All-pairs suffix/prefix in optimal time using Aho-Corasick space

Grigorios Loukides[a] and Solon P. Pissis[b, c, *]

[a]Department of Informatics, King's College London, London, UK
[b]CWI, Amsterdam, the Netherlands
[c]Vrije Universiteit, Amsterdam, the Netherlands

Abstract

The all-pairs suffix/prefix (APSP) problem is a classic problem in computer science with many applications in bioinformatics. Given a set $\{S_1, \ldots, S_k\}$ of k strings of total length n, we are asked to find, for each string $S_i, i \in [1, k]$, its longest suffix that is a prefix of string S_j, for all $j \neq i, j \in [1, k]$.

Keywords

- Algorithms
- Data Structures
- String algorithms
- Aho-Corasick Machine

1 Introduction

Given a set $R = \{S1, ..., Sk\}$ of k strings of total length n, the APSP problem asks us to find, for each string $S_i, i \in [1, k]$, its longest suffix that is a prefix of string S_j, for all $j \neq i, j \in [1, k]$.

Other related work. In addition to l-APSP that is formulated in this paper, there are two other versions of APSP that have been studied in the literature.

Paper organization. Section 2 presents some preliminaries and our main result. Section 3 presents the algorithm we develop. Section 4 concludes the paper.

2 Preliminaries and main result

An *alphabet* Σ is a finite nonempty set whose elements are called *letters*. A string $S = S[1..m]$ is a sequence of *length* $|S| = m$ over Σ. The *empty string* ϵ is the string of length 0. The of two strings S and T is the string composed of the letters of S followed by the letters of T ; it is denoted by $S\dot{T}$ or simply by ST. For $1 \leq i \leq j \leq m$, $S[i]$ denotes the ith letter of S, and the fragment $S[i..j]$ denotes an occurrence of the underlying substring $P = S[i]S[j]$. We say that P *occurs* at (starting) *position* i in S. A fragment $S[i..j]$ is a *suffix* of S if $j = m$, and it is a prefix of S if $i = 1$. A substring of S is called proper if it is not equal to S. Given two strings S and T , a suffix/prefix overlap of S and T is a suffix U of S that is a prefix of T ; when U is the longest such suffix, then U is called the maximal suffix/prefix overlap of S and T .

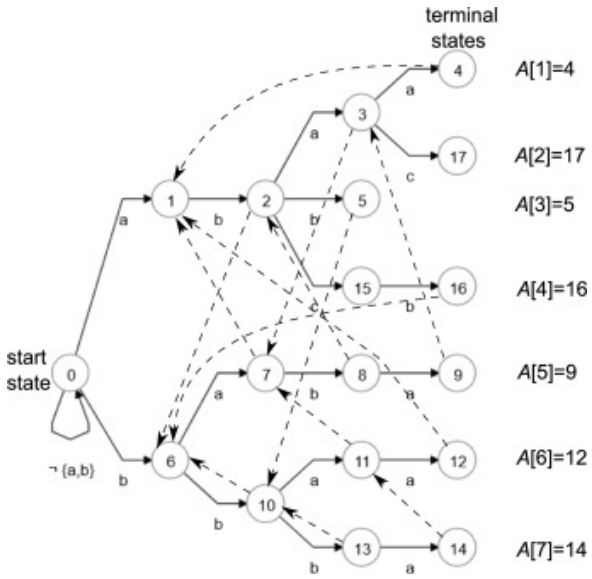

Figure 1: The AC machine over $R = \{abaa, abac, abb, abcb, baba, bbaa, bbba\}$. Solid arrows correspond to goto transitions and dashed arrows to failure transitions. The lexicographic ranks of the strings in R are the indices of array A

Lemma 2.1 *(Aho-Corasick lemma [1]). Let state s correspond to string U and state t correspond to string V in the AC machine of a set R of strings. Then, we have that f (s) = t if and only if V is the longest proper suffix of U that is also a prefix of some string in R*

3 The Algorithm

3.1 Reducing the l-APSP problem to the ULIT problem

Given a collection $I = \{[i1, j1]d1, ..., [ir, jr]dr\}$ of r labeled intervals, we define the union of I as the set

$U(I) = \{e_d : e \in [i, j]_d \in I \textbf{ and } e \in [i', j']_{d'} \in I : d' > d\}$

Definition 1 (Failure transition tree (FTtree)). Given a set R of strings, the failure transition tree

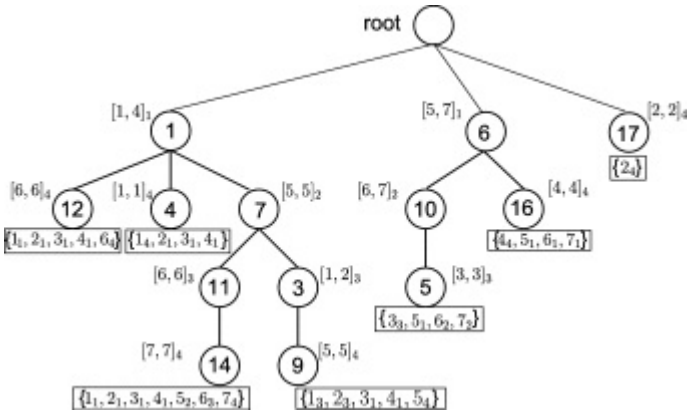

Figure 2: FTtree T_1 for $l = 1$. The output set per leaf node is in a squared box

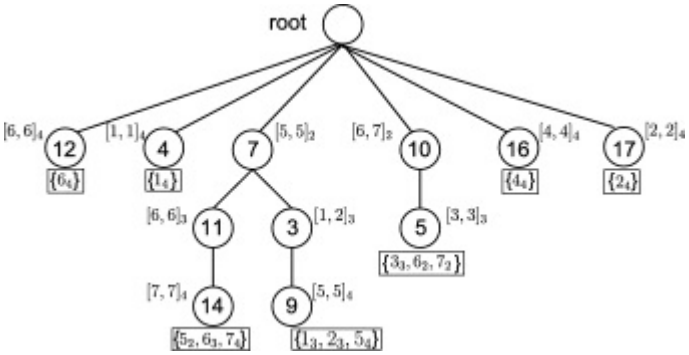

Figure 3: FTtree T_2 for $l = 2$. The output set per leaf node is in a squared box

(FTtree, for short) of R is the rooted tree induced by the set of (reversed) failure transitions of the AC machine of R.

We prove the following lemma.

Lemma 3.1 ***Lemma 2.*** *Any instance of the -APSP problem can be reduced to some instance of the ULIT problem in O(n) time.*

3.2 Solving the ULIT Problem

A branching node of T is a non-root node with at least two children. A branchless subpath is an up-

ward maximal path of nodes starting from a branching node or a leaf node and ending at the node right before a branching node

Branchless Subpaths	Sorted Tuples	Compact Representation of the Union
1	(1,1,4,1)	$\{[1,4]_1\}$
4	(4,1,1,4)	$\{[1,1]_4\}$
5→10	(5,3,3,3) (5,6,7,2)	$\{[3,3]_3, [6,7]_2\}$
6	(6,5,7,1)	$\{[5,7]_1\}$
7	(7,5,5,2)	$\{[5,5]_2\}$
9→3	(9,1,2,3) (9,5,5,4)	{[1,2]_3,[5,5]_4}
12	(12,6,6,4)	$\{[6,6]_4\}$
14→11	(14,6,6,3) (14,7,7,4)	{[6,6]_3,[7,7]_4}
16	(16,4,4,4)	$\{\{4,4]_4\}$
17	(17,2,2,4)	$\{[2,2]_4\}$

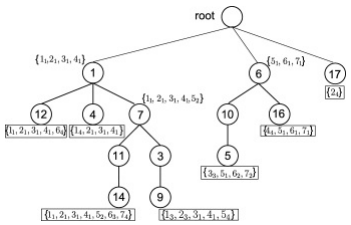

Figure 4: Step 3 using the FTtree in Fig. 2. At each branching node, we store the resulting union. At each leaf node, we store the output in a squared box.

(or the root node if no branching node exists). For exam- ple, the branchless subpaths in Fig. 2 are shown in Table 1. The algorithm for solving the ULIT problem works as fol- lows:

1. Decompose T_l into a set of branchless subpaths. Using a DFS traversal on T_l, for each node decorated with [i, j]d of every branchless subpath starting with node u, construct a tuple (u, i, j, d).
2. Sort all tuples constructed in Step 1 together. For each branchless subpath, take the compact representation of the union of its set of labeled intervals, which are now sorted. Inspect Table 1.
3. Visit the branching and leaf nodes using a BFS traver- sal on T_l. From node u to node v, such that v is the child of u, take the union of the two sets of labeled intervals (one from u and one from v), and update the union associated with v. At the leaf nodes we have the output sets. Inspect Fig. 4.

The following lemma together with Lemma 2 implies Theorem 1.

Lemma 3.2 ***Lemma 3.*** *The ULIT problem can be solved in $\mathcal{O}(\mathcal{N}) + |OUTPUTK|)$ time using $O(N)$ space.*

4 Final Remarks

All existing optimal algorithms for APSP rely on sorting the suffixes of all strings in R.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank Anne Luesink (Vrije Universiteit) for implementing the algorithm underlying Theorem 1. The source code is freely available at wat

References