

Universität Paderborn
Institut für Informatik
Prof. Dr. Stefan Böttcher

Proseminar Datenkompression – WS 2016/2017

Linear-Time Suffix-Sorting

Clemens Damke

Matrikelnummer 7011488

Inhaltsverzeichnis

1 Problemstellung	5
1.1 Was ist ein Suffix-Array?	5
1.2 Einsatzgebiete von Suffix-Arrays	5
2 Ansätze zur Suffix-Array-Konstruktion	6
2.1 Naiver Ansatz	6
2.2 Überblick über bisherige Linearzeitansätze	6
3 Der GSACA-Algorithmus	7
3.1 Grundidee	7
3.1.1 Induziertes Sortieren	7
3.1.2 Definitionen	7
3.1.3 Die zwei Phasen von GSACA	7
3.2 Phase 1	7
3.3 Phase 2	7
4 Performanceanalyse	7
5 Fazit	7
Literaturverzeichnis	7



1 Problemstellung

Diese Proseminar-Arbeit beschreibt den GSACA-Algorithmus. Hierbei handelt es sich um den ersten rekursionsfreien Linearzeitalgorithmus zur Konstruktion von Suffix-Arrays.

Im Folgenden wird zunächst erörtert, was Suffix-Arrays sind und wozu sie benutzt werden.

1.1 Was ist ein Suffix-Array?

Das Suffix-Array SA einer Zeichenkette S ist definiert als die lexiographisch aufsteigend sortierte Folge aller Suffixe von S .

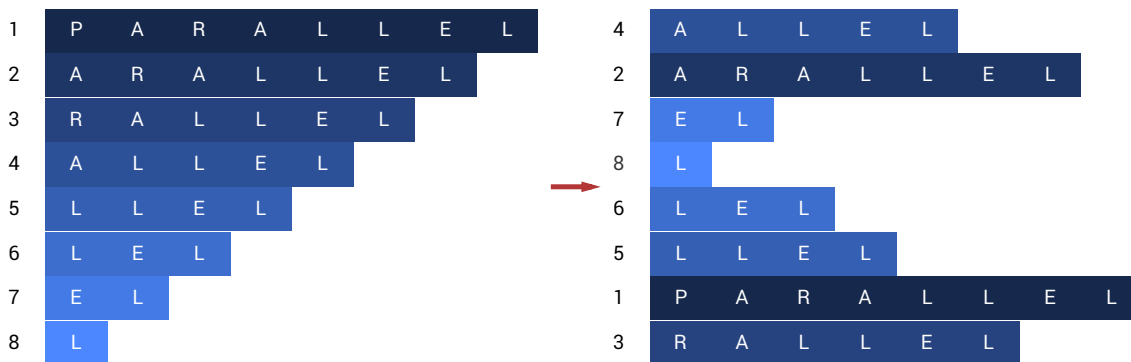


Abbildung 1: Suffixarray für $S = \text{'parallel'}$

Um Speicher zu sparen wird SA allerdings nicht als Folge der Suffix-Zeichenketten, sondern als Folge der Startpunkte der Suffixe repräsentiert. Für $S = \text{'parallel'}$ wäre also $SA = (4, 2, 7, 8, 6, 5, 1, 3)$. Formal bedeutet dies:

$$\begin{aligned}
 \Sigma &:= \text{streng total geordnetes endliches Alphabet} \\
 S &:= \text{Eingabezeichenkette} = (S[1], \dots, S[n]) \in \Sigma^n, |S| := n \in \mathbb{N} \\
 S[i..j+1) &= S[i..j] := (S[i], \dots, S[j]) \\
 S_i &:= S[i..n] \\
 S \sqsubseteq T &:\Leftrightarrow S = T[1..|S|] \\
 S <_{lex} T &:\Leftrightarrow (\exists i : S[i] < T[i] \wedge S[1..i) = T[1..i)) \vee (|S| < |T| \wedge S \sqsubseteq T) \\
 SA &:= \text{Permutation von } \{1, \dots, |S|\}, \text{ sodass } \forall i < j : S_{SA[i]} <_{lex} S_{SA[j]}
 \end{aligned}$$

1.2 Einsatzgebiete von Suffix-Arrays

Suffix-Arrays finden in vielen Bereichen als Index-Datenstruktur Verwendung. Ein typisches Problem, dessen Lösung durch Suffix-Arrays beschleunigt werden kann, ist z. B. die Substringsuche. Bei dieser soll bestimmt werden, *ob* und wenn ja, *wo* in einem Text T ein Pattern P vorkommt. Ohne einen Index benötigt dieses Problem z. B. mit Knuth-Morris-Pratt $\mathcal{O}(|T| + |P|)$. Mit einem Suffix-Array als Index über T hingegen lassen sich



Matches durch eine binäre Suche in $\mathcal{O}(|P| \log |T|)$ finden. Da i. d. R. $|P| \ll |T|$, ist dies ein deutlicher Speedup, welcher z. B. in Datenbanksystemen für Volltextsuchen und in der Bioinformatik für das Suchen in DNA-Daten nützlich ist.

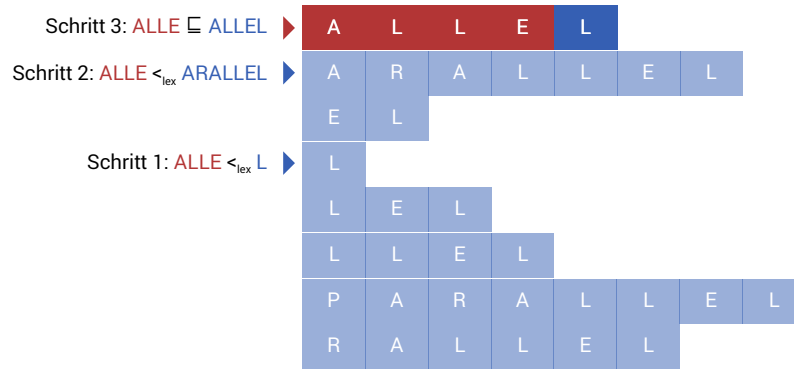


Abbildung 2: Substringsuche mit $P = \text{'alle'}$ und $T = \text{'parallel'}$

Ein weiteres Einsatzgebiet für Suffix-Arrays ist als Suchstruktur für das Sliding Window in Implementationen des LZ77-Kompressionsalgorithmus.

2 Ansätze zur Suffix-Array-Konstruktion

Nachdem nun der Begriff des Suffix-Arrays definiert wurde, wird im Folgenden betrachtet wie sich dieses prinzipiell algorithmisch berechnen lässt.

2.1 Naiver Ansatz

Da es sich bei der Suffix-Array-Berechnung im Wesentlichen um ein Sortierproblem handelt, liegt die Idee nahe dies mit einem allgemeinen Sortiervorgehen zu lösen. Dazu bietet sich z. B. der Quicksort-Algorithmus an. Im average case wären dann $\mathcal{O}(n \log n)$ Vergleiche notwendig. Für den lexiographischen Vergleich zweier Suffixe müssen wiederum bis zu $\mathcal{O}(n)$ Zeichen miteinander verglichen werden. Insgesamt ergibt sich also eine Laufzeit von $\mathcal{O}(n^2 \log n)$. Dies ist weit von der angestrebten $\mathcal{O}(n)$ -Laufzeit entfernt. Allgemeine Sortiervorgehen sind daher für die Suffix-Array-Konstruktion unbrauchbar.

2.2 Überblick über bisherige Linearzeitansätze

Es sind bereits zahlreiche Linearzeitalgorithmen zur Konstruktion von Suffix-Arrays bekannt. Allerdings sind all diese Verfahren rekursiv. Das bedeutet, dass sie neben der Eingabe und evtl. Hilfsdatenstrukturen zudem mindestens $\mathcal{O}(\log n)$ Speicher für die Stackframes der rekursiven Aufrufe benötigen.

Zwei dieser rekursiven Linearzeitalgorithmen sind der Algorithmus von Skew und der SA-IS-Algorithmus. Skew ist primär wegen seiner Kompaktheit und Eleganz interessant. SA-IS basiert auf dem Konzept der induzierten Sortierung und gehört zu den

schnellsten bekannten SACAs (suffix array construction algorithms).

	Skew	SA-IS	GSACA
Art	rekursiv	rekursiv	iterativ
Zeit	$O(n)$	$O(n)$	$O(n)$
Speicher	$O(\log n) + \max 24n$	$O(\log n) + \max 2n$	$O(1) + ?$

Tabelle 1: Vergleich von Skew, SA-IS und GSACA

Im Rest dieser Arbeit wird es um den GSACA-Algorithmus gehen, welcher der erste bekannte rekursionsfreie Linearzeit-SACA ist. Skew und SA-IS werden dabei als Referenzalgorithmen dienen, mit denen GSACA verglichen wird.

3 Der GSACA-Algorithmus

3.1 Grundidee

3.1.1 Induziertes Sortieren

3.1.2 Definitionen

3.1.3 Die zwei Phasen von GSACA

3.2 Phase 1

3.3 Phase 2

4 Performanceanalyse

test

5 Fazit

test

Literatur

