Assignment 2. Pattern Matching

Algorithms

In this assignment we explore some algorithmic ideas for the same **Pattern Search Problem**: given a string *P* of length *M* and a larger string *T* of length *N*, find all occurrences of *P* in *T*.

Create a copy of this document, and type results directly in your document. You can either provide me with the link to the google doc (don't forget to share the document with me), or you can download the file as pdf, and upload it to the blackboard.

Results

Question	Points	Out of
1.1		10
1.2		10
1.3		15
2		30
Total:		65

Part 1. More shifting heuristics

In the naive algorithm, we check the occurrence of pattern *P* in text *T*, each time aligning the start of *P* with the next character in *T*. In the KMP algorithm we speed up the search by using some information about the pattern P and the characters of T that were already aligned with P. This allows us to shift the start of P by more than one position in T.

Let's explore some alternative shifting heuristics.

As before, we first align the position 0 of the pattern with position 0 of the text. However, we compare characters of *P* **starting from the last position of *P***, position *M* - 1 (where *M* is the length of *P*).

So we are moving the start position *i* in *T* from left to right, but we are comparing the characters of P with characters of T **from right to left**.

2.1. [10 points] Bad character

Consider the following snapshot of the first alignment of *P* with *T*:

T	a	b	a	c	d	a	b	a	a	a	b	a	b
i	0	1	2	3	4	5	6	7	8	9	10	11	12

P	a	b	a	b									
j	0	1	2	3									

We compare *P*[3] and *T*[3], and we discover that they do not match.
In addition we notice that character 'c' at position 3 of *T* does not appear anywhere in *P*.

Where should we position *P* for the next alignment? (draw)

T	a	b	a	c	d	a	b	a	a	a	b	a	b
i	0	1	2	3	4	5	6	7	8	9	10	11	12

P													
j													

I will align the start position of *P* with text position *i* = _____
After this, my first comparison will be between *T*[__] and *P*[__]

How many characters of *T* were completely skipped and not compared at all? _____

1.2. [10 points] How far to shift?

Consider the following snapshot of the algorithm:

T	a	b	a	c	d	a	b	a	a	a	b	a	b
i	0	1	2	3	4	5	6	7	8	9	10	11	11

P						a	b	a	b				
j						0	1	2	3				

Again, we start character comparison from the end. We compare *P*[3] and *T*[8] and they do not match. We do not need to continue comparing *P*[0:2] with *T*[5:7].

Note, however, that unlike in the previous scenario, the current character 'a' of *T* **does** occur inside *P*: in fact, it occurs at two positions -- 0 and 2.

Let's assume that string *T* is generated from a finite alphabet σ and we know the alphabet in advance. Let the alphabet be: σ = {'a', 'b', 'c', 'd'}. Before performing the search, we precompute positions in *P* for each letter of the alphabet. If *P* does not contain some letters, their positions are marked as -1.

Table A: for each character in σ, its positions inside *P*

char	Pos in P
a	0, 2
b	1, 3
c	-1
d	-1

Given this table and the knowledge that the current letter *T*[i] is 'a', how many positions forward should we shift P to start the next comparison?

Choose the correct option from the three options below:
Option 1. We will align P to start past T[8] as we did in 2.1

T	a	b	a	c	d	a	b	a	a	a	b	a	b
i	0	1	2	3	4	5	6	7	8	9	10	11	11

P										a	b	a	b
j										0	1	2	3

Option 2. We will align current character T[8] =’a’ with the **first** occurrence of ‘a’ in P, yielding the following alignment for the next step:

T	a	b	a	c	d	a	b	a	a	a	b	a	b
i	0	1	2	3	4	5	6	7	8	9	10	11	11

P									a	b	a	b	
j									0	1	2	3	

Option 3. We will align current character T[8] =’a’ with the **last** occurrence of ‘a’ in P, yielding the following alignment for the next step:

T	a	b	a	c	d	a	b	a	a	a	b	a	b
i	0	1	2	3	4	5	6	7	8	9	10	11	11

P							a	b	a	b			
j							0	1	2	3			

1.3. [15 points]. General shift

Because only one of the three options in the previous questions is correct, we now can reduce the number of entries in Table A. Draw the new Table A:

Based on your answer, and assuming that table A is precomputed before search, can you come up with the general formula of how much you should shift the start of the pattern in each iteration of this algorithm?

Next i = current i + _____

This shifting heuristic is a part of the ***Boyer-Moore pattern matching algorithm***. You can read more about it in your textbook Chapter 2.2.

Part II. [20 points] Circular rotation

Use the existence of a linear-time exact matching algorithm to solve the following problem in linear time:

Problem: is string *A* a circular rotation of string *B*?
Given 2 strings, *A* and *B*, determine if *A* is a cyclic rotation of *B*.

Definition: Any string *X* is a ***circular rotation*** of another string *Y*, if both have the same length and *Y* can be obtained from X by concatenating some suffix X[i:N] with the corresponding prefix X[1:i-1].

For example:
String ***def****abc* is a cyclic rotation of *abc****def***.
String *ab****babab*** is a cyclic rotation of ***babab****ab.*
String *abccba* is NOT a cyclic rotation of *ccabba*.

Explain your algorithm in English and give a high-level pseudocode.
Argue that the algorithm runs in linear time.