Strings

William Fiset

Outline

- Knuth–Morris–Pratt (KMP) string matching algorithm
- What is a suffix array?
- Suffix array introduction
- Suffix array construction, $n^2\log(n)$
- Suffix array construction, $n\log^2(n)$
- Longest Common Prefix (LCP) array construction using Kasai Algorithm

Outline

- Counting all unique substrings
- Longest Repeated Substring (LRS)
- LCP Array Kattis problem
- Efficient substring containment algorithm
- Longest Common Substring (LCS)
- Suggested problems

KMP

Knuth–Morris–Pratt (KMP) named after its three authors is an efficient pattern matching algorithm which is able to find all the occurrences of a pattern in a given text. This is all done in O(n+m) time instead of the naive O(nm) way.

KMP is divided into two stages. In the first stage we generate partial match array and in the second stage we actually do the matching.

KMP Stage 1

For each position in the pattern find the length of the longest **proper prefix** which is equal to a **proper suffix** starting at position i.

KMP Stage 1

A B A A A B A A B A A B A A

KMP Stage 1

A B A A A B A A B A A B A A
0

KMP Stage 1

A B A A A B A A B A A B A A
0 0

KMP Stage 1

A B A A A B A A B A A B A A
0 0 1

KMP Stage 1

A B A A A B A A B A A B A A

0 0 1 1

KMP Stage 1

A B A A A B A A B A A B A A
0 0 1 1 1

KMP Stage 1

A B A A A B A A B A A B A A

0 0 1 1 1 2

KMP Stage 1

A B A A A B A A B A A B A A

0 0 1 1 1 2 3

KMP Stage 1

A B A A A B A A B A A B A A
0 0 1 1 1 2 3 4

KMP Stage 1

A B A A A B A A B A A B A A

0 0 1 1 1 2 3 4 2

KMP Stage 1

A	B	A	A	A	B	A	A	B	A	A	B	A	A
0	0	1	1	1	2	3	4	2	3				

KMP Stage 1

A	B	A	A	A	B	A	A	B	A	A	B	A	A
0	0	1	1	1	2	3	4	2	3	4			

KMP Stage 1

A B A A A B A A B A A B A A

0 0 1 1 1 2 3 4 2 3 4 2

KMP Stage 1

A	B	A	A	A	B	A	A	B	A	A	B	A	A
0	0	1	1	1	2	3	4	2	3	4	2	3	

KMP Stage 1

A B A A A B A A B A A B A A

0 0 1 1 1 2 3 4 2 3 4 2 3 4

KMP Stage 1

0 0 1 1 1 2 3 4 2 3 4 2 3 4

Partial match array

KMP Stage 2

C A T C A C A C A A C T C A C
A C A C A T A

Partial match array:
P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

Shift amount = 1

C A T C A C A C A A C T C A C
A C A C A T A

Partial match array:
P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

Shift amount = 1–P[1–1] = 1–P[0] = 1–0 = 1

C A T C A C A C A A C T C A C
  A C A C A T A

Partial match array:
P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

Shift amount = 1

C A T C A C A C A A C T C A C
A C A C A T A

Partial match array:
P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

Shift amount = 1

C A T C A C A C A A C T C A C
A C A C A T A

Partial match array:
P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

Shift amount = 5−P[5−1] = 5−P[4] = 5−3 = 2

C A T C A C A C A A C T C A C
A C A C A T A

Partial match array:
P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

Shift amount = 3−P[3−1] = 3−P[2] = 3−1 = 2

```
C A T C A C A C A A C T C A C
            A C A C A T A
```

Partial match array:

P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

C A T C A C A C A A C T C A C

A C A C A T A

Partial match array:

P = [0, 0, 1, 2, 3, 0, 1]

KMP Stage 2

```
C A T C A C A C A A C T C A C
                A C A C A T A
```

Partial match array:

P = [0, 0, 1, 2, 3, 0, 1]

The Suffix Array

What is a Suffix Array?

A **suffix array** is a space efficient alternative to **suffix tree** which itself is a compressed version of a **trie**.

Suffix arrays can do everything suffix trees can, with some additional information such as a Longest Common Prefix (LCP) array

What is a Suffix Array?

A **suffix array** is a space efficient alternative to **suffix tree** which itself is a compressed version of a **trie**.

Index	Suffix
0	unicorn
1	nicorn
2	icorn
3	corn
4	orn
5	rn
6	n

Text: ‘unicorn’

What is a Suffix Array?

A **suffix array** is a space efficient alternative to **suffix tree** which itself is a compressed version of a **trie**.

Index	Suffix
0	unicorn
1	nicorn
2	icorn
3	corn
4	orn
5	rn
6	n

Index	Suffix
3	corn
2	icorn
6	n
1	nicorn
4	orn
5	rn
0	unicorn

What is a Suffix Array?

A **suffix array** is a space efficient alternative to **suffix tree** which itself is a compressed version of a **trie**.

The suffix array

Index	Suffix
3	corn
2	icorn
6	n
1	nicorn
4	orn
5	rn
0	unicorn

When and where is a Suffix Array used?

Heavily used in the field of bioinformatics for DNA sequencing.

Used to compute the Burrows–Wheeler Transformation (BWT) used in data compression.

Find all occurrences of a substring in the larger text

Longest repeated substring

Quickly determine if a substring occurs in a piece of text

When and where is a Suffix Array used?

Most frequently occurring substrings

Counting the number of unique substrings

The longest common substring(s) amongst multiple strings

Shortest unique string in a text (this problem comes up in bioinformatics)

Circular string linearization (given a string find the smallest lexicographic rotation)

Suffix Array Construction

Suffix Array Naive Construction

The naive construction of a suffix array is as follows: Generate all the suffixes of our text (T) and sort them all. However, **comparing two strings takes O(n) time**, so the whole process along with using a sorting algorithm such as merge sort takes $O(n^2 \log(n))$ time and **a lot of space**.

Suffix Array
$O(n\log^2(n))$ Construction

Idea: We want to sort the suffixes of the original string by the first 2^i characters until $2^i > N$. This works because each suffix has something in common with another suffix. These are not random strings.

Suffix Array
$O(n\log^2(n))$ Construction

Text: ‘abracadabra’

a	a	a	a
abracadabra	abracadabra	abra	abra
abra	abra	abracadabra	abracadabra
acadabra	acadabra	acadabra	acadabra
adabra	adabra	adabra	adabra
bracadabra	bra	bra	bra
bra	bracadabra	bracadabra	bracadabra
cadabra	cadabra	cadabra	cadabra
dabra	dabra	dabra	dabra
racadabra	ra	ra	ra
ra	racadabra	racadabra	racadabra
Sorted up to the **first** character	Sorted up to the **second** character	Sorted up to the **fourth** character	Sorted up to the **eighth** character

SA construction setup

Fill table with the values for
the first letter of each suffix

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0										

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1									

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17								

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0							

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2						

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0					

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3				

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0			

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1		

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

SA construction setup

Fill table with the values for the first letter of each suffix. This will assign a relative ranking system.

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	
1	1	
2	17	
3	0	
4	2	
5	0	
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign -1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

0	0	
1	1	
2	17	
3	0	
4	2	
5	0	
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	1
1	1	
2	17	
3	0	
4	2	
5	0	
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	**17**	0	2	0	3	0	1	17	0

0	0	**1**
1	1	**17**
2	17	
3	0	
4	2	
5	0	
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	1
1	1	17
2	17	0
3	0	
4	2	
5	0	
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	
5	0	
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	**0**
5	0	
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	**0**
7	0	
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	0
7	0	1
8	1	
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	0
7	0	1
8	1	17
9	17	
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	0
7	0	1
8	1	17
9	17	0
10	0	

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	0
7	0	1
8	1	17
9	17	0
10	0	**-1**

Offset = 2^0 = 1

For each suffix at index i **get the value of the character at i + 1**. If index i + 1 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

0	0	**1**
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	0
7	0	1
8	1	17
9	17	0
10	0	-1

Offset = 2^0 = 1

Sort by the second column, then by the third column. This sorts all the suffixes by their first character.

0	0	1
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	0
7	0	1
8	1	17
9	17	0
10	0	-1

Sort →

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the index of the first pair give it a ranking of zero.

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the index of the first pair give it a pair ranking of zero.

Pair Rank Counter = 0

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
										0

10	**0**	**-1**
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 1

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1										0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 1

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1							1			0

10	0	-1
0	**0**	**1**
7	**0**	**1**
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 2

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1			2				1			0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 3

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1			2		3		1			0

10	0	-1
0	0	1
7	0	1
3	**0**	**2**
5	**0**	**3**
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 4

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4		2		3		1			0

10	0	-1
0	0	1
7	0	1
3	0	2
5	**0**	**3**
1	**1**	**17**
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 4

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4		2		3		1	4		0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	**1**	**17**
8	**1**	**17**
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 5

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4		2	5	3		1	4		0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	**1**	**17**
4	**2**	**0**
6	3	0
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 6

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4		2	5	3	6	1	4		0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	**2**	**0**
6	**3**	**0**
2	17	0
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 7

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4		0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	**3**	**0**
2	**17**	**0**
9	17	0

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 7

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	**17**	**0**
9	**17**	**0**

Offset = 2^0 = 1

For the next pairs, if the **pairs are different increment the counter** (meaning the last pair is better than the current pair). This process assigns a **relative ranking** to each pair for the next iteration.

Pair Rank Counter = 7

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

10	0	-1
0	0	1
7	0	1
3	0	2
5	0	3
1	1	17
8	1	17
4	2	0
6	3	0
2	17	0
9	17	0

Offset = 2^1 = 2

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

10		
0		
7		
3		
5		
1		
8		
4		
6		
2		
9		

Offset = 2^1 = 2

a	b	r	a	c	a	d	a	b	r	a
0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	
1	4	
2	7	
3	2	
4	5	
5	3	
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	
1	4	
2	7	
3	2	
4	5	
5	3	
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	
2	7	
3	2	
4	5	
5	3	
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	
3	2	
4	5	
5	3	
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign -1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	
4	5	
5	3	
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	**3**	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	**3**
4	5	
5	3	
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	1
6	6	
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign -1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	**4**	7	0

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	1
6	6	**4**
7	1	
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	1
6	6	4
7	1	7
8	4	
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	1
6	6	4
7	1	7
8	4	0
9	7	
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	1
6	6	4
7	1	7
8	4	0
9	7	**-1**
10	0	

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign −1 to give that suffix sorting priority

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	1
6	6	4
7	1	7
8	4	0
9	7	-1
10	0	**-1**

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

Offset = 2^1 = 2

For each suffix at index i **get the value of the character at i + 2**. If index i + 2 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0

0	1	7
1	4	2
2	7	5
3	2	3
4	5	6
5	3	1
6	6	4
7	1	7
8	4	0
9	7	-1
10	0	-1

Offset = 2^1 = 2

Sort by the second column, then by the third column. This sorts all the suffixes by their first character, then by their first two characters.

0	0	1
1	1	17
2	17	0
3	0	2
4	2	0
5	0	3
6	3	0
7	0	1
8	1	17
9	17	0
10	0	-1

Sort →

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 0

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
										0

10	**0**	**-1**
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 1

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1										0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 1

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1							1			0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 2

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1			2				1			0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 3

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1			2		3		1			0

10	0	-1
0	1	7
7	1	7
3	**2**	**3**
5	**3**	**1**
8	4	0
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 4

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1			2		3		1	4		0

10	0	-1
0	1	7
7	1	7
3	2	3
5	**3**	**1**
8	**4**	**0**
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 5

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5		2		3		1	4		0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	**4**	**0**
1	**4**	**2**
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 6

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5		2	6	3		1	4		0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	**4**	**2**
4	**5**	**6**
6	6	4
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 7

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5		2	6	3	7	1	4		0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	**5**	**6**
6	**6**	**4**
9	7	-1
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 8

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5		2	6	3	7	1	4	8	0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	**6**	**4**
9	**7**	**-1**
2	7	5

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 9

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	6	4
9	**7**	**-1**
2	**7**	**5**

Offset = 2^1 = 2

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 9

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

10	0	-1
0	1	7
7	1	7
3	2	3
5	3	1
8	4	0
1	4	2
4	5	6
6	6	4
9	7	-1
2	7	5

Offset = 2^2 = 4

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Offset = 2^2 = 4

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	**5**	**9**	**2**	**6**	**3**	**7**	**1**	**4**	**8**	**0**

0	**1**	
1	**5**	
2	**9**	
3	**2**	
4	**6**	
5	**3**	
6	**7**	
7	**1**	
8	**4**	
9	**8**	
10	**0**	

`0ffset` = 2^2 = 4

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	
1	5	
2	9	
3	2	
4	6	
5	3	
6	7	
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	
2	9	
3	2	
4	6	
5	3	
6	7	
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	
3	2	
4	6	
5	3	
6	7	
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	
4	6	
5	3	
6	7	
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	
5	3	
6	7	
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign -1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	
6	7	
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	**0**

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	**0**
7	1	
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	0
7	1	-1
8	4	
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	0
7	1	-1
8	4	**-1**
9	8	
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	0
7	1	-1
8	4	-1
9	8	-1
10	0	

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	0
7	1	-1
8	4	-1
9	8	-1
10	0	**-1**

Offset = 2^2 = 4

For each suffix at index i **get the value of the character at i + 4**. If index i + 4 is out of bounds assign −1 to give that suffix sorting priority

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	0
7	1	-1
8	4	-1
9	8	-1
10	0	-1

Offset = 2^2 = 4

Sort by the second column, then by the third column. This sorts all the suffixes by their first two characters, then by their first four characters.

0	1	6
1	5	3
2	9	7
3	2	1
4	6	4
5	3	8
6	7	0
7	1	-1
8	4	-1
9	8	-1
10	0	-1

Sort →

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 0

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
										0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 1

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
							1			0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 2

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2							1			0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 3

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2			3				1			0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 4

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2			3		4		1			0

10	0	-1
7	1	-1
0	1	6
3	**2**	**1**
5	**3**	**8**
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 5

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2			3		4		1	5		0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	**3**	**8**
8	**4**	**-1**
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 6

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	6		3		4		1	5		0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	**4**	**-1**
1	**5**	**3**
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 7

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	6		3	7	4		1	5		0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 8

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	6		3	7	4	8	1	5		0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	**6**	**4**
6	**7**	**0**
9	8	-1
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 9

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	6		3	7	4	8	1	5	9	0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	**7**	**0**
9	**8**	**-1**
2	9	7

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 10

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	6	10	3	7	4	8	1	5	9	0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	**8**	**-1**
2	**9**	**7**

Offset = 2^2 = 4

For the index of the first pair give it a ranking of zero.

Pair Rank Counter = 10

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	6	10	3	7	4	8	1	5	9	0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^3 = 8

Our suffixes are already sorted, so we can stop early without doing offset = 8

a	b	r	a	c	a	d	a	b	r	a
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	6	10	3	7	4	8	1	5	9	0

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Offset = 2^3 = 8

The suffix array can be found in the bottom of our matrix

a b r a c a d a b r a

0	1	2	3	4	5	6	7	8	9	10
0	1	17	0	2	0	3	0	1	17	0
1	4	7	2	5	3	6	1	4	7	0
1	5	9	2	6	3	7	1	4	8	0
2	**6**	**10**	**3**	**7**	**4**	**8**	**1**	**5**	**9**	**0**

Suffix Array

10	0	-1
7	1	-1
0	1	6
3	2	1
5	3	8
8	4	-1
1	5	3
4	6	4
6	7	0
9	8	-1
2	9	7

Construction Optimizations

You only ever need to **maintain two rows** of data in the larger table, this will save a lot of room.

You can stop sorting all your suffixes once the **pair counter reaches n-1**. This means the sorting is done and there is no longer any ambiguity about which suffixes come before any others.

The Kasai Algorithm for LCP Array Generation

Kasai Algorithm

The Kasai Algorithm is able to generate the LCP array for a text given that **the suffix array has already be constructed prior**.

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0		aabb
1		abaabb
2		ababaabb
3		abababaabb
4		abb
5		b
6		baabb
7		babaabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	**a**abb
1		**a**baabb
2		ababaabb
3		abababaabb
4		abb
5		b
6		baabb
7		babaabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	**aba**abb
2		**aba**baabb
3		abababaabb
4		abb
5		b
6		baabb
7		babaabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	**ababa**abb
3		**ababa**baabb
4		abb
5		b
6		baabb
7		babaabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	ababaabb
3	2	**ab**abababb
4		**ab**b
5		b
6		baabb
7		babaabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	ababaabb
3	2	abababaabb
4	0	abb
5		b
6		baabb
7		babaabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	ababaabb
3	2	abababaabb
4	0	abb
5	1	**b**
6		**b**aabb
7		babaabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	ababaabb
3	2	abababaabb
4	0	abb
5	1	b
6	2	**ba**abb
7		**ba**baabb
8		bababaabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	ababaabb
3	2	abababaabb
4	0	abb
5	1	b
6	2	baabb
7	4	**baba**abb
8		**baba**baabb
9		bb

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	ababaabb
3	2	abababaabb
4	0	abb
5	1	b
6	2	baabb
7	4	babaabb
8	1	**b**ababaabb
9		**b**b

Kasai Algorithm

LCP Array Review

The LCP Array contains the length of the **Longest Common Prefix** between adjacent pairs of suffixes.

i	LCP[i]	
0	1	aabb
1	3	abaabb
2	5	ababaabb
3	2	abababaabb
4	0	abb
5	1	b
6	2	baabb
7	4	babaabb
8	1	bababaabb
9	0	bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6		aabb
1	4		abaabb
2	2		ababaabb
3	0		abababaabb
4	7		abb
5	9		b
6	5		baabb
7	3		babaabb
8	1		bababaabb
9	8		bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6		aabb
1	4		abaabb
2	2		ababaabb
3	0		abababaabb
4	7		abb
5	9		b
6	5	0	baabb
7	3		babaabb
8	1		bababaabb
9	8		bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6		aabb
1	4		abaabb
2	2		ababaabb
3	0		abababaabb
4	7	1	abb
5	9		b
6	5	0	baabb
7	3		babaabb
8	1		bababaabb
9	8		bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6		aabb
1	4		abaabb
2	2	2	ababaabb
3	0		abababaabb
4	7	1	abb
5	9		b
6	5	0	baabb
7	3		babaabb
8	1		bababaabb
9	8		bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6	3	aabb
1	4		abaabb
2	2	2	ababaabb
3	0		abababaabb
4	7	1	abb
5	9		b
6	5	0	baabb
7	3		babaabb
8	1		bababaabb
9	8		bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6	3	aabb
1	4		abaabb
2	2	2	ababaabb
3	0		abababaabb
4	7	1	abb
5	9		b
6	5	0	baabb
7	3	4	babaabb
8	1		bababaabb
9	8		bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6	3	aabb
1	4		abaabb
2	2	2	ababaabb
3	0		abababaabb
4	7	1	abb
5	9		b
6	5	0	baabb
7	3	4	babaabb
8	1		bababaabb
9	8	5	bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6	3	aabb
1	4		abaabb
2	2	2	ababaabb
3	0		abababaabb
4	7	1	abb
5	9	6	b
6	5	0	baabb
7	3	4	babaabb
8	1		bababaabb
9	8	5	bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6	3	aabb
1	4		abaabb
2	2	2	ababaabb
3	0	7	abababaabb
4	7	1	abb
5	9	6	b
6	5	0	baabb
7	3	4	babaabb
8	1		bababaabb
9	8	5	bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6	3	aabb
1	4	8	abaabb
2	2	2	ababaabb
3	0	7	abababaabb
4	7	1	abb
5	9	6	b
6	5	0	baabb
7	3	4	babaabb
8	1		bababaabb
9	8	5	bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	6	3	aabb
1	4	8	abaabb
2	2	2	ababaabb
3	0	7	abababaabb
4	7	1	abb
5	9	6	b
6	5	0	baabb
7	3	4	babaabb
8	1	9	bababaabb
9	8	5	bb

Kasai Algorithm

Let inv be the inverse lookup of SA[i], namely inv[SA[i]] = i

i	SA[i]	inv[i]	
0	**6**	**3**	aabb
1	**4**	**8**	abaabb
2	**2**	**2**	ababaabb
3	**0**	**7**	abababaabb
4	**7**	**1**	abb
5	**9**	**6**	b
6	**5**	**0**	baabb
7	**3**	**4**	babaabb
8	**1**	**9**	bababaabb
9	**8**	**5**	bb

Kasai Algorithm

ababababaabb	aabb
babababaabb	abaabb
abababaabb	ababaabb
bababaabb	abababaabb
ababaabb	abb
babaabb	b
abaabb	baabb
baabb	babaabb
aabb	bababaabb
abb	bb

Kasai Algorithm

ababababaabb	aabb
babababaabb	abaabb
abababaabb	abababaabb
bababaabb	ababababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	bababaabb
b	bb

Kasai Algorithm

ababababaabb	aabb
babababaabb	abaabb
abababaabb	ababaabb
bababaabb	ababababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	bababaabb
b	bb

Kasai Algorithm

abababaabb	aabb
bababaabb	abaabb
ababaabb	ababaabb
babaabb	abababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	bababaabb
b	bb

Kasai Algorithm

ababababaabb	aabb
babababaabb	abaabb
abababaabb	ababaabb
babaabb	abababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	bababaabb
b	bb

Kasai Algorithm

ababababaabb	aabb
babababaabb	abaabb
ababaabb	ababaabb
babaabb	ababababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	babababaabb
b	bb

Kasai Algorithm

ababababaabb	aabb
babababaabb	abaabb
abababaabb	ababaabb
babaabb	ababababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	babababaabb
b	bb

Kasai Algorithm

abababaabb	aabb
bababaabb	abaabb
ababaabb	ababaabb
babaabb	abababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	bababaabb
b	bb

Kasai Algorithm

abababaabb	aabb
bababaabb	abaabb
ababaabb	ababaabb
babaabb	abababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	bababaabb
b	bb

Kasai Algorithm

```
ababababaabb      aabb
 bababaabb        abaabb
  ababaabb        ababaabb
   babaabb        ababababaabb
    abaabb        abb
     baabb        b
      aabb        baabb
       abb        babaabb
        bb        bababaabb
         b        bb
```

Kasai Algorithm

abababaabb	aabb
bababaabb	abaabb
ababaabb	ababaabb
babaabb	abababaabb
abaabb	abb
baabb	b
aabb	baabb
abb	babaabb
bb	bababaabb
b	bb

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

LCP_LEN = 0

Let **k** = SA[inv[i]-1]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

Let **k** = SA[inv[i]-1]

```
0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
```

```
0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
```

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

LCP_LEN = 0

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b

Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

LCP_LEN = 1

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

LCP_LEN = 2

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

LCP_LEN = 3

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

LCP_LEN = 4

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b

Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	5
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	
8	1	9	
9	8	5	

LCP_LEN = 5

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	
2	2	2	5
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 4

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	
6	5	0	
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 3

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b

Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 2

Let **k** = SA[inv[i]-1]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[k + LCP_LEN]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 1

Let **k** = SA[inv[i]-1]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[k + LCP_LEN]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 0

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 1

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 0

Let k = SA[inv[i]-1]

Problem: inv[6]-1 = -1

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[k + LCP_LEN]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 0

Let **k** = SA[inv[i]-1]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[k + LCP_LEN]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	
4	7	1	
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 1

Let **k** = SA[inv[i]-1]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[k + LCP_LEN]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	2
4	7	1	
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	
9	8	5	

LCP_LEN = 2

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	2
4	7	1	
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	1
9	8	5	

LCP_LEN = 1

Let **k** = SA[inv[i]-1]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[k + LCP_LEN]

0 1 2 3 4 5 6 7 8 9
a b a b a b a a b b
Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	2
4	7	1	0
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	1
9	8	5	

LCP_LEN = 0

Let **k** = SA[inv[i]-1]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[k + LCP_LEN]

0	1	2	3	4	5	6	7	8	9
a	b	a	b	a	b	a	a	b	b

Text[i + LCP_LEN]

Kasai Algorithm

i	SA[i]	inv[i]	LCP[i]
0	6	3	1
1	4	8	3
2	2	2	5
3	0	7	2
4	7	1	0
5	9	6	1
6	5	0	2
7	3	4	4
8	1	9	1
9	8	5	0

LCP
Array!

Counting all Unique Substrings

Unique Substrings

Text: 'AZAZA'

All n(n+1)/2 substrings:

A, AZ, AZA, AZAZ,

AZAZA, Z, ZA, ZAZ,

ZAZA, **A**, **AZ**, **AZA**,

Z, **ZA**, **A**

Number of unique substrings: 9

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ, AZAZA, Z, ZA, ZAZ, ZAZA, **A**, **AZ**, **AZA**, **Z**, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	AZA
0	AZAZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	**A**
3	**A**ZA
0	AZAZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	**A**
3	**A**ZA
0	AZAZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	**A**ZA
0	**A**ZAZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ, AZAZA, Z, ZA, ZAZ, ZAZA, **A**, **AZ**, **AZA**, **Z**, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	**A**ZA
0	**A**ZAZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	**AZ**A
0	**AZ**AZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

LCP	Sorted Suffixes
1	A
3	**AZ**A
0	**AZ**AZA
2	ZA
0	ZAZA

A, AZ, AZA, AZAZ, AZAZA, Z, ZA, ZAZ, ZAZA, **A**, **AZ**, **AZA**, **Z**, **ZA**, **A**

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	**AZA**
0	**AZA**ZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	**AZA**
0	**AZA**ZA
2	ZA
0	ZAZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	AZA
0	AZAZA
2	**Z**A
0	**Z**AZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	AZA
0	AZAZA
2	**Z**A
0	**Z**AZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	AZA
0	AZAZA
2	**ZA**
0	**ZA**ZA

Unique Substrings

Text: 'AZAZA'

A, AZ, AZA, AZAZ,
AZAZA, Z, ZA, ZAZ,
ZAZA, **A**, **AZ**, **AZA**,
Z, **ZA**, **A**

LCP	Sorted Suffixes
1	A
3	AZA
0	AZAZA
2	**ZA**
0	**ZA**ZA

Unique Substrings

$$\text{Unique substring count} = n(n+1)/2 - \sum_{i=0}^{n} lcp[i]$$

If text = 'AZAZA', n = 5

$$5(5+1)/2 - (1+3+0+2+0) = 9$$

Unique Substrings

Text: ’100000'

All n(n+1)/2 substrings:

1, 10, 100, 1000,

10000, 100000, 0, 00,

000, 0000, 00000, **0**,

00, **000**, **0000**, **0**, **00**,

000, **0**, **00**, **0**

Number of unique substrings: 11

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, **0**, **00**, **000**, **0000**, **0**, **00**, **000**, **0**, **00**, **0**

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	0000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, **0**, **00**, **000**, **0000**, **0**, **00**, **000**, **0**, **00**, **0**

LCP	Sorted Suffixes
0	100000
1	**0**
2	**0**0
3	000
4	0000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, 0, 00, 000, 0000, 0, 00, 000, 0, 00, 0

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	0000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, **0**, **00**, **000**, **0000**, **0**, **00**, **000**, **0**, **00**, **0**

LCP	Sorted Suffixes
0	100000
1	0
2	**00**
3	**00**0
4	0000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, 0, 00, 000, 0000, 0, 00, 000, 0, 00, 0

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	0000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, 0, 00, 000, 0000, 0, 00, 000, 0, 00, 0

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	0000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000,
10000, 100000, 0, 00,
000, 0000, 00000, 0,
00, 000, 0000, 0, 00,
000, 0, 00, 0

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	00000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, 0, 00, 000, 0000, 0, 00, 000, 0, 00, 0

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	0000
0	00000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, **0**, **00**, **000**, **0000**, **0**, **00**, **000**, **0**, **00**, **0**

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	**00**00
0	**00**000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, 0, 00, 000, 0000, 0, 00, 000, 0, 00, 0

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	00000
0	000000

Unique Substrings

Text: ’100000'

1, 10, 100, 1000, 10000, 100000, 0, 00, 000, 0000, 00000, 0, 00, 000, 0000, 0, 00, 000, 0, 00, 0

LCP	Sorted Suffixes
0	100000
1	0
2	00
3	000
4	0000
0	00000

Unique Substrings

$$\text{Unique substring count} = n(n+1)/2 - \sum_{i=0}^{n} lcp[i]$$

If text = '100000', n = 6

$$6(6+1)/2 - (0+1+2+3+4+0) = 11$$

Longest Repeated Substring (LRS)

LRS

In general we only care about repeated substrings that **occur at least twice**.

LRS(‘abcde’) = N/A

LRS(‘abracadabra’) = ‘abra’

LRS(‘aaaaa’) = ‘aaaa’

LRS(‘AAABBB$BBBAAA’) = ‘AAA’, ’BBB’

LRS(‘aabbabbaabab’) = ‘abba’

LRS(‘ZXYZABC’) = ‘Z’

LRS(‘ABCBCACAB’) = ‘AB’, ‘BC’, ‘CA’

LRS

We can solve this problem using a number of methods, in particular:

Brute force: $O(n^3)$

Dynamic Programming: $O(n^2)$

Suffix Trees/Suffix Arrays: $O(n)$

LRS

The Longest Repeated Substring (LRS) is the substring with highest Longest Common Prefix (LCP) count.

Text: 'ABBABABAA'

LCP	Sorted Suffixes
1	A
1	AA
3	ABAA
2	ABABAA
0	ABBABABAA
2	BAA
4	**BABA**A
1	BABABAA
0	BBABABAA

LRS

Text:
'ABCBCACAB'

LCP	Sorted Suffixes
1	BCACAB
0	$CAB
1	A$CAB
2	**AB**
0	ABCBCACAB
1	B
2	**BC**BCACAB
0	BCA$CAB
1	CBCACAB
2	**CA**$CAB
0	CAB

LRS

Text:
'ABCDE'

No LRS!

LCP	Sorted Suffixes
0	ABCDE
0	BCDE
0	CDE
0	DE
0	E

LCP Array Problem

Automatic Trading

Given a sequence of trade transactions as a string where each character represents that a certain stock has been traded find the longest sequence of identical trades starting at positions i and j.

Max string length: 100000, max queries: 100000

```
i               j
A C A A C A B A B C A A D
```

Automatic Trading

Given a sequence of trade transactions as a string where each character represents that a certain stock has been traded find the longest sequence of identical trades starting at positions i and j.

Max string length: 100000, max queries: 100000

i j

A C A A C A B A B C A A D

Longest sequence of identical trades: 3

Automatic Trading

```
  i              j
A C A A C A B A B C A A D
```

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

i j

A C A A C A B A B C A A D

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

i j

A C A A C A B A B C A A D

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

i j

A C A A C A B A B C A A D

The LCP between both suffixes is three, so we conclude that there are three similar trades both starting at i and j

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

```
    i          j
A C A A C A B A B C A A D
```

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

```
    i         j
A C A A C A B A B C A A D
```

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

```
    i         j
A C A A C A B A B C A A D
```

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

```
    i         j
A C A A C A B A B C A A D
```

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

```
    i          j
A C A A C A B A B C A A D
```

The longest common prefix between both suffixes is one, but they're not adjacent in the LCP array, so how do we efficiently determine that their LCP is indeed one?

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Automatic Trading

i j

A C A A C A B A B C A A D

Ans: Query the minimum value in the LCP array between the two suffixes.

You can use a min segment tree or a similar data structure on the LCP array to query the range i to j efficiently

LCP	Suffix
2	AACABABCAAD
1	AAD
2	ABABCAAD
1	ABCAAD
3	ACAACABABCAAD
1	ACABABCAAD
0	AD
1	BABCAAD
0	BCAAD
3	CAACABABCAAD
2	CAAD
0	CABABCAAD
0	D

Text Contains Substring

Contains Substring

Text: ‘abracadabra’

Substring: ‘braca’

Remember Binary Search?

low = 0, high = 9

mid = (low+high)/2 = 4

lex_compare(‘braca’,’bra’) > 0

Update low = mid + 1

Suffix Array

0 abra
1 abracadabra
2 acadabra
3 adabra
4 bra
5 bracadabra
6 cadabra
7 dabra
8 ra
9 racadabra

Contains Substring

Text: ‘abracadabra’

Substring: ‘braca’

Remember Binary Search?

low = 5, high = 9

mid = (low+high)/2 = 7

lex_compare(‘braca’,’dabra’) < 0

Update high = mid – 1

Suffix Array

0 abra
1 abracadabra
2 acadabra
3 adabra
4 bra
5 bracadabra
6 cadabra
7 dabra
8 ra
9 racadabra

Contains Substring

Text: ‘abracadabra’
Substring: ‘bracadabra’

Remember Binary Search?

low = 5, high = 6

mid = (low+high)/2 = 5

lex_compare(‘braca’,’bracadabra’) == 0

Substring exists!

Takes O(mlog(n))

Suffix Array

0 abra
1 abracadabra
2 acadabra
3 adabra
4 bra
5 bracadabra
6 cadabra
7 dabra
8 ra
9 racadabra

Contains Substring

Text: ‘abracadabra’
Substring: ‘zzzz’

Remember Binary Search?

low = 0, high = 9
mid = (low+high)/2 = 4

lex_compare(‘zzzz’,’bra’) > 0

Update low = mid + 1

Suffix Array

0	abra
1	abracadabra
2	acadabra
3	adabra
4	bra
5	bracadabra
6	cadabra
7	dabra
8	ra
9	racadabra

Contains Substring

Text: ‘abracadabra’

Substring: ‘zzzz’

Remember Binary Search?

low = 5, high = 9

mid = (low+high)/2 = 7

lex_compare(‘zzzz’,’dabra’) > 0

Update low = mid + 1

Suffix Array

0 abra
1 abracadabra
2 acadabra
3 adabra
4 bra
5 bracadabra
6 cadabra
7 dabra
8 ra
9 racadabra

Contains Substring

Text: ‘abracadabra’

Substring: ‘zzzz’

Remember Binary Search?

low = 8, high = 9

mid = (low+high)/2 = 8

lex_compare(‘zzzz’,’ra’) > 0

Update low = mid + 1

Suffix Array

0 abra
1 abracadabra
2 acadabra
3 adabra
4 bra
5 bracadabra
6 cadabra
7 dabra
8 ra
9 racadabra

Contains Substring

Text: ‘abracadabra’

Substring: ‘zzzz’

Remember Binary Search?

low = 9, high = 9

mid = (low+high)/2 = 9

lex_compare(‘zzzz’,’racadabra’) > 0

Update low = mid + 1

Suffix Array

0 abra
1 abracadabra
2 acadabra
3 adabra
4 bra
5 bracadabra
6 cadabra
7 dabra
8 ra
9 racadabra

Contains Substring

Text: ‘abracadabra’

Substring: ‘zzzz’

Remember Binary Search?

low = **10**, high = 9

mid = (low+high)/2 = 9

Substring not found

Suffix Array

0	abra
1	abracadabra
2	acadabra
3	adabra
4	bra
5	bracadabra
6	cadabra
7	dabra
8	ra
9	racadabra

Contains Substring

To total time complexity of this algorithm is O(mlog(n)) where m is the length of the substring and n is the length of the text. This beats KMP when m is small.

There exists a variant of this algorithm which runs in O(m+log(n)) which uses information in the LCP array, but this is hardly used in a competitive programming setting.

Longest Common Substring (LCS)

LCS

Suppose we have n strings containing only letters a-z and A-Z. How do we find the **longest common substring** that appears in at least $2 \leq k \leq n$ of the strings?

Note: The LCS may not be unique

Consider n = 3, k = 2 with:

S_1 = ‘abca’

S_2 = ‘bcad’

S_3 = ‘daca’

LCS

One approach is to use dynamic programming running in $O(n_1 * n_2 * n_3 * … * n_m)$, where n_i is the length of the string S_i. This may be ok for a few small strings, but rapidly gets unwieldy.

An alternative method is to use a suffix array which can find the solution in $O(n_1 + n_2 + … + n_m)$ time

LCS

Consider again:

S_1 = abca, S_2 = bcad, S_3 = daca

Form a larger string T which is the concatenation of all the S_i strings separated by **unique sentinels**. The sentinels must be unique and lexicographically less than any of the characters contained in any of the strings S_i.

T = S_1 + ‘#’ + S_2 + ‘$’ + S_3 + ‘%’

= abca#bcad$daca%

LCS

Once we generate the suffix array for T the placement of the sentinel values will allow us to compare the suffixes of T with each other.

We then have to find a sequence of K suffixes coming from the different strings which obtain the highest LCP value. This in turn will tell us the LCS.

T = abca#bcad$daca%

```
0 #bcad$daca%
0 $daca%
0 %
1 a#bcad$daca%
1 a%
1 abca#bcad$daca%
1 aca%
0 ad$daca%
3 bca#bcad$daca%
0 bcad$daca%
2 ca#bcad$daca%
2 ca%
0 cad$daca%
1 d$daca%
0 daca%
```

We can
ignore these

T = abca#bcad$daca%

```
0 #bcad$daca%
0 $daca%
0 %
1 a#bcad$daca%
1 a%
1 abca#bcad$daca%
1 aca%
0 ad$daca%
3 bca#bcad$daca%
0 bcad$daca%
2 ca#bcad$daca%
2 ca%
0 cad$daca%
1 d$daca%
0 daca%
```

T = abca#bcad$daca%

Suppose k = 3 what is the LCS?

Remember that we need one string of each colour and the maximum LCP between them

LCP	Suffixes
1	a#bcad$daca%
1	a%
1	abca#bcad$daca%
1	aca%
0	ad$daca%
3	bca#bcad$daca%
0	bcad$daca%
2	ca#bcad$daca%
2	ca%
0	cad$daca%
1	d$daca%
0	daca%

T = abca#bcad$daca%

Suppose k = 3 what is the LCS?

We can achieve a LCP value of two using these three strings

LCP	Suffixes
1	a#bcad$daca%
1	a%
1	abca#bcad$daca%
1	aca%
0	ad$daca%
3	bca#bcad$daca%
0	bcad$daca%
2	ca#bcad$daca%
2	ca%
0	cad$daca%
1	d$daca%
0	daca%

T = abca#bcad$daca%

Suppose k = 2 what is the LCS?

Remember that we need two different string colours and the maximum LCP between them

LCP	Suffixes
1	a#bcad$daca%
1	a%
1	abca#bcad$daca%
1	aca%
0	ad$daca%
3	bca#bcad$daca%
0	bcad$daca%
2	ca#bcad$daca%
2	ca%
0	cad$daca%
1	d$daca%
0	daca%

T = abca#bcad$daca%

There is a unique solution for k = 2 which is 'bca' with a length of 3

LCP	Suffixes
1	a#bcad$daca%
1	a%
1	abca#bcad$daca%
1	aca%
0	ad$daca%
3	bca#bcad$daca%
0	bcad$daca%
2	ca#bcad$daca%
2	ca%
0	cad$daca%
1	d$daca%
0	daca%

Things can get more messy when suffixes of different colours are not exactly adjacent.

```
K = 3  T = bbabbebbf#bbc#bbg
…
1 babbebbe#bbc#bbg
2 bbabbebbe#bbc#bbg
2 bbc#bbg
3 bbe#bbc#bbg
2 bbebbe#bbc#bbg
1 bbg
1 bc#bbg
…
```

LCS Algorithm

Use a **sliding window** to capture the correct amount of suffix colours. At each step advance the left endpoint and adjust the right endpoint such that the window contains exactly k suffixes of different colours.

For each valid window perform a range query on the LCP array between the left and right endpoint indexes to determine the LCS between all the strings and update accordingly. Luckily for us the **sliding range query problem** can be solved in O(n) time! Alternatively, you can use quick range query DS to perform queries in log(n) time which may be easier.

LCS Algorithm

You will also want a DS to keep track of how many suffixes of each colour are currently in the window you’re considering to know if you need to ‘grow’ or ‘shrink’ the interval.

Additionally, you will also want to track the length of the current LCS to update the LCS set accordingly.

LCS Example 1

Consider three strings S_1, S_2, S_3. Find the LCS that appears in at least three of these strings (K = 3)

S_1 = AAGAAGC, S_2 = AGAAGT, S_3 = CGAAGC

T = AAGAAGC#AGAAGT$CGAAGC%

LCS(S_1,S_2,S_3) = { GAAG }

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 0
LCS = { }

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 0
LCS = { }

$\boxed{3}$	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 0
LCS = { }

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 0
LCS = { }

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 3
Window LCS = AAG

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 3
Window LCS = AAG

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 1
Window LCS = A

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 1
Window LCS = A

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

```
3 AAGAAGC#AGAAGT$CGAAGC%
4 AAGC#AGAAGT$CGAAGC%
3 AAGC%
1 AAGT$CGAAGC%
5 AGAAGC#AGAAGT$CGAAGC%
2 AGAAGT$CGAAGC%
3 AGC#AGAAGT$CGAAGC%
2 AGC%
0 AGT$CGAAGC%
1 C#AGAAGT$CGAAGC%
1 C%
0 CGAAGC%
5 GAAGC#AGAAGT$CGAAGC%
4 GAAGC%
1 GAAGT$CGAAGC%
2 GC#AGAAGT$CGAAGC%
1 GC%
0 GT$CGAAGC%
0 T$CGAAGC%
```

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 2
Window LCS = AG

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 2
Window LCS = AG

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 2
Window LCS = AG

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { AAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 4
Window LCS = GAAG

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 1
Window LCS = G

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 1
Window LCS = G

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	`AAGAAGC#AGAAGT$CGAAGC%`
4	`AAGC#AGAAGT$CGAAGC%`
3	`AAGC%`
1	`AAGT$CGAAGC%`
5	`AGAAGC#AGAAGT$CGAAGC%`
2	`AGAAGT$CGAAGC%`
3	`AGC#AGAAGT$CGAAGC%`
2	`AGC%`
0	`AGT$CGAAGC%`
1	`C#AGAAGT$CGAAGC%`
1	`C%`
0	`CGAAGC%`
5	`GAAGC#AGAAGT$CGAAGC%`
4	`GAAGC%`
1	`GAAGT$CGAAGC%`
2	`GC#AGAAGT$CGAAGC%`
1	`GC%`
0	`GT$CGAAGC%`
0	`T$CGAAGC%`

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = 1
Window LCS = G

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don’t excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 4
LCS = { GAAG }

Do a range query on LCP array but don't excluding the right endpoint in our case because our LCP array finds the LCP between suffix i and i+1

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

S_1 = AAGAAGC
S_2 = AGAAGT
S_3 = CGAAGC
with K = 3

Window LCP = NA
Window LCS = NA

LCS length = 4
LCS = { **GAAG** }

3	AAGAAGC#AGAAGT$CGAAGC%
4	AAGC#AGAAGT$CGAAGC%
3	AAGC%
1	AAGT$CGAAGC%
5	AGAAGC#AGAAGT$CGAAGC%
2	AGAAGT$CGAAGC%
3	AGC#AGAAGT$CGAAGC%
2	AGC%
0	AGT$CGAAGC%
1	C#AGAAGT$CGAAGC%
1	C%
0	CGAAGC%
5	GAAGC#AGAAGT$CGAAGC%
4	GAAGC%
1	GAAGT$CGAAGC%
2	GC#AGAAGT$CGAAGC%
1	GC%
0	GT$CGAAGC%
0	T$CGAAGC%

LCS Example 2

Consider four strings S_1, S_2, S_3, S_4. Find the LCS that appears in at least two of the strings (K = 2)

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

T = AABC#BCDC$BCDE%CDED&

LCS(S_1,S_2,S_3,S_4) = { BCD, CDE }

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Before we find the LCS for K = 2 observe that if all the sentinels had the same value we would have a problem here

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 0
LCS = { }

$\boxed{1}$	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 0
LCS = { }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 0
LCS = { }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 0
Window LCS = “”

LCS length = 0
LCS = { }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 0
Window LCS = “”

LCS length = 0
LCS = { }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 2
Window LCS = BC

LCS length = 2
LCS = { BC }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 2
LCS = { BC }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
$\boxed{3}$	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 3
Window LCS = BCD

LCS length = 3
LCS = { BCD }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
$\boxed{0}$	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { BCD }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
$\boxed{1}$	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 1
Window LCS = C

LCS length = 3
LCS = { BCD }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD }

1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 1
Window LCS = C

LCS length = 3
LCS = { BCD }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 2
Window LCS = CD

LCS length = 3
LCS = { BCD }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
$\boxed{3}$	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 3
Window LCS = CDE

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
$\boxed{0}$	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD, CDE }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 1
Window LCS = D

LCS length = 3
LCS = { BCD, CDE }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
$\boxed{1}$	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 1
Window LCS = D

LCS length = 3
LCS = { BCD, CDE }

1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
$\boxed{2}$	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 2
Window LCS = DE

LCS length = 3
LCS = { BCD, CDE }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
$\boxed{0}$	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 0
Window LCS = “”

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
0	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = 1
Window LCS = E

LCS length = 3
LCS = { BCD, CDE }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { BCD, CDE }

1	AABC#BCDC$BCDE%CDED&
0	ABC#BCDC$BCDE%CDED&
2	BC#BCDC$BCDE%CDED&
3	BCDC$BCDE%CDED&
0	BCDE%CDED&
1	C#BCDC$BCDE%CDED&
1	C$BCDE%CDED&
2	CDC$BCDE%CDED&
3	CDE%CDED&
0	CDED&
1	D&
1	DC$BCDE%CDED&
2	DE%CDED&
0	DED&
1	E%CDED&
$\boxed{0}$	ED&

S_1 = AABC, S_2 = BCDC
S_3 = BCDE, S_4 = CDED

Window LCP = NA
Window LCS = NA

LCS length = 3
LCS = { **BCD**, **CDE** }

```
1 AABC#BCDC$BCDE%CDED&
0 ABC#BCDC$BCDE%CDED&
2 BC#BCDC$BCDE%CDED&
3 BCDC$BCDE%CDED&
0 BCDE%CDED&
1 C#BCDC$BCDE%CDED&
1 C$BCDE%CDED&
2 CDC$BCDE%CDED&
3 CDE%CDED&
0 CDED&
1 D&
1 DC$BCDE%CDED&
2 DE%CDED&
0 DED&
1 E%CDED&
0 ED&
```

Suggested Problems

LCP array related

Aliens
Automatic trading
Substrings

Suffix array related

Burrows wheeler
Suffix array reconstruction
Suffix sorting

Suggested Problems

Touching topics covered

Dvaput (LRS)
Clock pictures (KMP)
Lifeforms (LCS)

Other string problems

Permagrams
Messages
Bing it on
Power strings
Chasing subs

References

Competitive Programming 3 The New Lower Bound of Programming Contests by Steven and Felix Halim

KMP reference: http://jakeboxer.com/blog/2009/12/13/the-knuth-morris-pratt-algorithm-in-my-own-words/

LCS reference: Presentation by Maxim A. Babenko, Tatiana A. Starikovskaya. http://lpcs.math.msu.su/~pritykin/csr2008presentations/starikovskaya.pdf

http://wcipeg.com/wiki/Longest_common_substring