MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS


Jesper Sindahl Nielsen
Aarhus University

Danmarks Grundforskningsfond
Danish National Research Foundation

Hardness of Several Document Indexing Problems

Background

As large amounts of data become part of our daily lives we need efficient data structures to search the data. One fundamental problem is to index a text such that we can decide fast whether a word or phrase appears in the text. Modifying this problem slightly, we want to index a collection of documents such that we can find all documents that contain a certain word or phrase fast. In search engines we are actually not necessarily interested in *all* the documents that match, only the "best" should be reported. This class of problems is known as top-k document retrieval. Now we have to define what "best" means. Usually a scoring function is designed where several important factors are considered. The measure that Google uses is called PageRank where each document gets a score depending on how many other documents link to it, in this sense the score of a document is somewhat independent of the query (it has to appear in it though!). A measure known as *frequency* is how many times the word or phrase appears in the document. Another one is *term proximity* where a document gets a score based on how close occurrences of the query are. A different approach to filter documents is to give more than one word as a query, for instance if searching through Wikipedia we might be interested in 'Anatomy' and 'Lion', that is we want all documents that contain both words, or alternatively we might interested in 'Elephants' but not 'Africa' to exclude articles about the elephants that live in Africa, while still being interested in the elephants living in Asia.

Formal Definition

Excluded Pattern Problem: Given a set $\mathcal{D} = \{d_1, d_2, \dots, d_D\}$ of documents with total length $n = \sum_{i=0}^{n} d_i$, pre-process $\mathcal{D}$ such that whenever two patterns P^+ and P^- come all documents where P^+occurs and P^- does not occur can be reported. We denote these patterns as the *positive* pattern and the *negative* pattern.

Two (or more) patterns Problem: Given a set $\mathcal{D} = \{d_1, d_2, \dots, d_D\}$ of documents with total length $n = \sum_{i=0}^{n} d_i$, pre-process $\mathcal{D}$ such that whenever two (or more) patterns P_1 and P_2 come as a query, report all documents that contain both P_1 and P_2.

A closely related problem known as the two-dimensional substring indexing problem has also been studied in the literature [2].

Two-dimensional Substring Indexing problem: Given a set of documents $\mathcal{D} = \{(d_{1,1}, d_{1,2}), (d_{2,1}, d_{2,2}), \dots, (d_{D,1}, d_{D,2})\}$ with total length $n = \sum_{i=0}^{n} d_i$, pre-process $\mathcal{D}$ such that whenever a pair (P_1, P_2) comes report all i where P_1 occurs in $d_{i,1}$ and P_2 occurs in $d_{i,2}$.

We show these three problems are all tightly related and we show that the problems are unlikely to have fast algorithms unless there is a fast Matrix Multiplication algorithm.

Matrix Multiplication

The **Matrix Multiplication** problem is defined as: Given two $\sqrt{n} \times \sqrt{n}$ matrices $(a_{i,j})$ and $(b_{i,j})$ compute their product $c_{i,j} = \sum_{k=1}^{\sqrt{n}} a_{i,k} b_{k,j}$.

- The fastest known algorithm for Matrix Multiplication runs in $O(n^{1.18})$ [3].
- That algorithm uses algebraic methods.
- The fastest combinatorial algorithm runs in $O(n^{1.5} / \operatorname{polylog}(n))$ [4].

This means, if we can solve Matrix Multiplication using a solution to one of the document indexing problems, we achieve a *conditional lower bound*. That is, unless a better algorithm for Matrix Multiplication is found, we should not expect to find a better algorithm for the document indexing problems.

Set Intersection
Matrix Multiplication can be used to solve the Empty Set Intersection problem. Let a bit vector of length $\sqrt{n}$ represent a subset of $\{0,1,2, \dots, \sqrt{n} - 1\}$ where index i equals 1 if element i is in the set. In this manner, if we have a set A of $\sqrt{n}$ subsets and a set B of $\sqrt{n}$ subsets we can create a matrix for A and a matrix for B. Computing the product of A and B, tells us that the intersection of the ith set in A and jth set in B is empty if and only if the (i, j) entry in the product is 0.

Looking at the two first problems, one solution is to first find all documents containing the first pattern, then all documents containing the second pattern and compute their intersection (with some appropriate modification). This is an inefficient algorithm but illustrates the underlying problem well.

Solving Matrix Multiplication using Excluded Pattern

For two 0/1 matrices A and B we define the *characteristic set* of each *row* of A as the indices where there is a one. Similarly we define the characteristic set of each *column* of B as the indices where there is a 0. We create documents based on these sets. For the ith cloumn and row one document is made by writing numbers from A's ith characteristic set in binary (each preceded by '0' and all separated by '#'), followed by all numbers in B's ith characteristic set (each preceded by '0' and all separated by '#'). Now build the data structure on these documents. It is not too hard to see that the query $(0i, 1j)$ gives the value of the (i, j) entry in the product of A and B.

Since we are doing n queries to multiply two matrices we get a conditional lower bound of $\frac{n^{1.18}}{n} = n^{0.18}$ on average for a query.

Example of Reduction

Suppose we are given A and B as

$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ (the product $C = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ not given)

Then we define

$$A_1 = \{1,2\}, A_2 = \{0\}, A_3 = \{0,2\}, B_1 = \{0,1\}, B_2 = \{0,2\} \text{ and } B_3 = \{1,2\}.$$

We write the first document from A_i and B_i by using 2 bits per value in the sets and one bit in the beginning indicating if the number is from A or B and all numbers separated by a '#':

$$d_1 = 001\#010\#100\#101\#$$
$$d_2 = 000\#100\#110\#$$
$$d_3 = 000\#010\#101\#110\#$$

Looking at d_1 the first two numbers are from A and the last two from B as indicated by the first bit in the numbers (0 for A, 1 for B). Now we build the data structure on the three documents and execute the queries $(0i, 1j)$ for all values of i and j.

Query(000,100) := is there a document with 000 but not 100? Yes d_2
Query(000,101) := Yes, d_2. Query(000,110) := No. Query(001,100) := No.
Query(001,101) := No. Query(001,110) := Yes, d_1. Query(010,100) := Yes.
Query(010,101) := No. Query(010,110) := Yes.

Writing these answers in matrix form with '1' for yes and '0' for no we get:

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

This is exactly the matrix product of A and B.

References

[1] K. G. Larsen, J. I. Munro, J. S. Nielsen, S. V. Thankachan. *On Hardness of Several String Indexing Problems*. Combinatorial Pattern Matching, 2014.

[2] P. Ferragina, N. Koudas, S. Muthukrishnan, and D. Srivastava. *Two-dimensional substring indexing*. J. Comput. Syst. Sci., 2003.

[3] F. L. Gall. *Powers of tensors and fast matrix multiplication*. CoRR, abs/1401.7714, 2014.

[4] N. Bansal and R. Williams. *Regularity lemmas and combinatorial algorithms*. Theory of Computing, 2012.