Problem Set 3, Problem 3 **Jihong Tang**
COMPSCI 260 NetID: jt290
Work Date: 10/06/2018 Due Date: Fri 10/12/2018 5pm
Extension Due Date: Sun 10/14/2018 5pm

Introduction to the problem

Problem 3 from problem set 3 is based on the BWT method and the exact read-matching algorithm, covering different ways to implement them including handwriting and python coding.

For question 3a, we are asked to perform the BWT on the given sequence $CGGACTAA\$$ by hand.

For question 3b, we are asked to perform the reverse BWT on the given sequence $TTAGAC\$AG$ by hand.

For question 3c, we are asked to implement the BWT and reverse BWT function in python.

For question 3d, we are asked to write a program based on the exact read-matching algorithm, as known as FM indexing algorithm.

Problem 3a

After assuming the \$ character precedes every character alphabetically, the BWT on the original sequence $CGGACTAA\$$ can be performed as following steps:

- The first step is to give different indexes to different suffixes in the original sequence . The result can be found in Table1.

Table 1: BWT step one – Suffixes

0	C	G	G	A	C	T	A	A	$
1	G	G	A	C	T	A	A	$	
2	G	A	C	T	A	A	$		
3	A	C	T	A	A	$			
4	C	T	A	A	$				
5	T	A	A	$					
6	A	A	$						
7	A	$							
8	$								

The first column in the table is the 0-indexed position index of the first character behind the index in the original sequence, and the \$ is the end-of-file character.

- The second step is to sort the table alphabetically in the fist step to get the sorted suffixes table as shown in Table2.

Table 2: BWT step two – Sorted suffixes

8	\$								
7	A	\$							
6	A	A	\$						
3	A	C	T	A	A	\$			
0	C	G	G	A	C	T	A	A	\$
4	C	T	A	A	\$				
2	G	A	C	T	A	A	\$		
1	G	G	A	C	T	A	A	\$	
5	T	A	A	\$					

As I am sorting the table, I make the assumption that \$ precedes every character alphabetically. In addition, we could read out the suffix array from Table2 and it is shown as following:

$$[8, 7, 6, 3, 0, 4, 2, 1, 5]$$

.

- The third and the last step is to make cyclic permutations based on Table2 to get the cyclic permutation table in which we could find the final permuted sequence. The cyclic permutations table is shown as Table3.

Table 3: BWT step three – Cyclic permutations

8	\$	**C**	**G**	**G**	**A**	**C**	**T**	**A**	**A**
7	A	\$	**C**	**G**	**G**	**A**	**C**	**T**	**A**
6	A	A	\$	**C**	**G**	**G**	**A**	**C**	**T**
3	A	C	T	A	A	\$	**C**	**G**	**G**
0	C	G	G	A	C	T	A	A	\$
4	C	T	A	A	\$	**C**	**G**	**G**	**A**
2	G	A	C	T	A	A	\$	**C**	**G**
1	G	G	A	C	T	A	A	\$	**C**
5	T	A	A	\$	**C**	**G**	**G**	**A**	**C**

The final permuted sequence can be read out from the last column in Table3 to be

$$AATG\$AGCC$$

Problem 3b

After assuming the \$ character precedes every character alphabetically, the reverse BWT on the given sequence $TTAGAC\$AG$ can be performed as following steps:

1. The first step is to store the given sequence in one column as in Table4. I will call the given sequence and the column as BWT sequemce and BWT column respectively in following steps.

2. The second step is to sort the BWT column in Table4 alphabetically and append the sorted sequence to the BWT column to get Table5.

3. The third to ninth step is to do the similar procedure as following: Firstly, sort the result table from the last step alphabetically. Secondly, append the sorted table to the BWT column to get the result table of this step.The seven result table in this step can be shown as Table6 to Table12

Table 4: reverse BWT step one

T
T
A
G
A
C
$
A
G

Table 5: reverse BWT step two

T	$
T	A
A	A
G	A
A	C
C	G
$	G
A	T
G	T

Table 6: reverse BWT step three

T	$	G
T	A	A
A	A	C
G	A	T
A	C	G
C	G	A
$	G	T
A	T	$
G	T	A

Table 7: reverse BWT step four

T	$	G	T
T	A	A	C
A	A	C	G
G	A	T	$
A	C	G	A
C	G	A	T
$	G	T	A
A	T	$	G
G	T	A	A

Table 8: reverse BWT step five

T	$	G	T	A
T	A	A	C	G
A	A	C	G	A
G	A	T	$	G
A	C	G	A	T
C	G	A	T	$
$	G	T	A	A
A	T	$	G	T
G	T	A	A	C

Table 9: reverse BWT step six

T	$	G	T	A	A
T	A	A	C	G	A
A	A	C	G	A	T
G	A	T	$	G	T
A	C	G	A	T	$
C	G	A	T	$	G
$	G	T	A	A	C
A	T	$	G	T	A
G	T	A	A	C	G

Table 10: reverse BWT step seven

T	$	G	T	A	A	C
T	A	A	C	G	A	T
A	A	C	G	A	T	$
G	A	T	$	G	T	A
A	C	G	A	T	$	G
C	G	A	T	$	G	T
$	G	T	A	A	C	G
A	T	$	G	T	A	A
G	T	A	A	C	G	A

Table 11: reverse BWT step eight

T	$	G	T	A	A	C	G
T	A	A	C	G	A	T	$
A	A	C	G	A	T	$	G
G	A	T	$	G	T	A	A
A	C	G	A	T	$	G	T
C	G	A	T	$	G	T	A
$	G	T	A	A	C	G	A
A	T	$	G	T	A	A	C
G	T	A	A	C	G	A	T

Table 12: reverse BWT step nine

T	$	G	T	A	A	C	G	A
T	A	A	C	G	A	T	$	G
A	A	C	G	A	T	$	G	T
G	A	T	$	G	T	A	A	C
A	C	G	A	T	$	G	T	A
C	G	A	T	$	G	T	A	A
$	G	T	A	A	C	G	A	T
A	T	$	G	T	A	A	C	G
G	T	A	A	C	G	A	T	$

Therefore, we could get the original sequence using reverse BWT as following:

$$GTAACGAT\$$$

Problem 3c

The code solution of the BWT and reverse BWT implement in python can be found in **bwt_structure.py** as shown in function **forward_bwt** and **reverse_bwt** respectively.

I have checked the given example and the sequence in problem 3a and 3b to prove the correctness of my code. The detailed outputs can be found in the code file.

Problem 3d

The code solution of the FM-indexing algrithm to do the exact read-matching work can be found in **read_aligner.py**.

In my code, I implement the algorithm by focusing on the parameters that recording the start and end index of the range of rows in F column. I used parameters **start** and **end** respectively to play the role.

The expression to work out these two parameters can be divided into two parts: initiated step and iterative steps. Let's call the last character in the query sequence is C, then we could get the following expression for the initiated step:

$$start = counts[C] + 1$$

$$end = counts[C] + ranks[C][-1]$$

The iterative steps in my code are implemented under the following expressions:

$$start = counts[re\_query[j]] + ranks[re\_query[j]][s-1] + 1$$

$$end = counts[re\_query[j]] + ranks[re\_query[j]][end]$$

The re_query in the expression represents the reverse query sequence, and the j index represents different position in the reverse query sequence.

The detailed comments and the example outputs can be found in the code file.

Citation

I have offered some hints to my classmates Xi Li about the the implement of the exact read-matching algorithm.