Saarland University
Jun.-Prof. Dr. Tobias Marschall
Dr. Marcel Schulz

UNIVERSITÄT DES SAARLANDES

2.Assignment - Algorithms for Sequence Analysis, SS 2018

Exercise 1: Suffixes in different ways (1+1=2 Theory)
For this task, we define the lexicographical order $\$ < A < B < C < ... < Z$.

(a) Construct the suffix tree for the string $s = ALILAILIL\$$. Order the children of each suffix tree node lexicographically. Also specify the corresponding suffix array *pos*.

(b) In the string s given in task $1(a)$, search for the pattern $P = LIL$ using binary search. Use the suffix array constructed in task $1(a)$. Report the suffix array interval, where the pattern matches and also show the intermediate steps used for the search.

Exercise 2: Searching for palindromes in a DNA sequence (4 Theory)
A DNA sequence contains only the letters A, C, G and T. (Each letter represents a small molecule, and a DNA sequence is a "macromolecular" chain of them.) The reverse complement of a DNA sequence is formed by reversing the letters, interchanging A and T and interchanging C and G. Thus the reverse complement of ACCTGAG is CTCAGGT. We denote the reverse complement of string s by $rc(s)$. Let s be a DNA sequence of length n. An even length substring s' of s is called a *maximal palindrome*, if there are integers i and $k > 0$ such that $s' = s[i-k \dots i+k-1]$ and $s[i-k \dots i-1] = rc(s[i \dots i+k-1])$ and $s'[i-k-1] \neq rc(s'[i+k])$.
(**Notation**: For a string s and two indexes i and j, $s[i]$ denotes the letter at the ith position of s, and $s[i \dots j]$ denotes a substring of s that starts at position i and ends at position j.)

Example: The string $s = CACGTTAACGTCA$ contains the maximal palindromes $ACGT$ and $ACGTTAACGT$, but neither of the occurrences of CG are *maximal palindromes*.

Devise an algorithm that finds all maximal palindromes in a DNA sequence in optimal linear time. Explain why your solution runs in linear time. You can assume that there is an algorithm that for a string s and for every pair of indexes i and j, finds the length of the longest common prefix of the suffixes of s that start from the ith and the jth positions in constant time.

Exercise 3: Suffix arrays (2+4=6 Programming)
Implement a program that builds and searches in suffix arrays. The program should take a pattern P and a file with a text T as input as shown below. Implement suffix arrays with the following features (points will be given for the features correctly implemented):

(a) Implement the suffix array with naive sorting as shown in the lecture and post the sorted suffix array (labeled `pos` in the example) as a comma separated list of the lexicographically sorted positions.

(b) As the binary search proceeds, let L and R denote the left and right boundaries of the current search interval. At the start, L equals 1 and R equals n. Then in each iteration of the binary search a query is made at location $M = \lceil (R + L)/2 \rceil$ of Pos. We keep track of the longest prefixes of $Pos(L)$ and $Pos(R)$ that match a prefix of P. Let l and r denote the prefix lengths respectively and let $mlr = \min(l, r)$. Then we can use the value mlr to accelerate the lexicographical comparison of P and the suffix $Pos[M]$. Since Pos gives the lexical ordering of the suffixes of the text, it is clear that all suffixes between L and R share the same prefix with P in the first mlr positions. Hence we can start the first comparison of P and the suffix $Pos(M)$ at position $mlr + 1$, rather than from the first position.

Implement the above mentioned mlr binary search algorithm and report the suffix array interval where the pattern matches, or report NotFound if it doesn't exist in the text. The interval should be given on one line as in the example below (`interval`).

Note that your program should show these features explained above exactly in the way posted below. Also note that the string positions start with 0 for the output. You may assume that the alphabet consists only of the letters from the DNA alphabet, *i.e.* $\Sigma = \{A, C, G, T, \$\}$. The program gets as the first parameter the pattern $P = AAAA$ to search in the sequence TATAAAAAAT$ in the file input.txt.

```
1  $ ./program AAAA input.txt
2  pos=[10,3,4,5,6,7,8,1,9,2,0]
3  text=TATAAAAAAT$
4  pattern=AAAA
5  intervall=[1,3]
```

Remarks:

- There are 12 points to be earned on this assignment sheet.
- 50% of programing points (18/36) and 50% of theoretical exercises (18/36) are necessary to take the exam.
- You are allowed to work in groups of two.
- Hand in your solutions on paper (except for source code) by putting it in the letter box of Tobias Marschall in E2.1 (ground floor).
- Programming code is to be send as a *tar.gz* package by mail to the email address:
  - aryan.3264@gmail.com
- Source code is only considered if
  - it is in one of the languages Python, Perl, C, C++, or Java,
  - it is reasonably documented, commented, and readable,
  - command-lines for compiling and calling are provided,
  - compilation does not fail, and
  - executables are named according to:

    `lastname1_lastname2_assignment2_exercise3`
- For the programing task, there will be more than one input sequences. One is for you for testing and the others are for us for grading. The latter two are kept secret and points are awarded based on whether your program computes the right answer for these two input files.
- Do not forget to mention your names and matriculation numbers on your solutions!
- Copying between groups will result in zero points for all involved groups!

Hand in date: Tuesday, 22.05.2017, before 16:00.