CSCI3220 2018-19 First Term Assignment 2

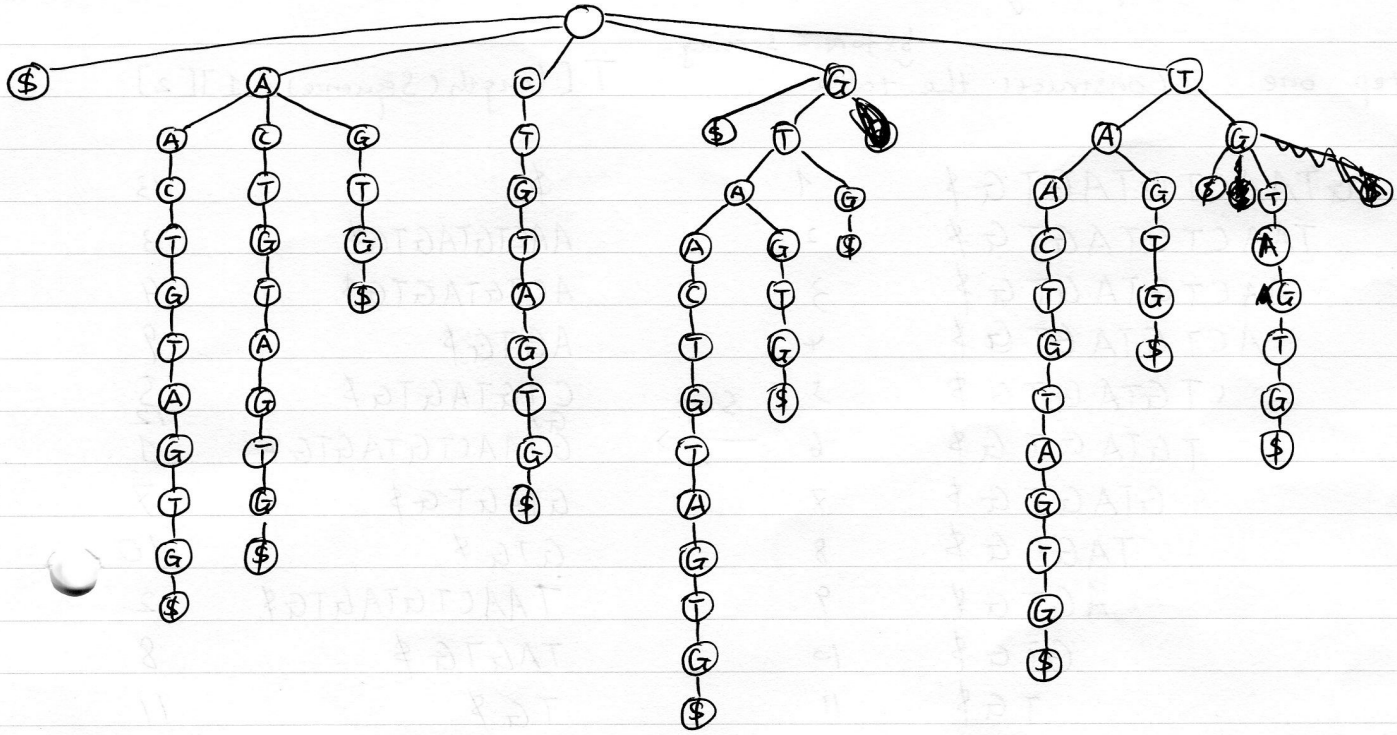
I declare that the assignment here submitted is original except for source material explicitly acknowledged, and that the same or closely related material has not been previously submitted for another course. I also acknowledge that I am aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the following websites.

University Guideline on Academic Honesty:
http://www.cuhk.edu.hk/policy/academichonesty/

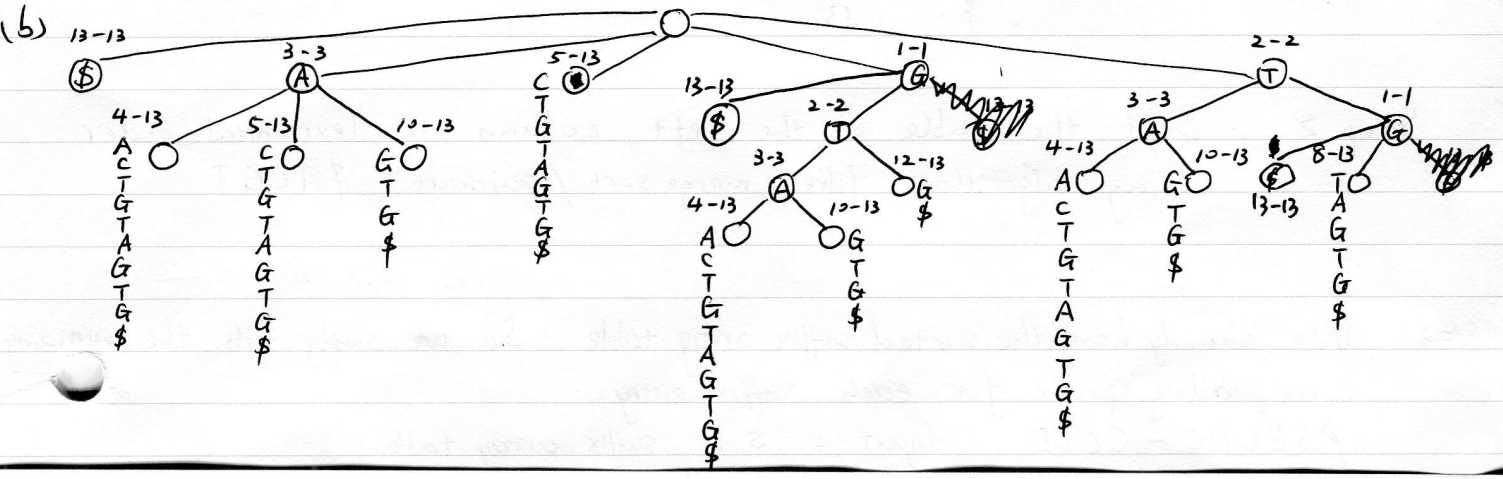
Student Name: Zuowen Wang
Student ID : 1155123906

1. (a)

S = GTAACTGTAGTG$



(b)



(c) We traverse the suffix tree (compressed) with index using DFS. When we encounter a leaf node, we output (its lower index − depth of its parent) to a dynamic array (for convenience). Notice the root has depth 0.

pseudo code: Input: Tree G, Output: Suffix Array SA.

```
Main:   call DFS(G, root$, -1)

DFS:    DFS(G, v, depth) // depth is depth of v's parent
            depth++;
            foreach child u of v in lexicographic order
                if u is a leaf.
                    SA.append(u.index.low - depth)
                else
                    DFS(G, u, depth)
```

Using this algorithm we can get suffix array from suffix tree

SA: 13, 4, 4, 9, 5, 12, 1, 7, 10, 2, 8, 11, 6

(d) Input : s = GTAACTGTAGTG$

Goal : Output : The Suffix Array A

Step 1. Create an array of String. The length of array is s.length. And store all suffixes (from longest to the $) in this array.

Step 2. Sort this array A using lexiconic order $ACGT
Better use in place merge sort, Since suffix array method ~~is~~ was created for memory limitation. or heap sort.

Step 3 Output the corresponding location for each entry of A.
The location can be computed as (s.length − A[i].length +1)

A		A'		
GTAACTGTAGTG$		$		13
TAACTGTAGTG$		AACTGTAGTG$		3
AACTGTAGTG$		ACTGTAGTG$		4
ACTGTAGTG$		AGTG$		9
CTGTAGTG$	Step 1 Sorting →	CTGTAGTG$	Step 3 Compute Location →	5
TGTAGTG$		G$		12
GTAGTG$		GTAACTGTAGTG$		1
TAGTG$		GTAGTG$		7
AGTG$		GTG$		10
GTG$		TAACTGTAGTG$		2
TG$		TAGTG$		8
G$		TG$		11
$		TGTAGTG$		6

= 13 − 1 + 1 = 13

(e) Assume we have suffix array sa. Now we want to get the corresponding BWT b of s.

Step 1. (actually there is only one step ...)
We can construct b directly from sa and s using the following pseudo code: initially b = "" // empty string.

```
for i from 1 to S.length
    b = b + S.CharAt( sa[i] - 1)
```

if this == 0, then return $!

Output b.

Our b is: GTATAT$TAGGGC

The reason why we can do that is because the last character in Sorted rotation should be right before the character at Location in the original sequence. For instance

Sorted Rotation

CTGTAGTG$GTAA ← Original Sequence

↑ GTAACT....

Sorted Suffix & Suffix Array Location

CTGTAGTG$ 5 -1 = 4

~~Namely b[i]=~~

(f) We use rotation matrix and sort the rows in lexicographic order. Then we output the sequence in the last column as b

Step 1. Construct the rotation matrix
Step 2. Sort by row. in lexi. order
Step 3. Output b.

✓ GTAACTGTAGTG$	$GTAACTGTAGTG		b= G
✓ TAACTGTAGTG$G	AACTGTAGTG$GT		T
✓ AACTGTAGTG$GT	ACTGTAGTG$GTA		A
✓ ACTGTAGTG$GTA	AGTG$GTAACTGT		T
✓ CTGTAGTG$GTAA	CTGTAGTG$GTAA		A
Step 1 → ✓ TGTAGTG$GTAAC	TAACTGTAGTG$G	Step 3 →	G
~~TAACT GTAAC~~	Step 2 → TAG ... G		G
	TG$ G		G
✓ GTAGTG$GTAACT	TGT ... C	Sorry.	C
✓ TAGTG$GTAACTG	G$ ... T	wrong	T
✓ AGTG$GTAACTGT	GTAA ... $	order...	$
✓ GTG$GTAACTGTA	GTAG ... T	G<T	T
✓ TG$GTAACTGTAG	GTG$ ... A		A
✓ G$GTAACTGTAGT			
✓ $GTAACTGTAGTG			

~~(g) Last~~

	1	2	3	4	5	6	7	8	9	10	11	12	13
(9) Last Column (b)	G	T	A	T	A	T	$	T	A	G	G	G	C

(Using this we can construct First Column:)

△ First Column $ A A A C G G G G T T T T

First Occurrence (F) in First Column

	A	C	G	T
	2	5	6	10

Occurences within the first i rows in the last column (O)	1	2	3	4	5	6	7	8	9	10	11	12	13
A	0	0	1	1	2	2	2	2	3	3	3	3	3
C	0	0	0	0	0	0	0	0	0	0	0	0	1
G	1	1	1	1	1	1	1	1	1	2	3	4	4
T	0	1	1	2	2	3	3	4	4	4	4	4	4
SA	13	3	4	9	5	12	1	7	10	2	8	11	6

△ Notice that we don't need to store First Column in order to get F since we can compute F simply based on b.

Now using b, F, O, we can construct the original sequence S and the suffix array sa. I only give the first several steps here since its also on the lecture notes and all the steps follow the same pattern.

① we start from the first character G in b. since it must be right before $ in S. We see O[1, G] = 1, the check F[G] + O[1, G] - 1 = 6 then jump to 6th character in b, b[6] = T, and then check O[6, T] = 3, F[T] + O[6, T] - 1 = 12, so we got next character is b[12] = G ...

We can represent these steps as follow:

(*) $ → G_1 → T_3 → G_4 → A_3 → T_2 → G_3 → T_4 → C_1 → A_2 → A_1 → T_1 → G_2

So the original sequence S is GTAACTGTAGTG$

② Now we can construct SA, which is very useful and will be stored on hard disk since it will be accessed not very frequent in ~~text~~ search. (only once)

If we construct it from original S it will cost O(n log n)

But if we hash the characters in string (*) above, using its ~~index~~ position as value, character as key. we can get SA in O(n)

(*)	$	→ G_1	→ T_3	→ G_4	→ A_3	→ T_2	→ G_3	→ T_4	→ C_1	→ A_2	→ A_1	→ T_1	→ G_2
hash into	13	12	11	10	9	8	7	6	5	4	3	2	1

Then we get the hash value of b (with occurences in O)

b:	G	T	A	T	A	T	$	T	A	G	G	G	C
O(b):	G_1	T_1	A_1	T_2	A_2	T_3	$	T_4	A_3	G_2	G_3	G_4	C_1
SA	13	3	4	9	5	12	1	7	10	2	8	11	6

= hash.get(O(b)) + 1

if (14), ~~add~~ change to 1 ~~can be done with mod (%) etc.~~

Now we have constructed all tables we need for text search. Let's start the searching

To search GTA we first search the occurence of A then T then G.

1. O[13, A] = 3 occurrences of A in total
2. In F these A's appear on row F[A] = 2 to row F[A] + 3 - 1 = 4
3. on row 2 - 1 = 1, O[2-1, T] = 0 T's has appeared in last column
4. on row 4, O[4, T] = 2 T's have appeared in last column.
5. Therefore these rows cover the (0+1) = 1st to the 2nd occurrences of T
6. In the first column these T's appear on row F[T] + 1 - 1 = 10 and F[T] + 2 - 1 = 11

Then we check G. (actually GT)

7. on row 10 - 1 = 9, O[9, G] = 1 G's has appeared in last cd.
8. on row 11, O[11, G] = 3 G's has appeared in last column.
9. therefore these rows cover the (1+1) = 2nd to the 3rd occurrences of G.
10. In the first column these G's appear on row F[G] + 2 - 1 = 7 and F[G] + 3 - 1 = 8
11. These GTA's appear at position SA[7] = 1 and SA[8] = 7 of the input string s.

(h) Well apparently not equal in each of the n+1 positions, since in BWT output b, $ can never appear in the first place of b, since $ is smallest in lexicongraphical order, so the first line of the sorted rotation matrix is always $.....