Technische Universität
Kaiserslautern

Department of Computer Science
Algorithms & Complexity Group

Issue Date: 06.05.2014
Version: 2014-05-19 10:24

Exercise Sheet 2 for Computational Biology (Part 2), SS 14

Hand In: *Until Tuesday, 20.05.2014, 10:00 am, email to* `wild@cs...` *or in lecture.*

Exercise 4 – again

4 points

We consider the data structure from the lecture for efficiently solving the *lce*-problem. Recall: It is based on a compact suffix tree and uses binary numbers in additional node labels.

Find necessary and sufficient conditions for a node u being a predecessor of node v. The condition may only involve the binary numbers i and j that u respectively v are labelled with.

Hint: The function h may be useful for that, where $h(k)$ is the position (counted from the right end) of the least significant 1 in the binary representation of k.
For example $h(8) = h(1000_2) = 4$ and $h(5) = h(101_2) = 1$.

Problem 5

2 + 3 + 5 points

In this exercise, we consider algorithms for *fuzzy* string matching, where we would like to find all occurrences of a pattern $P \in \Sigma^m$ in a text $T \in \Sigma^n$ $(n > m)$, but we do not require to have an exact match. There are two variants of the problem.

The k-***Mismatch*** *Inexact String Matching Problem* consists in finding all occurrences of P in T with up to k mismatches, i. e., formally to find all positions i in the text with

$$\Big|\{j \in [1..m] : P_j \neq T_{i+j-1}\}\Big| \quad \leq \quad k \; .$$

A generalization is the so-called k-***Difference*** *Inexact String Matching Problem*: There, a subword $T_{i,j}$ of T is considered an occurrence of search string P iff $T_{i,j}$ and P have *edit distance*[1] $\leq k$.

To solve these problems, an algorithm is expected to return the set of *all* indices i, such that there is a j with $T_{i,j} \approx P$ (with the appropriate meaning of approximate matches).

[1] Find a definition of edit distance on page 66 of the lecture notes (last paragraph above "Globale Alignments").

a) Design a data structure based on compact suffix trees with which we can compute the *longest common extensions* of two positions in *two* words in constant time (as done for two positions in the *same* word in the lecture).

   Formally, we define for two words $u \in \Sigma^n$ and $v \in \Sigma^m$:

$$lce(i,j) := u_{i,i+\ell_{\max}} \quad \text{where} \quad \ell_{\max} := \max\{\ell \geq 0 : u_{i,i+\ell} = v_{j,j+\ell}\}$$

   Hint: Read/Review the section on the subword problem for a set of texts, page 59f in the lecture notes.

b) Give an efficient algorithm for solving the k-mismatch inexact string matching problem and analyze its running time.

   The algorithm only needs to be efficient for $k \ll m$.

   Hint: Use *lce*-queries (and the datastructure form a) to efficiently answer them).

c) Design an efficient algorithm for the k-difference inexact string matching problem and determine its running time.

   Hint: Use dynamic programming.

Problem 6

3 + 2 points

In the lecture, we considered an algorithm to compute all tandom repeats.

Formally, "all tandom repeats of T" means the following set

$$R \quad = \quad \{(i,\ell) : T_{i,i+\ell-1} = T_{i+\ell,i+2\ell-1}\}.$$

a) Describe a method based on this algorithm to compute all *triple repeats*, i. e. all subwords of shape xxx in a text T.

b) Generalize your method to *higher order repeats*, i. e. subwords of the form x^k for arbitrary $k \geq 2$.