Data structure for representing maximal repeats in strings

Master thesis

Michal Linhard

Thesis advisor: Mgr. Tibor Hegedüs

Comenius University
Faculty of Mathematics, Physics and Informatics
Department of Computer Science

Bratislava 2007

Abstract

In this diploma thesis we present data structure for representation of maximal repeats in strings – *R3 tree,* based on well known data structure - *suffix tree*. It requires $O(n)$ space and it can be constructed in $O(|\Sigma|n)$ time and $O(n)$ space for string of length n over alphabet Σ. We formalize repeat in string S as triple (p_1, p_2, l), where p_1, p_2 are two distinct positions in S and l is the length of the repeat. We formulate query for maximal repeats in S in the form of the function *findPairs*(p_1, k, S) that returns all pairs (p_2, l) such that (p_1, p_2, l) is maximal repeat with $l \geq k$. R3 tree allows computation of *findPairs* queries in optimal time $O(z)$, where z is the number of found pairs. We also describe design and functionality of *R3lib* – library written in C, for finding maximal repeats in arbitrary binary data, that works with proposed structure.

Table of Contents

1 Introduction

The most known motivation for maximal repeat algorithms comes from bioinformatics. The computation of maximal repeats in strings plays an important role in the analysis of genomic sequences. In general this area stimulates majority of research in area of string algorithms today. There are several other motivations for finding duplication in any data. Repeat discovery may help avoiding redundancy and can be useful in text analysis. For example, it is a good practice to avoid duplication in program source code because of the danger of bug fixes being applied to one copy but not all the others.

There are algorithms and software tools for finding all maximal repeats in a string. Optimal algorithm for finding all maximal repeats was first described in Baker93. This algorithm is based on suffix trees and finds all maximal repeats in $O(n.log|\Sigma|+z)$, where n is length of the string, $|\Sigma|$ is size of the alphabet and z is number of maximal repeats (output size). A space efficient version of this algorithm using suffix arrays is described in AbuOhl04. There are tools that can efficiently find maximal repeats in genome sequences, for example Vmatch[1] (new version of REPuter) and also recent version of MUMmer[2].

Maximal number of all maximal repeats in a string S of size n is $O(n^2)$. Some applications may occur, where we don't want to see all maximal repeats at once, but interactively analyze data or text and see only maximal repeats starting at position in currently viewed segment. Our approach is to build a data structure representing all maximal repeats in the data, that could answer such queries quickly. It turns out, that such structure requires only linear space and also can be constructed in linear time and space.

1 http://www.vmatch.de/
2 http://mummer.sourceforge.net/

2 Definitions and notation

basic notions

definition of tree

definition of suffix tree

definition of suffix array

definition of lcp-table

definition of lcp-interval tree

isomorphism of suffix tree and lcp-interval tree

3 R3 tree

definition of lc-buckets

definition of lc-bucket tree

findPairsNonOptimal query on lc-bucket tree

proof of correctness of findPairsNonOptimal

problems of findPairsNonOptimal

definition of union tree

linearity of union tree

definition of R3 tree

linearity of bpsize pointers

optimal findPairs query on R3 tree

proof of correctness of findPairs

proof of optimality of findPairs

4 R3Lib implementation

implementation of union trees

memory structures of R3 tree

space requirements of R3 tree

algorithm for construction of R3 tree

up table implementation

optimal findPairs query implementation