COT 6417 - Algorithms on Strings and Sequences

Fall 2020

Homework 2

Name: Nabila Shahnaz Khan

PID: 5067496

#Question 1:

<u>Solution:</u>

Let, S be a string of length m and let SA denote the suffix array for S. LCP array stores the length of the longest common prefixes between consecutive pair of suffixes in SA.

Algorithm to Construct LCP array from SA:

Suppose, R is an array of size m where R[j] is the index in SA of the suffix S[j, m]. If SA[i] = j, then R[j] = i. R and SA are inverses of each other; that is, R[SA[i]] = i and SA[R[j]] = j. The suffix that appears after suffix S[j, m] in SA is S[SA[R[j]+1], m]. We refer to this suffix as the right neighbor of S[j, m] in SA.

The LCP array can be constructed in m iterations. The steps are given below:

- In the first iteration, LCP[R[1]] is computed by identifying the lcp between S[1,m] and S[SA[R[1]+1], m] (right neighbor of S[1,m]) based on character comparisons.
- Let p = SA[R[k]+1] and let 'len' be the length of the lcp of S[k,m] and S[p,m]. In iteration k+1 (1 <= k < m), we can compute LCP[R[k+1]]. There can be two cases:
  - **Case 1 (len ≥ 1):** As len ≥ 1, S(k)=S(p). Suffix S[p, m] is lexicographically greater than suffix S[k, m] since it appears after S[k, m] in SA. This implies that suffix S[(p+1), m] is lexicographically greater than S[(k+1),m] and that the lcp of these two suffixes has length at least len-1. Thus, it follows that the lcp of S[(k+1),m] and its right neighbor in SA, namely S[SA[R[k+1]+1], m], has length ≥ (len - 1). So, comparison between these two suffixes can start at their len^{th} character to determine the correct value of their lcp length.
  - **Case 2 (len = 0):** In this case, comparison to determine the lcp length of S[(k+1),m] and its right neighbor need to be started at their first character.

Runtime:

Here, the total number of iterations is m. A comparison between characters is called successful if there's a match; otherwise it is known as failed. There is only one failed comparison at each iteration, so total number of failed comparisons is O(m). Moreover, each position in the string is compared only once for a successful comparison, so total number of successful comparisons is O(m).

In total, runtime = number of failed comparisons + no of successful comparisons

= O(m) + O(m)

= O(2m)

≈ O(m)

Pseudocode:

```
LCP[R[1]] = lcp(S[1,m] , S[SA[R[1]+1], m]) // returns the length of the longest common prefixes
                                                     between given suffixes
for(k = 1; k < m; k++) {
        len = LCP[R[k]]
        q = SA[R[k+1]+1]
        if(len ≥ 1){    //start comparison after len-1 match
                LCP[R[k+1]] = len - 1
                i = k + 1 + len – 1
                j= q + len – 1
        }
        else if (len == 0) {   //start comparison at first character
                i = k+1
                j = q
        }
        while (S[i] == S[j] and i ≤ m and j ≤ m){
                LCP[R[k+1]] = LCP[R[k+1]] + 1
                i = i + 1
                j = j + 1
        }
}
```

#Question2:

Solution:

Given, S = aabaaabbaacda$

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S	a	a	b	a	a	a	b	b	a	a	c	d	a	$

Construction of SA_1:

i	$SA_1[i]$ (= j)	1-length Prefix	Bucket Number
1	14	$	1
2	1	a	2
3	2	a	2
4	4	a	2
5	5	a	2
6	6	a	2
7	9	a	2
8	10	a	2
9	13	a	2
10	3	b	3
11	7	b	3
12	8	b	3
13	11	c	4
14	12	d	5

Construction of SA_2:

i	$SA_2[i]$ (= j)	2-length Prefix	Bucket Number of suffix A_{j+1} in SA_1	Bucket Number
1	14	$	-	1
2	13	a$	1	2
3	1	aa	2	3
4	4	aa	2	3
5	5	aa	2	3
6	9	aa	2	3
7	2	ab	3	4
8	6	ab	3	4
9	10	ac	4	5
10	3	ba	2	6
11	8	ba	2	6
12	7	bb	3	7
13	11	cd	-	8
14	12	da	-	9

Construction of SA_4:

i	$SA_4[i]$ (= j)	4-length Prefix	Bucket Number of suffix A_{j+2} in SA_2	Bucket Number
1	14	$	-	1
2	13	a$	-	2
3	4	aaab	4	3
4	1	aaba	6	4
5	5	aabb	7	5
6	9	aacd	8	6
7	2	abaa	3	7
8	6	abba	6	8
9	10	acda	-	9
10	3	baaa	3	10
11	8	baac	5	11
12	7	bbaa	-	12
13	11	cda$	-	13
14	12	da$	-	14

After construction of SA_4, we already have m (14) number of buckets. So, our final suffix array is SA_4.

#Question 3:

Solution:

Given, S = aabaaabbaacda$

The successor function Ψ values for S string are given in the table below:

i	SA[i]	S[SA[i],m]	Ψ(i)
1	14	$	4
2	13	a$	1
3	4	aaabbaacda$	5
4	1	aabaaabbaacda$	7
5	5	aabbaacda$	8
6	9	aacda$	9
7	2	abaaabbaacda$	10
8	6	abbaacda$	12
9	10	acda$	13
10	3	baaabbaacda$	3
11	8	baacda$	6
12	7	bbaacda$	11
13	11	cda$	14
14	12	da$	2

#Question 4:

<u>Solution:</u>

Given, S = aabaaabbaacda$

Cyclic shifts of string S:

i														
1	a	a	b	a	a	a	b	b	a	a	c	d	a	$
2	a	b	a	a	a	b	b	a	a	c	d	a	$	a
3	b	a	a	a	b	b	a	a	c	d	a	$	a	a
4	a	a	a	b	b	a	a	c	d	a	$	a	a	b
5	a	a	b	b	a	a	c	d	a	$	a	a	b	a
6	a	b	b	a	a	c	d	a	$	a	a	b	a	a
7	b	b	a	a	c	d	a	$	a	a	b	a	a	a
8	b	a	a	c	d	a	$	a	a	b	a	a	a	b
9	a	a	c	d	a	$	a	a	b	a	a	a	b	b
10	a	c	d	a	$	a	a	b	a	a	a	b	b	a
11	c	d	a	$	a	a	b	a	a	a	b	b	a	a
12	d	a	$	a	a	b	a	a	a	b	b	a	a	c
13	a	$	a	a	b	a	a	a	b	b	a	a	c	d
14	$	a	a	b	a	a	a	b	b	a	a	c	d	a

i	SA[i] = j	F													L
1	14	$	a	a	b	a	a	a	b	b	a	a	c	d	a
2	13	a	$	a	a	b	a	a	a	b	b	a	a	c	d
3	4	a	a	a	b	b	a	a	c	d	a	$	a	a	b
4	1	a	a	b	a	a	a	b	b	a	a	c	d	a	$
5	5	a	a	b	b	a	a	c	d	a	$	a	a	b	a
6	9	a	a	c	d	a	$	a	a	b	a	a	a	b	b
7	2	a	b	a	a	a	b	b	a	a	c	d	a	$	a
8	6	a	b	b	a	a	c	d	a	$	a	a	b	a	a
9	10	a	c	d	a	$	a	a	b	a	a	a	b	b	a
10	3	b	a	a	a	b	b	a	a	c	d	a	$	a	a
11	8	b	a	a	c	d	a	$	a	a	b	a	a	a	b
12	7	b	b	a	a	c	d	a	$	a	a	b	a	a	a
13	11	c	d	a	$	a	a	b	a	a	a	b	b	a	a
14	12	d	a	$	a	a	b	a	a	a	b	b	a	a	c

The **F** and **L** arrays for string **S** are shown above. String given by column **L** is the Burrows-Wheeler Transform of **S.** So, $\mathbf{S}^{\mathbf{bwt}}$ = **L**.

Searching for pattern “aaab” in string S using Backward_Search algorithm:

C(c) denote the number of occurrences in S of characters alphabetically smaller than c, and Occ(L,i,c) denote the number of occurrences of character c in L[1,i]

Initialization:

i = 4; sp = 1; ep = 14

So, <sp, ep> = <1, 4>

Loop iteration 1:

i = 4; <sp, ep> = <1, 4>

c   = P[4] = ‘b’

sp   = C[c] + Occ(L, sp-1, c) + 1

         = C[‘b’] + Occ(L, 0, ‘b’) + 1

         = 9 + 0 + 1

= 10
ep = C[c] + Occ(L, ep, c)
= C[‘b’] + Occ(L, 14, ‘b’)
= 9 + 3
= 12
So, <sp, ep> = <10, 12>

Loop iteration 2:
i = 3; <sp, ep> = <10, 12>
c = P[3] = ‘a’
sp = C[c] + Occ(L, sp-1, c) + 1
= C[‘a’] + Occ(L, 9, ‘a’) + 1
= 1 + 5 + 1
= 7
ep = C[c] + Occ(L, ep, c)
= C[‘a’] + Occ(L, 12, ‘a’)
= 1 + 7
= 8
So, <sp, ep> = <7, 8>

Loop iteration 3:
i = 2; <sp, ep> = <7, 8>
c = P[2] = ‘a’
sp = C[c] + Occ(L, sp-1, c) + 1
= C[‘a’] + Occ(L, 6, ‘a’) + 1
= 1 + 2 + 1
= 4
ep = C[c] + Occ(L, ep, c)
= C[‘a’] + Occ(L, 8, ‘a’)
= 1 + 4
= 5
So, <sp, ep> = <4, 5>

Loop iteration 4:
i = 1; <sp, ep> = <4, 5>
c = P[1] = ‘a’
sp = C[c] + Occ(L, sp-1, c) + 1

= C[‘a’] + Occ(L, 3, ‘a’) + 1
= 1 + 1 + 1
= 3

ep = C[c] + Occ(L, ep, c)
= C[‘a’] + Occ(L, 5, ‘a’)
= 1 + 2
= 3

So, <sp, ep> = <3, 3>

So, pattern ‘aaab’ is present at index position 4 of string S.

#Question 5:

<u>Solution:</u>

Given, S = aabaaabbaacda$

After removing the terminating character ‘$’, initial alphabet = {a, b, c, d}