Assignment 06_Theory
Algorithms for Sequence Analysis
Summer Semester 2022

Sreeja Chidambaram_7015645
Jibin Varghese_7015777

06.1: Distinct substrings of a string (4 Theory)

Task

Give a linear-time algorithm to compute the total number of distinct substrings of a string *s* (without sentinel).

Hint

Think in terms of *s*$ and the enhanced suffix array,
but do not count substrings with the sentinel $.

Example

baaa has 7 distinct substrings: a, b, aa, ba, aaa, baa, baaa.

Solution:

Example 1: "baaa"

Out of all n(n+1)/2 substrings "**baaa**" has 7 distinct substrings.

1	2	3	4	5	6	7	8	9	10
b	a	**a**	**a**	ba	aa	**aa**	baa	aaa	baaa

10 → Total Substrings
7 → Unique Substrings

Now, let's construct LCP in linear time using Kasai's Algorithm

r	pos[r]	lcp[r]	T[pos[r]:]
0	3	0	a
1	2	1	aa
2	1	2	aaa
3	0	0	baaa

The lcp value at index 1: is 2 (a & aa are repeated)
The lcp value at index 2: is 1 a is repeated)
The sum of all lcp values is i.e Σ lcp[r] is (0+1+2+0) = 3
The total number of distinct substrings is 10 -3 = 7, which means the difference between the total number of substrings (n(n+1)/2) and the sum of all lcp values (Σ lcp[r]).

Therefore, the total number of distinct substrings of a string *s of length n* (without sentinel) can be generalized as

n(n+1)/2 – Σ lcp [r]

Example 2: "bananas"

Out of all n(n+1)/2 substrings "**bananas**" has 22 distinct substrings.

1	2	3	4	5	6	7	8	9	10
b	a	n	**a**	**n**	**a**	s	ba	an	na
11	12	13	14	15	16	17	18	19	20
an	**na**	as	ban	ana	nan	**ana**	nas	bana	anan
21	22	23	24	25	26	27	28		
nana	anas	banan	anana	nanas	banana	ananas	bananas		

r	pos[r]	lcp[r]	T[pos[r]:]
0	1	0	ananas
1	3	3	anas
2	5	1	as
3	0	0	bananas
4	2	0	nanas
5	4	2	nas
6	6	0	s

total number of distinct substrings of a bananas is

n(n+1)/2 - – Σ lcp [r]

28 – 6 = 22

```python
import numpy as np
def build_suffixarray_naive(T):
    suffixes = lambda p: T[p:]
    return sorted(range(len(T)), key=suffixes)
def invert_sa(pos, n):
    rank = [-1] * n
    for r in range(n):
        rank[pos[r]] = r
    return rank
def lcp(pos, T):
    n = len(pos)
    lcp = [0] * (n)
    rank = invert_sa(pos, n)
    l = 0  # current common prefix length
    for p in range(n - 1):
        r = rank[p]
        # within length of T, and characters agree?
        while (pos[r - 1] + l < len(T) and
               p + l < len(T) and
               T[p + l] == T[pos[r - 1] + l]):
            l += 1
        lcp[r] = l
        l = max(l - 1, 0)  # next suffix: lose first character
    return lcp
if __name__ == '__main__':
```

```python
if __name__ == '__main__':
    T = "bananas"
    n = len(T)
    print(lcp(build_suffixarray_naive(T), T))
    tot = np.sum(lcp(build_suffixarray_naive(T), T))
    totSub = n * (n + 1) / 2
    print("The total number of distinct substrings of a string s is: " + str(totSub - tot))
```

Output for “bananas”

```
Kasai's Algorithm-Distinct substrings
C:\Users\gopis\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/gopis/PycharmProjects/ASA/Kasai's Algorithm-Distinct substrings.py"
[0, 3, 1, 0, 0, 2, 0]
The total number of distinct substrings of a string s is: 22.0

Process finished with exit code 0
```

Output for “baaa”

```
if __name__ == '__main__':
    T = "baaa"
    n = len(T)
    print(lcp(build_suffixarray_naive(T), T))
    tot = np.sum(lcp(build_suffixarray_naive(T), T))
    totSub = n * (n + 1) / 2
    print("The total number of distinct substrings of a string s is: " + str(totSub - tot))
```

```
if __name__ == '__main__'
Kasai's Algorithm-Distinct substrings
C:\Users\gopis\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/gopis/PycharmProjects/ASA/Kasai's Algorithm-Distinct substrings.py"
[0, 1, 2, 0]
The total number of distinct substrings of a string s is: 7.0

Process finished with exit code 0
```

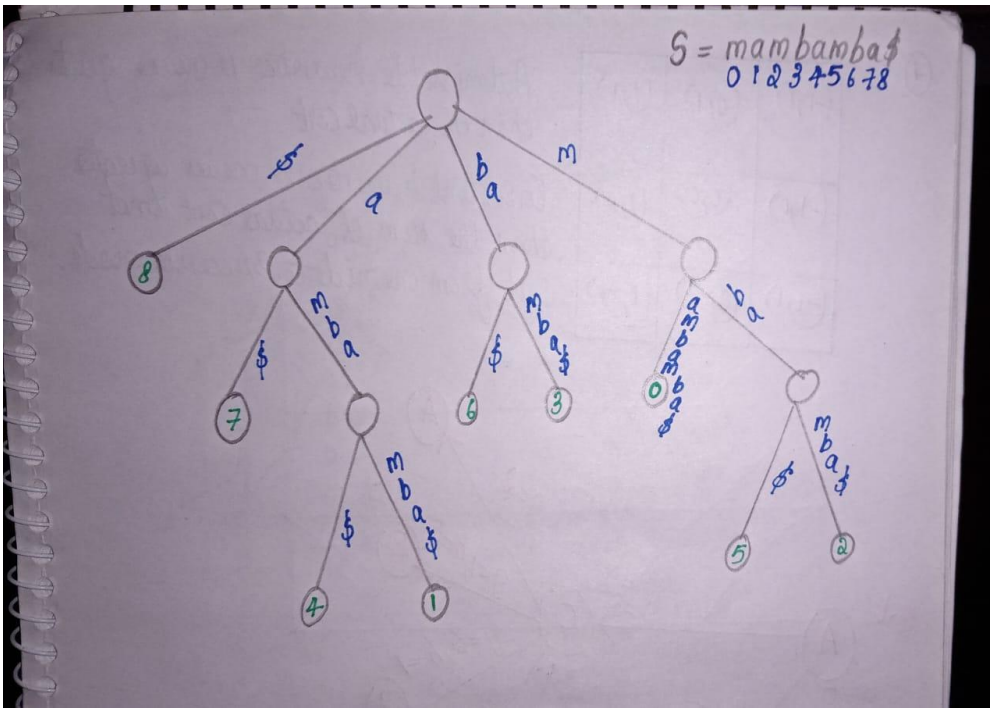
06.2 Supermaximal repeats (8 Theory)

Task

Give a linear-time algorithm to find all **supermaximal repeats** of a string *s*, using its enhanced suffix array and the definitions below. To develop the algorithm, characterize supermaximal repeats in terms of intervals in the enhanced suffix array.

Solution:

Let's first think about this problem in terms of suffix tree. Here which inner nodes are supermaximal repeats?



We can find the supermaximal repeats by considering the inner nodes which has only leaves. An inner node selected should be a left diverse (if there are two leaves namely i and j then s[i-1]! = s[j-1])

Here in our suffix tree, if we consider the node “amba” the corresponding I and j values are 1 and 4. As stated above, the strings at position i-1 (0) and j-1(3) are m and b which are not equal. Hence, we can say that “amba” is a super maximal repeat of the string s.

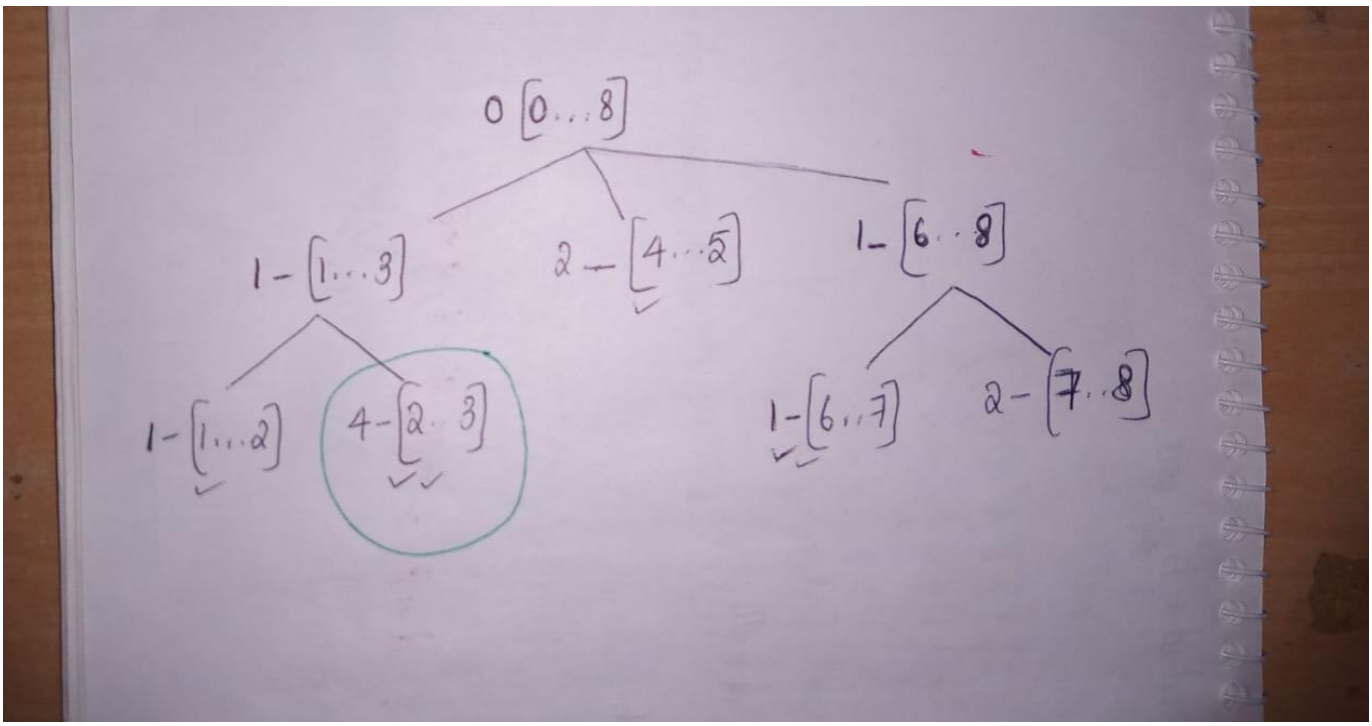
Similarly, if we consider the node “mba”, the strings at position i-1 (4) and j-1 (1) are m which is equal to each other. Therefore, we can conclude that mba is not a supermaximal repeat.

Now consider the below table, here the local maxima in the lcptab are 4, 2, 3.
And the inner nodes in the suffix tree can be represented as intervals in the lcp array.
Since we consider inner nodes with only leaves, that becomes our local lcp maxima. In our example it is (4, 2, 3).

After calculating the bwttab, if we consider all the pairs of suftab and corresponding substring pairs where all bwt symbols differ we get the pair 1,4 which is our supermaximal of s. i.e; (1,4,4) “amba”

i	suftab	lcptab	bwttab	T[pos[r]...]
0	8	0	a	$
1	7	0	b	a$
2	4	1	b	amba$
3	1	4	m	ambamba$
4	6	0	m	ba$
5	3	2	m	bamba$
6	0	0		mambamba$
7	5	1	a	mba$
8	2	3	a	mbamba$

The lcp interval as represented below will help us to find the local lcp maxima and finally find the supermaximal pair as explained above.

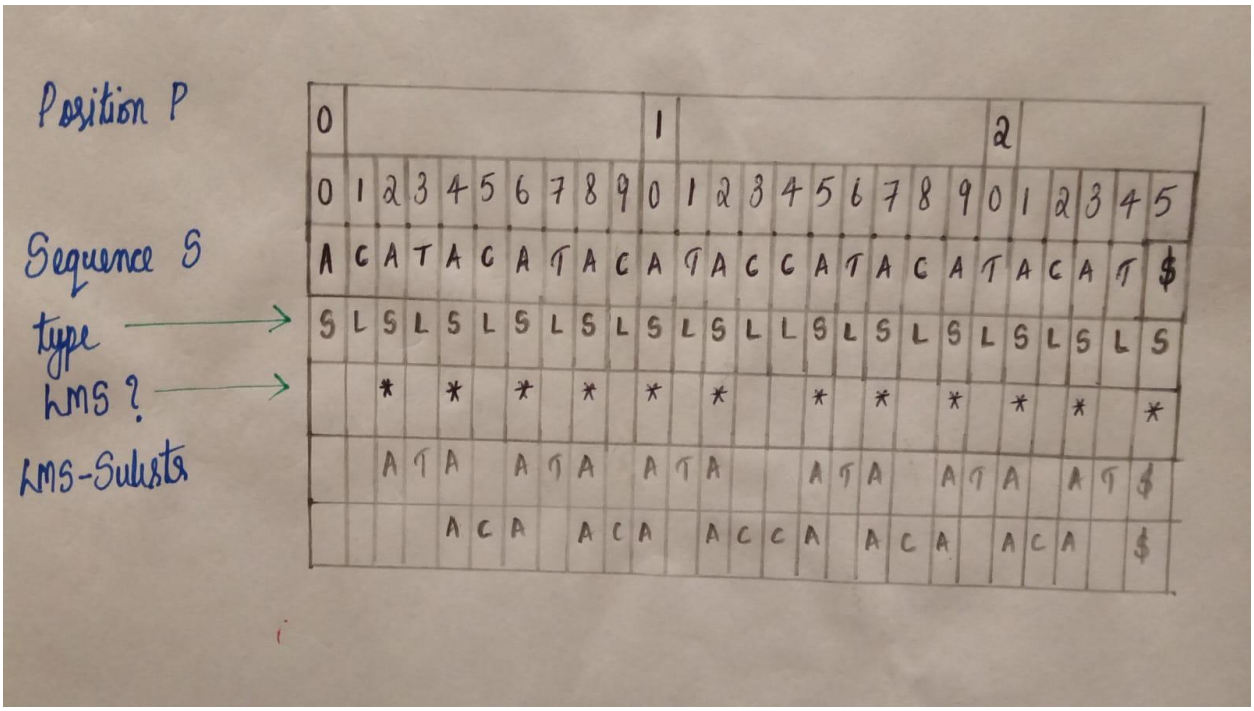


06.3: SAIS Example (4 Theory)

For text *T* = ACATACATACATACCATACATACAT$, do the following:

1 Compute the type array, LMS-substrings, and the string's representation based on the reduced alphabet.

Solution:

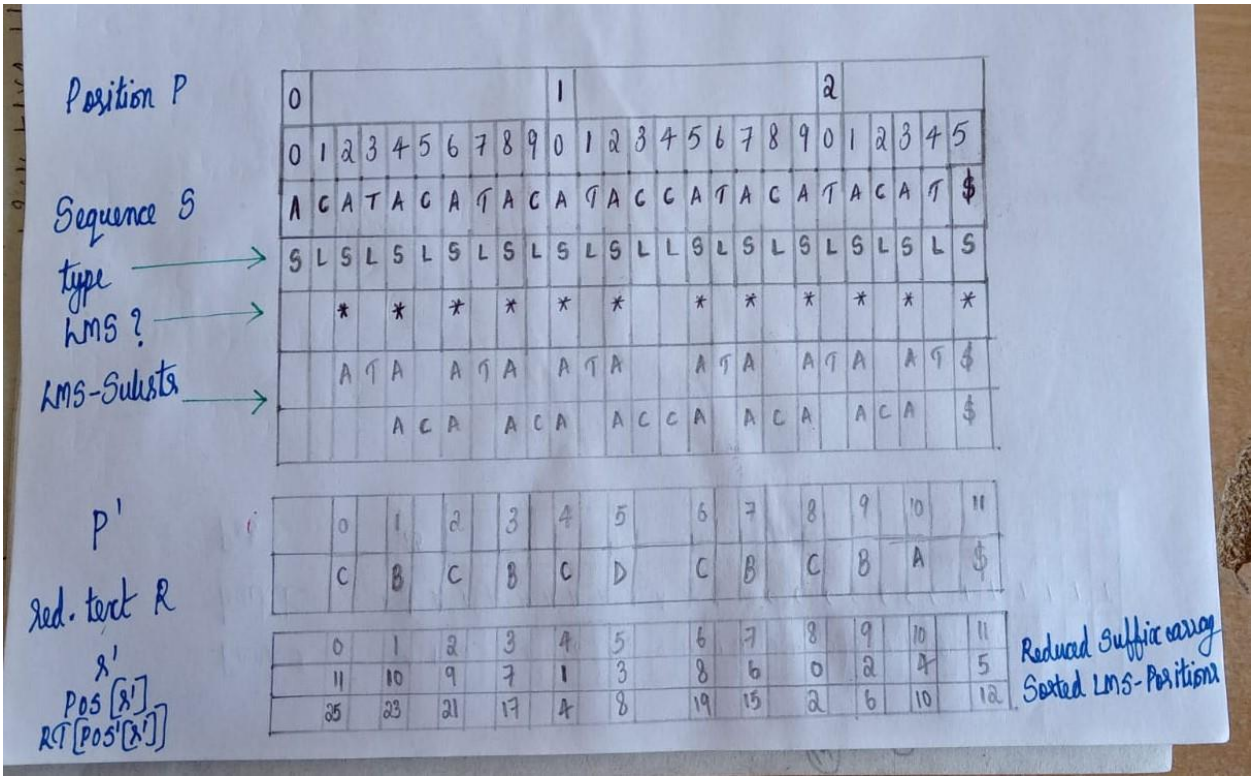


2 Is the order of suffixes at LMS-positions determined by the LMS-substrings computed in the previous subtask?
If yes, show the suffix array of the reduced text.
If not, illustrate how the induced sorting algorithm recurses to find the suffix array of the reduced text,

Solution:

The answer is “NO” here, because our LMS substrings are identical. Therefore, we need to compute suffix array of reduced text (recursively with SAIS), use that to infer correct order of LMS suffixes.

However, suppose we had distinct LMS substrings the order of the sub strings would already precisely determine the order of the suffixes.



3 Compute the suffix array by induced sorting from the sorted LMS suffixes.
It is recommended to do this task by pen and paper.

Solution:

Suffix array of the original text:

r	pos[r]	T[pos[r]:]
0	25	$
1	21	ACAT$
2	17	ACATACAT$
3	0	ACATACATACATACCATACATACAT$
4	4	ACATACATACCATACATACAT$
5	8	ACATACCATACATACAT$
6	12	ACCATACATACAT$
7	23	AT$
8	19	ATACAT$
9	15	ATACATACAT$
10	2	ATACATACATACCATACATACAT$
11	6	ATACATACCATACATACAT$
12	10	ATACCATACATACAT$
13	22	CAT$
14	18	CATACAT$
15	14	CATACATACAT$
16	1	CATACATACATACCATACATACAT$
17	5	CATACATACCATACATACAT$
18	9	CATACCATACATACAT$
19	13	CCATACATACAT$
20	24	T$
21	20	TACAT$
22	16	TACATACAT$
23	3	TACATACATACCATACATACAT$
24	7	TACATACCATACATACAT$
25	11	TACCATACATACAT$

Reduced Suffix Array

r	pos[r]	T[pos[r]:]
0	11	$
1	10	A$
2	9	BA$
3	7	BCBA$
4	1	BCBCDCBCBA$
5	3	BCDCBCBA$
6	8	CBA$
7	6	CBCBA$
8	0	CBCBCDCBCBA$
9	2	CBCDCBCBA$
10	4	CDCBCBA$
11	5	DCBCBA$

Calculating the suffix array by induced sorting from the sorted LMS suffixes.

Sorting the S positions - Induced Sorting

Rank s	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Bucket	$	A	A	A	A	A	A	A	A	A	A	A	A	C	C	C	C	C	C	C	T	T	T	T	T	T
POS(2)	25		23	21	17	4	8	19	15	2	6	10	12	22	18	14	1	5	9	13	24	20	16	3	7	11
POS	25												10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25											6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25										2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25									15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25								19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25							23	19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25						12	23	19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25					8	12	23	19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25				4	8	12	23	19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25			0	4	8	12	23	19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25		17	0	4	8	12	23	19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11
	25	21	17	0	4	8	12	23	19	15	2	6	10	22	18	14	1	5	9	13	24	20	16	3	7	11