Seminar: Algorithmen für große Datenmengen (SS 01)
Einsatz von Suffix Arrays in Datenbanken

Christian Diehl
chrdiehl@studcs.uni-sb.de

30. Juli 2001

1 Vorwort

In den folgenden Abschnitten werden die Vor- und Nachteile, sowie die Effizenz von Suffix Arrays in Datenbanken aufgezeigt. Dabei werden wir drei Datenbanktypen untersuchen:

- Datenbanken in WWW-Suchmaschinen
- Volltextdatenbanken in Bibliotheken
- Gendatenbanken

Es werden jeweils die Probleme herausgestellt und falls vorhanden existierende Systeme vorgestellt.

2 Suffix Array in WWW-Suchmaschienen

Bevor ich auf den Einsatz von Suffix Arrays in Web-Suchmaschinen eingehe, möchte ich den Aufbau einer solchen zunächst kurz vorstellen.

Eine Suchmaschine besteht aus mehreren Komponenten. Die Suche auf dem Web beginnt mit einem Crawler. Dieser lädt die WWW-Seiten und sendet sie an den Storeserver. Dieser komprimiert und speichert die Seiten in einer Datenbank, die Repository[1] genannt wird. Jede dieser Web-Seiten hat eine ID, die so genannte Doc-ID. Die Indexierung wird durch den Indexer und Sorter vorgenommen, wobei der Indexer mehrere Funktionen übernimmt. Er ließt das Repository, dekomprimiert das Dokument und parst die enthaltenen Elemente. Parsen bedeutet in diesem Fall, daß eine Webseite so zusagen in ihre Bestandteile zerlegt wird. Das heißt also, daß eine Webseite in Bilder, Texte und Links unterteilt wird. Jedes Dokument wiederum wird in eine Menge von Wortvorkommen konvertiert, den sogenannten Hits. Diese speichern das Wort, die Position im Dokument und weitere Größen. Diese Hits werden dann ihrerseits in Barrels abgelegt und daraus wird ein, nach Doc-ID sortierter Forward Index erzeugt. Der Forward Index enthält die Doc-IDs und die dazu gehörigen Word-IDs. Das bedeutet also, daß in den Barrels jeweils die Dokumente und dessen enthaltene Wörter gespeichert sind. Der Sorter nimmt nun die Barrels und sortiert diese nach Ihrer Word-ID um, daraus wird dann der Inverted Index generiert. Dieser Inverted Index enthält nun für jedes in einem HTML-Dokument existierende Wort Zeiger auf die entsprechenden Barrels, in denen das Wort vorkommt. So entsteht eine Liste von Doc-IDs. Diese Liste repräsentiert alle Vorkommen eines Wortes in allen HTML-Dokumenten. Gleichzeitig erzeugt der Sorter eine WordID-Liste. Anschließend entsteht aus dieser Liste und dem zuvor aus dem Indexer erzeugten Lexikon[2] ein neues Lexikon. Dieses wird dann dazu verwendet angestellte Queries zu bearbeiten. Nähere Informationen zum Aufbau einer WWW-Suchmaschine in [4].

Wenn man sich den Aufbau einer Suchmaschine also näher betrachtet stellt man fest, daß man ein Suffix Array direkt im Indexer verwenden könnte, um die Speicherung der Hits zu übernehmen. Dort wird jeweils das Wort selbst und die Position im Text gespeichert.

Der Einsatz eines Suffix Arrays bietet weitere Vorteile, beispielsweise wenn es in einer Suchmaschine zum Einsatz kommt. In heutigen

[1]Repository beinhaltet alle traversierten HTML-Dokumente in komprimierter Form.
[2]Lexikon beinhaltet alle vorkommenden Stammformen jeweils mit Term-ID.

Suchmaschinen werden so geannte Stoppwörter während der Indizierung gelöscht. Dazu gehören the, to, also, and, or, um nur Einige zu nennen. Würde man nun z.B. nach „to be or not to be“ suchen würde man kein vernünftiges Ergebnis erhalten, da außer „be“ jedes Wort der Anfrage ein Stoppwort ist. Mittels des Suffix Arrays könnte man also komplette Substrings in Webseiten suchen. Außerdem müßte man nicht komplizierte Anfragen stellen, z.B. mit der Bedingung, daß die gesuchten Worte dicht beieinander stehen müssen und dergleichen.

Andererseits birgt ein Suffix Array aber auch viele Nachteile in sich. Unter der Annahme ein Suffix Array wäre in einer Suchmaschine implementiert und über alle HTML-Dokumente würde ein einziges Suffix Array gebilden, so müßte bei jeder Aktualisierung einer Seite jeweils das gesamte Suffix Array neu erzeugt werden. Da wegen der Übersichlichkeit der Webseiten meist kurze, aber dafür viele Webseiten erzeugt werden, müssen dementsprechend viele, d.h. mehrere Millionen Webseiten gelesen werden. Da die Seiten in Suchmaschinen aber nur selten aktualisiert werden, kann man diesen Fall vernachlässigen. Betrachtet man jedoch die Speichergröße eines solchen Arrays würde auch die Konstruktion dementsprechend lange dauern. Der zur Zeit schnellste Algorithmus für den internen Aufbau eines Suffix Arrays benötigt nach Manber und Myers O(N log_2 N) Konstruktionszeit [2][3]. Hierbei wäre N die Summe der Buchstaben aller Webseiten. Da dies eine enorme Speichermenge darstellt, hat man nach einer Lösung gesucht, die zu indexierende Datenmenge einzuschränken. Man entwickelte den Word-Index. Hierbei werden nicht mehr jeder Buchstabe, sondern jeweils die Startposition des gesamten Wortes indexiert. Den interessierten Leser verweise ich hier auf [2] Kapitel 5.6. Dennoch ist die Datenmenge derartig Groß, daß eine Suffix-Array-Konstruktion nicht rentabel ist. Um nochmals die Speicherfrage aufzugreifen, wir wissen, daß ein Suffix Array aus Zeigern auf Textpositionen besteht. Hierbei benötigt jeder Zeiger 4 Byte Speicherplatz, so daß man zu je einem Byte Text jeweils 4 Byte Index speichern müßte, also 400% zusätzlichen Speicherplatz benötigen würde. Sucht ein Benutzer nun nach einem bestimmten Wort oder Substring muß das Gesamt-Suffix-Arrays durchsucht werden. Somit würde sich also in etwa folgende Suchzeit ergeben:

$$O(log_2 M) I/Os$$

Wobei M die Summe der Länge[3] aller Webseiten darstellt.

Aber was ist eigentlich der Sinn einer WWW-Suchmaschine?

Möchte ein Benutzer wirklich alle Vorkommen des gesuchten Wortes als Resultat erhalten?

Man stelle sich einmal vor, man gebe als Suchbegriff „Bill Clinton“ in eine Suchmaschine. Möchte man jetzt wirklich weltweit alle Webseiten angezeigt bekommen, in denen dieser Name vorkommt? Nein. Im Allgemeinen sucht der Benutzer ein zur seiner Anfrage passendes Themengebiet. Also ist die Aufgabe einer Suchmaschine, die Suchanfrage einer Kategorie zuzuordnen bzw. den Inhalt einer Anfrage wiederzugeben. Dazu werden bereits im Voraus die gescannten Wörter aus dem Indexer in Klassen eingeteilt. Die Suchmaschine muß also in der Lage sein dem Suchstring „Bill Clinton“ die Kategorie „Präsident der Vereinigten Staaten“ oder „Weisses Haus“ oder ähnliches zuzuordnen. Weiterhin muß es ihr möglich sein, den Inhalt einer Anfrage zu erkennen und dem Benutzer die entsprechenden Resultate zurückzuliefern. Weiterhin ist das Internet ein globales Netzwerk, in dem es mehrere Sprachen und somit auch unterschiedliche Schreibweisen für bestimmte Begriffe gibt. Darüber hinaus wäre es nicht möglich, Webseiten mit falsch geschriebenen Begriffen ausfindig zu machen.

Allein aus der Aufgabenstellung an die Suchmaschine ist also klar, daß ein Suffix Arrays für den Einsatz in einer WWW-Suchmaschine ungeeignet ist. Aus diesem Grund gibt es bisher auch keine existiertenden Systeme, die ein Suffix Array als Indexer benutzen.
Dennoch wird versucht die Indizierung von Texten schnell und platzsparend zu realisieren. P. Ferragina und G. Manzini haben diesbezüglich einen FM-Index[4] entwickelt. Dieser kombiniert die Komprimierung eines Textes mit dessen Indexierung. Der benötigte Speicherplatz ist eine Funktion der Entropy der zu indizierenden Daten.

[3]Länge einer Seite = #Buchstaben, die die Seite enthält.
[4]Full-text index in Minute space

Dabei wird neben einem Suffix Array eine zusätzliche Matrix verwendet, so daß ein Text T$[1, n]$ in $O(H_k(T)) + o(1)$ Bit pro eingebenem Symbol gespeichert werden kann. Dieser FM-Index kann die Vorkommen von Wörtern im komprimierten Originaltext zählen und lokalisieren an welcher Textstelle die Wörter stehen. Mit Hilfe der implementierten Matrix ist es möglich, daß immer nur kleine Textstücke des komprimierten Files betrachtet werden müssen. In einem Versuch stellte sich heraus, daß ein File mit mehreren Megabyte Speicherplatz in wenigen Millisekunden durchsucht werden konnte. Der Vorteil liegt hier hauptsächlich in dem noch kleineren Speicherplatz, da das Originalfile nun in komprimierter Form vorliegt. Genauere Informationen hierzu sind in [5] zu finden.

Auf dieser Idee aufbauend entwickelte U. Manber und Sun Wu [6] einen Algorithmus der als „Glimpse[5]" bezeichnet wird. Dieser Algorithmus unterstützt einerseits das Indizieren von Texten. Andererseits ermöglicht er eine vollwertige Textsuche, wozu er allerdings lediglich ca. 2 - 4 % Speicherplatz des Originaltextes benötigt. Innerhalb eines Textes können nun Volltextsuchen, Boolsche Suchanfragen, reguläre Ausdrücke sowie Ähnlichkeitsanfragen durchgeführt werden. Die Idee, die dahinter steckt, ist nicht mehr die genaue Position jedes Wortes, sondern lediglich einen Zeiger auf den Bereich des Textes zu speichern, indem es vorkommt. Dazu wird der gesamte Text in gleichgroße Teilstücke, so genannte Blocks geteilt. Nun wird der gesamte Text Wort für Wort gescannt. Zum Vergleich, in einem Inverted Index wird jedes Wortvorkommen und deren Position berücksichtigt. In diesem neuen Verfahren, wird auch jedes Wort indiziert, jedoch nicht jede Position, sondern es wird lediglich ein Zeiger auf den aktuell gesannten Block gespeichert. Das heißt also, daß ein Wort nur einmal indiziert wird, selbst wenn es mehrfach in einem Textstück vorhanden ist. Da jeder Block mit einem Byte repräsentiert werden kann[6] , bedeutet dies einen enormen Speicherplatzgewinn. Wird nun eine Anfrage gestellt, wird zunächst der Index durchsucht, der eine Liste von Blöcken zurückgibt. Anschließend wird jeder Block einzeln nach dem gesuchten Wort durchsucht. Es ist klar, daß die Antwortzeit länger ist als bei einem Inverted Index, da es vorkommen kann, daß sehr viele Blöcke durchsucht werden

[5] GLobal IMPlicit SEarch

[6] Unter der Annahme, daß der gesamte Text in maximal 256 Blöcke unterteilt wird, kann jeder Block mit 8 Bit, also ingesant einem Byte adressiert werden.

müssen. Für einzelne Benutzer ist die Antwortzeit dieses Algorithmus aber durchaus zufriedenstellend. Betrachtet man jedoch eine Suchmaschine mit mehreren Millionen Anfragen pro Tag ist eine Antwortzeit, wie die im folgenden Beispiel nicht tragbar. Dennoch ist die Antwortzeit für die meisten Einzelplatz-Applikationen schnell genug.
In einem Versuch wurde, um auf 4300 Files mit insgesamt 69 MByte Speicherplatz die 19 Vorkommen der Wörter „Usenix“ AND „Winter“ zu finden gerade einmal 5 Sekunden benötigt. Die Größe des Index betrug 1,9 MByte, das sind ca. 2,7% der Originalgröße.
Je nach Einsatz eines Suffix Array muß man also abgewägen, ob eine schnelle Antwortzeit nötig ist und somit mehr Speicherplatz benötigt wird oder ob wegen Speichermangels eine längere Antwortzeit in Kauf genommen werden kann.

3 Recherche in Volltextdatenbanken

Betrachten wir nun den Einsatz in einer Bibliotheksdatenbank. Es ergeben sich zwei Möglichkeiten für die Konstruktion eines Suffix Arrays. Zum einen kann man für jedes Buch ein Suffix Array implementieren. Dies wäre jedoch im Bezug auf die spätere Suche sehr ineffizient, da dann soviele verschiedene Suffix Arrays durchsucht werden müßten, wie es Bücher gibt. Andererseits kann man aber auch ein Gesamtarray für alle Texte/Bücher erzeugen. Der Nachteil in einem großen Suffix Array liegt jedoch im Aktualisieren[7]. Nehmen wir an, - was in einer Bibliothek häufig vorkommt - daß ein neues Buch hinzugefügt werden muß. Da nur ein Gesamtarray existiert, muß ein neues Suffix Array über alle Bücher erzeugt werden. Dabei kann man natürlich verschiedene Verfahren verwenden, aber dennoch würde es viel Zeit in Anspruch nehmen. Ein großes Suffix Array hätte bei einer Suche natürlich viele Vorteile. Würde man z.B. nach einem Textausschnitt wie „to be or not to be“ suchen, würde man mit 100%-iger Genauigkeit auf die Literatur von Shakespeare stoßen. Ebenso hat es nun den Vorteil, daß man zusammenhängende Textpassagen sucht und nicht, wie in einer Suchmaschine nur einzelne Worte, die durch eine boolsche Verknüpfung miteinander

[7] Aktualiesieren bedeutet nicht innerhalb einer Buchseite zu korrigieren, sondern neue Bücher hinzuzufügen und ältere Bücher zu entfernen.

verbunden sind. Erinnert man sich nur noch an eine Textstelle, hat den Titel aber vergessen, so findet man mit Hilfe des Suffix Arrays schnell den gesuchten Text. Es stellt sich jedoch die Frage, wie oft eine solche Suche notwendig ist. Vielmehr sucht man nach Titeln, Autoren, Verlagen und ISBN-Signaturen. Außerdem darf man auch in diesem Fall den Speicherplatz nicht vergessen. Es wird auch hier jeweils der 4-fache Speicherplatz benötigt. Der Text selbst kann und soll natürlich digital erfaßt werden, aber für alle Texte/Bücher ein großes Suffix Array zu erzeugen ist unter Berücksichtigung der Häufigkeit der Benutzung uninteressant.

Eine interessante Alternative ergibt sich im Bereich von wissenschaftlichen Papieren. Diese enthalten am Anfang meist einen „Abstract“. Diesen Abstract könnte man zusätzlich dem Titel des Papers hinzufügen und dann auf diesem ein Suffix Array bilden bzw. ein Suffix Array über die Abtracts aller Papers[8] generieren. Würde man dann z.B. nach „Suffix Array“ suchen, würde nicht wie zuvor nach „Suffix“ AND „Array“ gesucht sondern nach einem String „Suffix Array“, in dem sogar das Leerzeichen in der Mitte berücksichtigt würde. Somit würde der Speicherverbrauch gering gehalten was sowohl die Treffergenauigkeit als auch die benötigte Suchzeit enorm optimieren würde.

Die Suchzeit würde im Falle eines Suffix Arrays für alle Texte/Bücher wie folgt aussehen:

$$O(\log_2 M) I/Os$$

Wobei M die Summe über die Länge[9] der Seiten aller indizierten Texte/Bücher darstellt.

4 Suffix Array in Gendatenbanken

Den letzten Datenbanktyp den wir hier betrachten ist der der Gendatenbank. Eine Gendatenbank besteht aus mehreren aneinanderhängenden Symbolen. Dies können Buchstaben, Ziffern oder ähnliches sein. Man hat festgestellt, daß sich Gene bei Menschen und Tieren nur

[8]Da die Länge der Abstracts von wissenschaftlichen Papieren im Vergleich zur Anzahl von Bücherseiten gering ist, ist ein Suffix Array über die Abstracts aller Paper in diesem Fall durchaus sinnvoll.

[9]Länge einer Seite = #Buchstaben, die die Seite enthält.

geringfügig unterscheiden. So ist das menschliche Gen nur eine Mutation des Mäusegens und umgekehrt. Eine kleine Veränderung in einem Gen zieht jedoch eine Reaktionskette mit sich, so daß sich Millionen von Teilgenen ändern können. Der Vorteil eines Suffix Arrays auf einer solchen Datenbank ist natürlich das exakt gleiche Gen aus vielen Millionen Teilstücken herauszufinden. Im biologischen Bereich ist der Speicherplatz weniger relevant als die Zeit für das Ergebnis. Wie eben bereits angesprochen ergeben sich aber Probleme, ein mutiertes Gen in einer solchen Gendatenbank zu finden. Aus diesem Grund wurde von S. Burkhardt, et al. [7] QUASAR (q-gram Based Database Searching Using a Suffix Array), ein Algorithmus entwickelt, mit dem es möglich ist mutierte Genteilstücke auf Datenbeständen zu finden. Zunächst wird der Datenbestand D in gleich große Blockgrößen aufgeteilt. Die Gensequenzanfrage wird in eine bestimmte Anzahl gleich langer Teilsuchsequenzen zerlegt. Dabei sucht man nach allen Vorkommen der q-grams in der Datenbank und zählt für jeden Block die Anzahl der q-grams und deren Vorkommen in diesem Block. An diesem Punkt angelangt, erlaubt uns eine gut gewählte Konstante für das Vorkommen der q-grams, daß auffinden der Blöcke, die relativ ähnlich zu der Anfragesequenze sind. Pro Block wird ein Zählarray verwendet, um den Zählprozeß schneller zu machen. Ein Suffix Array wird dabei als platzsparende Struktur benutzt, um schnell die Position zu erhalten, an der das gesuchte q-gram in der Datenbank vorkommt.
Die Konstruktion des Suffix Arrays und das Vorberechnung der Sucharrays ist in $O(D \cdot log_2\ D)$ zu erreichen.
Das Suchen nach einem q-gram benötigt konstante Zeit, das Zählen der Vorkommen geht in $|D|$. So ergibt sich bei $O(|S|)$ q-grams eine Zeit von $O(|S| \cdot |D|)$ zum Suchen. Schließlich ergibt sich eine Gesamtzeit von:

$$O(|D| \cdot \log_2 |D|) I/Os$$

Wobei $|D|$ den Datenbestand in der Datenbank darstellt.

5 Zusammenfassung

Abschließend läßt sich konstatieren, daß der Einsatz eines Suffix Arrays in den Bereichen Zustimmung findet, in denen innerhalb unstrukturierter Texte gesucht werden muß, wie beispielsweise einer Gendatenbank. Außerdem darf der Platzbedarf nicht außer acht gelassen werden. Wie man gesehen hat, benötigt ein Suffix Array zusätzlich zum Originaltext den 4-fachen Speicherplatz. Aus diesem Grund muß man abwägen, ob der zusätzlich aufgebrachte Speicher auch am Ende die erwartete Zeitverbesserung mit sich bringt. Des Weiteren stellt sich die Frage, ob der Aufbau eines Suffix Arrays überhaupt notwendig ist. Hierzu vergleiche man den Einsatz in Bibliotheksdatenbanken. Die Suche nach bestimmten Textpassagen auf einer Bibliotheksdatenbank wird meiner Meinung nach so selten sein, daß sich der eingestzte Speichplatz für das erzeugte Suffix Array einfach nicht bezahlt macht.

Literatur

[1] L. Arge and J.S. Vitter. Optimal dynamic interval management in external memory (extended abstract). In IEEE Symposium on Foundations of Coumputer Science, page 560-569, 1996.

[2] Andreas Crauser. LEDA-SM External Memory Algorithms and Data Structures in Theory and Practice. PhD thesis, Universitt des Saarlandes and MPII, 2001.

[3] A. Crauser and P. Ferragina. A Theoretical and Experimental Study on the Construction of Suffix Array in External Memory. ACM Journal of Experimental Algorithmics, 2001. to appear.

[4] Sergey Brin, Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. available from the webpage: http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm.

[5] Ferragina, P. and Manzini, G.: Opportunistic data structures with applications. Danielle C. Young (Ed.): Proceedings of the 41st Annual Symposium on Foundations of Computer Science. Redondo Beach, California (2000). IEEE Computer Society, Los Alamitos, California (2000). 390-398.

[6] Udi Manber and Sun Wu. GLIMPSE: A Tool to Search Through Entire File Systems. Proceedings of the USENIX Winter 1994

Technical Conference. Usenix Association, San Francisco, CA, USA (1994). 23-32.

[7] Stefan Burkhardt, Andreas Crauser, Paolo Ferragina, Hans-Peter Lenhof, Eric Rivals, Martin Vingron. QUASAR - (q-gram Based Database Searching Using a Suffix Array) Max-Planck-Institut für Informatik Saarbrücken : Forschungsbericht 98-1-024, Max-Planck-Institut fr Informatik, 1998.