i]}$
0	13	`$`
1	12	`a$`
2	8	`abera$`
3	0	`aberakadabera$`
4	6	`adabera$`
5	4	`akadabera$`
6	9	`bera$`
7	1	`berakadabera$`
8	7	`dabera$`
9	10	`era$`
10	2	`erakadabera$`
11	5	`kadabera$`
12	11	`ra$`
13	3	`rakadabera$`

In der rechten Spalte der Suffixtabelle sind die durch das Suffixarray indizierten Suffixe aufgetragen. Man sieht, dass sie durch das Suffixarray in alphabetischer Reihenfolge geordnet vorliegen.

Musterlösung:

c) Die Tripelsequenzen lauten:

$R^1 = \langle \texttt{ber}, \texttt{aka}, \texttt{dab}, \texttt{era}, \texttt{\$\$\$} \rangle$
$R^2 = \langle \texttt{era}, \texttt{kad}, \texttt{abe}, \texttt{ra\$} \rangle$

Beachten Sie, dass das letzte Tripel von R^1 mit weiteren Abschlusszeichen \$ ergänzt wurde.

d) Die sortierten Tripel von R^{12} ohne Duplikate ergeben folgende Ränge:

Index i	4	7	1	0	2	3, 5	6	8
$R^{12}[i]$	\$\$\$	abe	aka	ber	dab	era	kad	ra\$
Rang	0	1	2	3	4	5	6	7

e) Die Ränge definieren eine eindeutige Bezeichnung der Tripel von R^{12}. Damit kann R^{12} dargestellt werden als $s^{12} = \langle 3, 2, 4, 5, 0, 5, 6, 1, 7 \rangle$.

Die Sequenz s^{12} enthält das selbe Zeichen (5) mehrfach. Ansonsten wäre über die Ränge bereits eine vollständige Sortierung von s^{12} –und damit auch von C^{12}– bestimmt. Um diese Sortierung zu erhalten, bestimmt man rekursiv mit dem DC3-Algorithmus das Suffixarray für s^{12}.

f) Das Suffixarray für R^{12} ist –nach Konstruktion– gleich dem Suffixarray für s^{12}. Es ergibt sich zu $\mathrm{SA}^{12} = \langle 9, 4, 7, 1, 0, 2, 3, 5, 6, 8 \rangle$.

Index i	$\mathrm{SA}^{12}[i]$	$s^{12}_{\mathrm{SA}^{12}[i]}$	$R^{12}_{\mathrm{SA}^{12}[i]}$
0	9	$\langle . \rangle$	$\langle \rangle$
1	4	$\langle 0, 5, 6, 1, 7, . \rangle$	$\langle \texttt{\$\$\$}, \texttt{era}, \texttt{kad}, \texttt{abe}, \texttt{ra\$}, \rangle$
2	7	$\langle 1, 7, . \rangle$	$\langle \texttt{abe}, \texttt{ra\$}, \rangle$
3	1	$\langle 2, 4, 5, 0, 5, 6, 1, 7, . \rangle$	$\langle \texttt{aka}, \texttt{dab}, \texttt{era}, \texttt{\$\$\$}, \texttt{era}, \texttt{kad}, \texttt{abe}, \texttt{ra\$}, \rangle$
4	0	$\langle 3, 2, 4, 5, 0, 5, 6, 1, 7, . \rangle$	$\langle \texttt{ber}, \texttt{aka}, \texttt{dab}, \texttt{era}, \texttt{\$\$\$}, \texttt{era}, \texttt{kad}, \texttt{abe}, \texttt{ra\$}, \rangle$
5	2	$\langle 4, 5, 0, 5, 6, 1, 7, . \rangle$	$\langle \texttt{dab}, \texttt{era}, \texttt{\$\$\$}, \texttt{era}, \texttt{kad}, \texttt{abe}, \texttt{ra\$}, \rangle$
6	3	$\langle 5, 0, 5, 6, 1, 7, . \rangle$	$\langle \texttt{era}, \texttt{\$\$\$}, \texttt{era}, \texttt{kad}, \texttt{abe}, \texttt{ra\$}, \rangle$
7	5	$\langle 5, 6, 1, 7, . \rangle$	$\langle \texttt{era}, \texttt{kad}, \texttt{abe}, \texttt{ra\$}, \rangle$
8	6	$\langle 6, 1, 7, . \rangle$	$\langle \texttt{kad}, \texttt{abe}, \texttt{ra\$}, \rangle$
9	8	$\langle 7, . \rangle$	$\langle \texttt{ra\$}, \rangle$

Beachten Sie das zusätzliche Abschlusszeichen . für s^{12}. Es ist funktional identisch zum \$ für s. Zur leichteren Unterscheidung wurde aber ein anderes Zeichen gewählt. Das Abschlusszeichen benötigt keine Entsprechung in R^{12}, daher ist hier nur ein leeres Tripel eingetragen. Es wird für die Bestimmung von SA^{12} benötigt, kann aber im weiteren Verlauf ignoriert werden.

Das Suffixarray gibt eine Sortierung der Elemente von R^{12} bzw. s^{12} und damit auch von C^{12} an (für C^{12} ist dies spätestens dann ersichtlich, wenn man in der rechten Spalte alle Tripel nach \$\$\$ entfernt und die restlichen in einer Zeile konkateniert).

g) Aus dem Suffixarray lässt sich folgende Rang-Funktion ableiten:

Index i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
$s[i]$	a	b	e	r	a	k	a	d	a	b	e	r	a	\$	\$	...
$rank[i]$	$\perp$	4	7	$\perp$	3	8	$\perp$	5	2	$\perp$	6	9	$\perp$	1	0	...

Beachten Sie, dass $rank[i] = \perp$ anstatt 0 gesetzt wurde f.a. $(i \bmod 3) = 0$. Dies ist zulässig, da diese Einträge von *rank* nie verwendet werden.

h) Es ergeben sich die folgenden Tupel (für $s_i \in C^0$ gilt $(i \bmod 3) = 0$, $i < |s|$):

Index i	0	3	6	9	12
$(s[i], rank[i+1])$	(a, 4)	(r, 3)	(a, 5)	(b, 9)	(a, 1)

Sortiert ergibt sich: $\langle (\texttt{a}, 1), (\texttt{a}, 4), (\texttt{a}, 5), (\texttt{b}, 9), (\texttt{r}, 3) \rangle$. Diese Sortierung entspricht einer Sortierung von C^0. Alle unterschiedlichen Anfangsbuchstaben der Suffixe sind korrekt sortiert. Bei gleichem Anfangsbuchstaben ist über *rank* eine korrekte Sortierung ab dem zweiten Zeichen gegeben.

Musterlösung:

i) Für das Mischen werden C^0 und C^{12} in sortierter Reihenfolge benötigt. Diese Reihenfolge wurde in den vorherigen Teilaufgaben berechnet. Für den Vergleich beim Mischen wird zusätzlich für jedes Suffix eine bestimmte Tupeldarstellung benötigt.

Für C^0 ergeben sich die sortierte Reihenfolge und die benötigten Tupel wie folgt:

Index i	s_i	$(s[i], rank[i+1])$	$(s[i..i+1], rank[i+2])$
12	a$ $\in C^0$	(a, 1)	(a$, 0)
0	aberakadabera$ $\in C^0$	(a, 4)	(ab, 7)
6	adabera$ $\in C^0$	(a, 5)	(ad, 2)
9	bera$ $\in C^0$	(b, 6)	(be, 9)
3	rakadabera$ $\in C^0$	(r, 3)	(ra, 8)

Für C^{12} sind die sortierte Reihenfolge (z.B. aus $rank$ abzulesen) und die benötigten Tupel:

Index i	s_i	$(s[i], rank[i+1])$	$(s[i..i+1], rank[i+2])$
13	$ $\in C^1$	($, 0)	
8	abera$ $\in C^2$		(ab, 6)
4	akadabera$ $\in C^1$	(a, 8)	
1	berakadabera$ $\in C^1$	(b, 7)	
7	dabera$ $\in C^1$	(d, 2)	
10	era$ $\in C^1$	(e, 9)	
2	erakadabera$ $\in C^2$		(er, 3)
5	kadabera$ $\in C^2$		(ka, 5)
11	ra$ $\in C^2$		(ra, 1)

Zusammengemischt ergibt sich C zu:

Index i	s_i	$(s[i], rank[i+1])$	$(s[i..i+1], rank[i+2])$
13	$ $\in C^1$	($, 0)	
12	a$ $\in C^0$	(a, 1)	(a$, 0)
8	abera$ $\in C^2$		(ab, 6)
0	aberakadabera$ $\in C^0$	(a, 4)	(ab, 7)
6	adabera$ $\in C^0$	(a, 5)	(ad, 2)
4	akadabera$ $\in C^1$	(a, 8)	
9	bera$ $\in C^0$	(b, 6)	(be, 9)
1	berakadabera$ $\in C^1$	(b, 7)	
7	dabera$ $\in C^1$	(d, 2)	
10	era$ $\in C^1$	(e, 9)	
2	erakadabera$ $\in C^2$		(er, 3)
5	kadabera$ $\in C^2$		(ka, 5)
11	ra$ $\in C^2$		(ra, 1)
3	rakadabera$ $\in C^0$	(r, 3)	(ra, 8)

Die Suffixe in C sind offensichtlich lexikographisch sortiert (zweite Spalte). Damit erhält man das Suffixarray für s als SA $= \langle 13, 12, 8, 0, 6, 4, 9, 1, 7, 10, 2, 5, 11, 3 \rangle$.

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

Gegeben sei die Zeichenkette $s =$ `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 SA[·] und dessen LCP-Array LCP[·] 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.

Musterlösung:

a) Das Suffixarray SA für s lautet:
SA $= \langle 10, 5, 2, 6, 7, 3, 9, 8, 4, 1 \rangle$.

Index i	SA$[i]$	$s_{\text{SA}[i]}$
1	10	`$`
2	5	`adipp$`
3	2	`alsadipp$`
4	6	`dipp$`
5	7	`ipp$`
6	3	`lsadipp$`
7	9	`p$`
8	8	`pp$`
9	4	`sadipp$`
10	1	`salsadipp$`

In der rechten Spalte der Suffixtabelle sind die durch das Suffixarray indizierten Suffixe aufgetragen. Man sieht, dass sie durch das Suffixarray in alphabetischer Reihenfolge geordnet vorliegen.

b) Das LCP-Array LCP für s lautet:
LCP $= \langle 0, 0, 1, 0, 0, 0, 0, 1, 0, 2 \rangle$.

Der Eintrag LCP$[i]$ gibt die Länge des gemeinsamen Präfixes von $s_{\text{SA}[i-1]}$ und $s_{\text{SA}[i]}$ an. Damit ist LCP[1] nicht definiert. Nach Konvention setzt man normalerweise LCP$[1] = 0$.

c) Das Suffixarray SA$[\cdot]$ enthält alle Suffixe von T lexikographisch sortiert. Damit sind die Einträge mit dem längstem gemeinsamen Präfix –und damit mit dem längsten Substring– benachbart. Da jeder Eintrag LCP$[i]$ im LCP-Array die Länge des gemeinsamen Präfixes von benachbarten Suffixen $s_{\text{SA}[i-1]}$ und $s_{\text{SA}[i]}$ angibt, genügt es, das Maximum von LCP$[\cdot]$ zu bestimmen, um die Position des längsten sich wiederholenden Substring zu erhalten.

d) Die Menge aller Substrings eines Strings ist durch die Menge aller Präfixe seiner Suffixe gegeben. In einem String mit maximal vielen unterschiedlichen Substrings besitzen keine zwei Suffixe ein gemeinsames Präfix. Damit steuert jedes Suffix s genau $|s|$ zur Menge der paarweise unterschiedlichen Substrings bei. Somit ist die gesuchte Anzahl $\sum_{i=1}^{n} |s_{SA[i]}| = \sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$.

Die Anzahl paarweiser unterschiedlicher Substrings für einen konkreten String t ist gegeben durch $\#_{pds} = \sum_{i=1}^{n} |s_{\text{SA}[i]}| - \text{LCP}[i]$. Dies folgt aus folgender Überlegung:
Sei LCP$[j] = k$. Dies bedeutet, dass $s_{\text{SA}[i-1]}$ und $s_{\text{SA}[i]}$ ein gemeinsames Präfix der Länge k besitzen. Das wiederum heißt, dass die k Suffixe dieses gemeinsamen Präfix nicht gezählt werden dürfen, da sie gemeinsam und nicht paarweise verschieden sind.

Die Länge eines Suffix lässt sich mit Hilfe des Suffix-Arrays bestimmen zu $|s_{\text{SA}[i]}| = n - SA[i]$.

e) Dies lässt sich direkt aus der vorherigen Teilaufgabe ableiten: $\frac{2}{n \cdot (n+1)} \sum_{i=1}^{n} P[i]$.

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 benutztung eines Wavelet Trees in $\log |\mathcal{U}|$ Zeit $\arg\min_i \{A[i] \mid i \in [a, b]\}$ für Parameter a, b berechnen lässt.

Musterlösung:
Sei WT der Wavelet Tree für unser Feld A.

```
1:  function RMQ(WT, a, b)
2:      if WT ist Blatt then
3:          return a
4:      end if
5:      a' ← rank_0(a)
6:      b' ← rank_0(b + 1) − 1
7:      if b' − a' > 0 then                          ▷ Minimum ist in linkem Teilbaum
8:          i' ← RMQ(WT → child_left, a', b')
9:          i ← select_0(i' + 1)
10:     else                                         ▷ Minimum ist in rechtem Teilbaum
11:         i' ← RMQ(WT → child_right, a − a', b − b' − 1)   ▷ rank_1(x) = x − rank_0(x)
12:         i ← select_1(i' + 1)
13:     end if
14:     return i
15: end function
```

Dieser Algorithmus ist leicht modifizierbar um das linkeste, rechteste oder mittlere Minimum auszugeben.