Universität Karlsruhe (TH)
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders, Dr. Rob van Stee
G. Veit Batz, Johannes Singler

6. Übungsblatt zur Algorithmentechnik WS 2007/08

http://algo2.iti.uni-karlsruhe.de/algotech.php
{sanders|vanstee|batz|singler}@ira.uka.de

Aufgabe 1 (*Suffix-Arrays und DC3-Algorithmus, Schwierigkeit* $1+2+2+2+2+2+1+2+2$)
Gegeben sei die Zeichenkette $S := \langle \texttt{mississippi.} \rangle$.

a) Geben Sie das Suffix-Array zu S an.

Zur Berechnung von Suffix-Arrays wurde in der Vorlesung ein rekursiver Algorithmus vorgestellt (er wird als DC3-Algorithmus bezeichnet). In den folgenden Teilaufgaben sollen die einzelnen Schritte dieses Algorithmus von Hand durchgeführt werden.

b) Geben Sie die Tripelsequenzen $R_k := \big\langle S[i..i+2] \;\big|\; (i \bmod 3) = k \big\rangle$ an, für $k \in \{1,2\}$. Dabei sei $S[i] = 0$ für $i \geq |S|$.

c) Sortieren Sie die Tripel in der Verkettung $R_1 \circ R_2$ mit LSD-Radix-Sort, entfernen Sie (unter Beibehaltung der Reihenfolge) alle mehrfachen Tripel und ordnen Sie dann jedem Tripel von links nach rechts beginnend bei 1 seine Position in der neu entstandenen Folge zu. So erhalten Sie ein ganzzahliges Alphabet.

d) Stellen Sie nun die Verkettung $R_1 \circ R_2$ mit Hilfe des Alphabets aus Teilaufgabe c) dar. Auf diese Weise erhalten Sie die Zeichenkette S^{12}. Führt der DC3-Algorithmus für dieses S^{12} eine Rekursion aus?

e) Geben Sie das Suffix-Array SA^{12} für S^{12} von Hand an (gleichgültig, ob der DC3-Algorithmus nun eine Rekursion durchführen würde oder oder nicht).

Sei $\mathcal{C}_k := \{S_i \mid 0 \leq i \leq n \text{ und } (i \bmod 3) = k\}$ und $\mathcal{C}_{12} := \mathcal{C}_1 \cup \mathcal{C}_2$.

f) Jedes Suffix S_i^{12} von S^{12} repräsentiert genau ein Suffix aus $\mathcal{C}_{12}$, nämlich das Suffix $S_{\varphi(i)}$ mit

$$\varphi : i \mapsto \begin{cases} 3i+1 & \text{für } 0 \leq i < \lceil \frac{n+2}{3} \rceil \\ 3i - \lceil \frac{n+2}{3} \rceil & \text{für } \lceil \frac{n+2}{3} \rceil \leq i < \lceil \frac{n-1}{3} \rceil + \lceil \frac{n-2}{3} \rceil \end{cases} .$$

Geben Sie alle Paare $S_i^{12} \mapsto S_{\varphi(i)}$ an.

Sei $A := \langle a_0, \ldots, a_k \rangle$ ein Array mit paarweise verschiedenen Einträgen. Dann liefert $ind_A(a_i) := i$ den Index von a_i in A. Werde weiter $S_j \in \mathcal{C}_{12}$ von S_i^{12} repräsentiert (wie in Teilaufgabe f) beschrieben). Dann heiße $r(S_j) := ind_{SA^{12}}(\varphi^{-1}(j))$ der Rang von S_j (dabei sei $r(S_{n+1}) := r(S_{n+2}) := 0$). Offenbar ist der Rang nur definiert für $(j \bmod 3) \in \{1,2\}$.

g) Schreiben Sie S auf. Annotieren Sie für alle $S_j \in \mathcal{C}_{12}$ jeweils $S[j]$ mit $r(S_j)$.

Durch diese Annotation liegt jetzt ein lexikographische Sortierung der Suffix-Menge $\mathcal{C}_{12}$ vor.

h) Jedes $S_i \in \mathcal{C}_0$ entspricht in natürlicher Weise dem Paar $(S[i], r(S_{i+1}))$. Sortieren Sie alle Paare $(S[i], r(S_{i+1}))$ mit $S_i \in \mathcal{C}_0$ mittels LSD-Radix-Sort. Liefert dies eine lexikographischen Sortierung aller $S_i \in \mathcal{C}_0$?

Es liegen jetzt *beide* Suffix-Mengen $\mathcal{C}_0$ und $\mathcal{C}_{12}$ in lexikographisch sortierter Form vor. Daher können Sie nun aus $\mathcal{C}_0$ und $\mathcal{C}_{12}$ durch vergleichsbasiertes Mischen eine lexikographische Sortierung *aller* Suffixe von S zu erzeugen, wobei die Vergleichsrelation $\leq'$ verwendet werden kann und soll:

$$S_i \leq' S_j \quad :\Longleftrightarrow \quad \begin{cases} \big(S[i], r(S_{i+1})\big) \leq_{lex} \big(S[j], r(S_{j+1})\big) & \text{für } S_i \in \mathcal{C}_1 \\ \big(S[i], S[i+1], r(S_{i+2})\big) \leq_{lex} \big(S[j], S[j+1], r(S_{j+2})\big) & \text{für } S_i \in \mathcal{C}_2 \end{cases}$$

Dabei sei $S_i \in \mathcal{C}_{12}$, $S_j \in \mathcal{C}_0$ und $\leq_{lex}$ die lexikographische Ordnung.

i) Schreiben Sie alle Suffixe $S_j \in \mathcal{C}_0$ in der bereits bekannten lexikographischen Sortierung auf und schreiben Sie zu jedem S_j sowohl das 2-Tupel $(S[j], r(S_{j+1}))$ als auch das 3-Tupel $(S[j], S[j+1], r(S_{j+2}))$ dazu. Schreiben Sie auch die Suffixe $S_i \in \mathcal{C}_{12}$ in der bereits bekannten lexikographischen Sortierung auf. Schreiben Sie wieder zu allen Suffixen S_i die entsprechenden Tupel dazu, aber diesmal nur das 2-Tupel (falls $S_i \in \mathcal{C}_1$) oder das 3-Tupel (falls $S_i \in \mathcal{C}_2$). Führen Sie nun das Mischen durch. Was ist der Vorteil, wenn man die Vergleichsrelation $\leq'$ verwendet?

Aufgabe 2 (*Multikey Quicksort, Schwierigkeit* $2 + 2 + 2$)

Sei S eine Sequenz von Strings (Zeichenketten). Nehmen Sie der Einfachheit halber an, dass jeder String mit einem speziellen Zeichen # endet, das sich von allen „normalen“ Zeichen unterscheidet, und als „kleiner“ als alle anderen Zeichen verglichen wird.

a) Zeigen Sie, dass der Aufruf *MultiKeyQuicksort*(S,1) eine Sequenz S *paarweise verschiedener* Strings lexikographisch aufsteigend sortiert. Der Operator $[i]$ wählt das i-te Zeichen eines Strings, wobei die Zählung bei 1 beginnt.

```
1: function MultiKeyQuicksort(S: Sequence of Strings, i: ℕ)
2: assert ∀e, e' ∈ S : e[1..i − 1] = e'[1..i − 1]
3: if |S| < 1 then
4:    return S
5: end if
6: pick p ∈ S uniformly at random
7: return concatenation of
       MultiKeyQuicksort(⟨e ∈ S : e[i] < p[i]⟩, i),
       MultiKeyQuicksort(⟨e ∈ S : e[i] = p[i]⟩, i + 1), and
       MultiKeyQuicksort(⟨e ∈ S : e[i] > p[i]⟩, i)
```

b) Was geht schief, wenn S gleiche Strings enthält? Lösen Sie dieses Problem.

c) Zeigen Sie, dass die Summe der Anzahl Elemente aller Mengen des mittleren Falls in $\mathcal{O}(N)$ ist, wobei $N = \sum_{e \in S} |e|$. Was folgt daraus für die Gesamtlaufzeit?

Aufgabe 3 (*Maximale Flüsse: Theorie, Schwierigkeit* $3 + 2 + 2$)

Definition. Eine Teilmenge der Knoten in einem Netzwerk, die s aber nicht t enthält, nennt man einen Schnitt. Die Kapazität eines Schnittes ist die Summe der Kapazitäten der aus dem Schnitt herausführenden Kanten. Ein minimaler Schnitt ist ein Schnitt mit minimaler Kapazität. □

Satz. Der Wert eines maximalen Flusses ist gleich der Kapazität eines minimalen Schnitts. □

Gegeben sei ein Netzwerk (V, E).

a) Geben Sie einen Algorithmus an, der zu einem gegebenen maximalen Fluss von s nach t einen minimalen Schnitt in nur $\mathcal{O}(|E|)$ Zeit findet.

b) Zeigen Sie: Falls eine Kante (i, j) für jeden maximalen Fluss voll ausgelastet ist, dann führt diese Kante aus einem minimalen Schnitt heraus.

c) Angenommen, Sie möchten einen ungerichteten Graphen in zwei Komponenten zerteilen, sodass die Anzahl Kanten zwischen den Komponenten minimal ist. Wie lösen Sie dieses Problem?