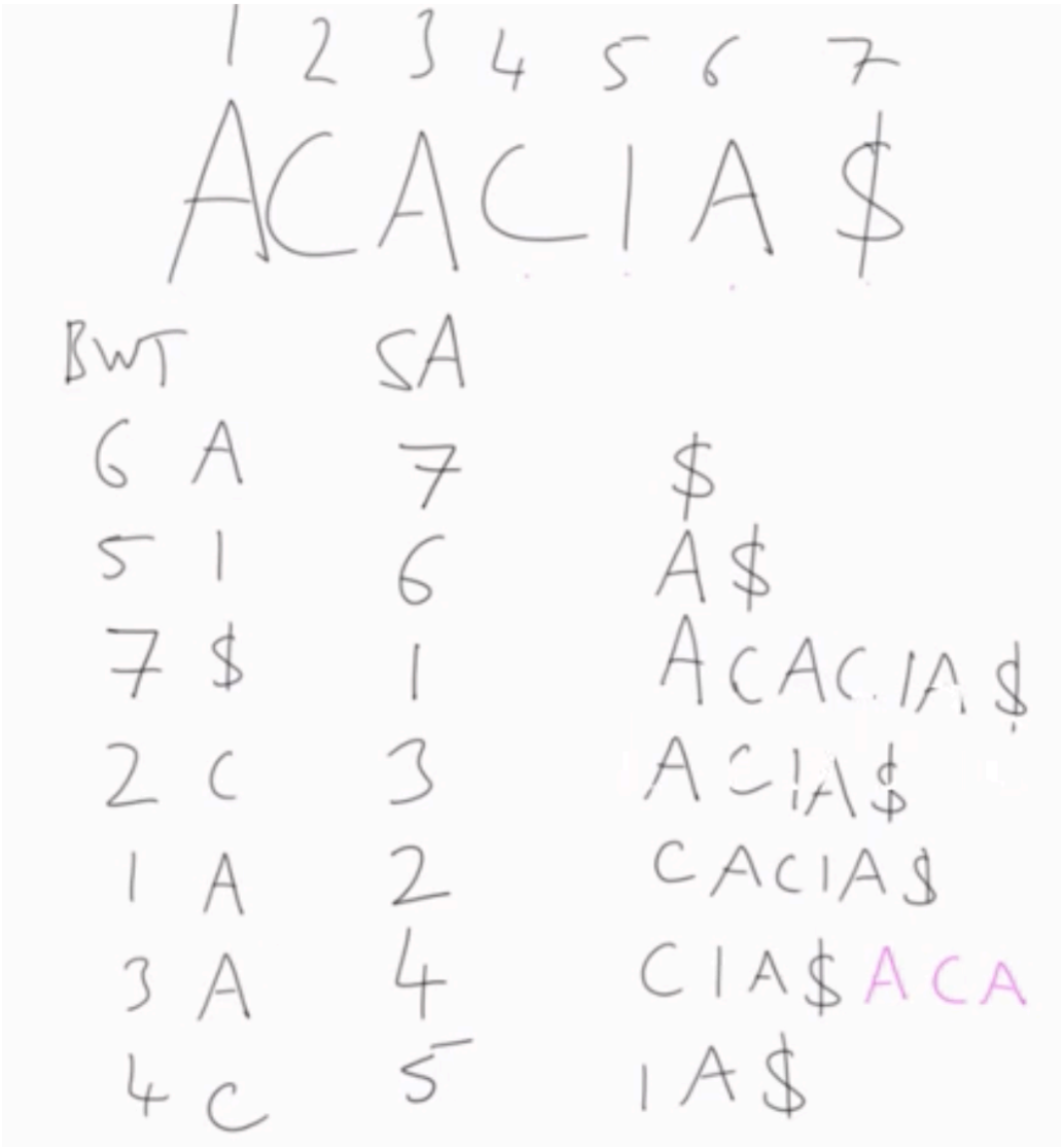
FIT2004 Week 8 Tutorial

Problem 1: Get the SUFFIX ARRAY (SA) & Burrows - Wheeler Transform



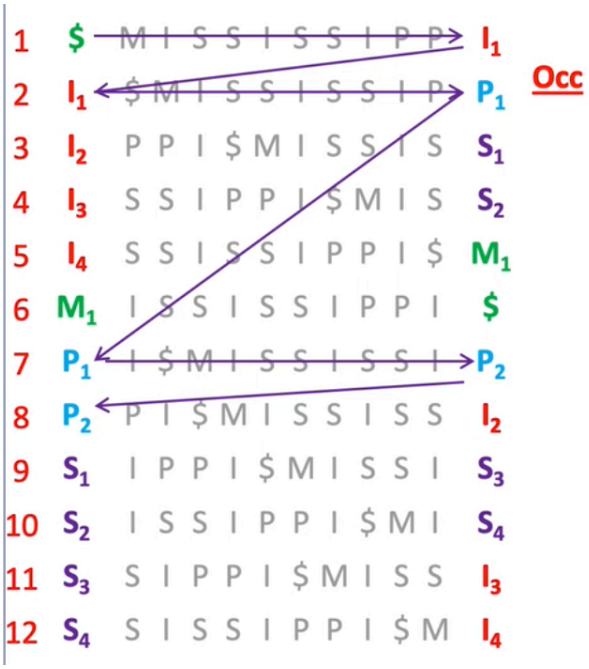
Problem 2: BWT Reversion

Naive Approach: sort the last column --- > append the last column before the sorted one ---> sorted two column together ---> keep doing so

Time Complexity: O(N^3)
--- O(N) sort calls, max Sort is the last one with radix sort O(N^2), N rows, each N chars

Space Complexity: O(N^2) Matrix

Better Approach: O(N) Time and O(N) Space



Occ

1	0
2	0
3	0
4	1
5	0
6	0
7	1
8	1
9	2
10	3
11	2
12	3

1	2	6	7	9
$	I	M	P	S

Rank

1	4	1	2	4
$	I	M	P	S

Count

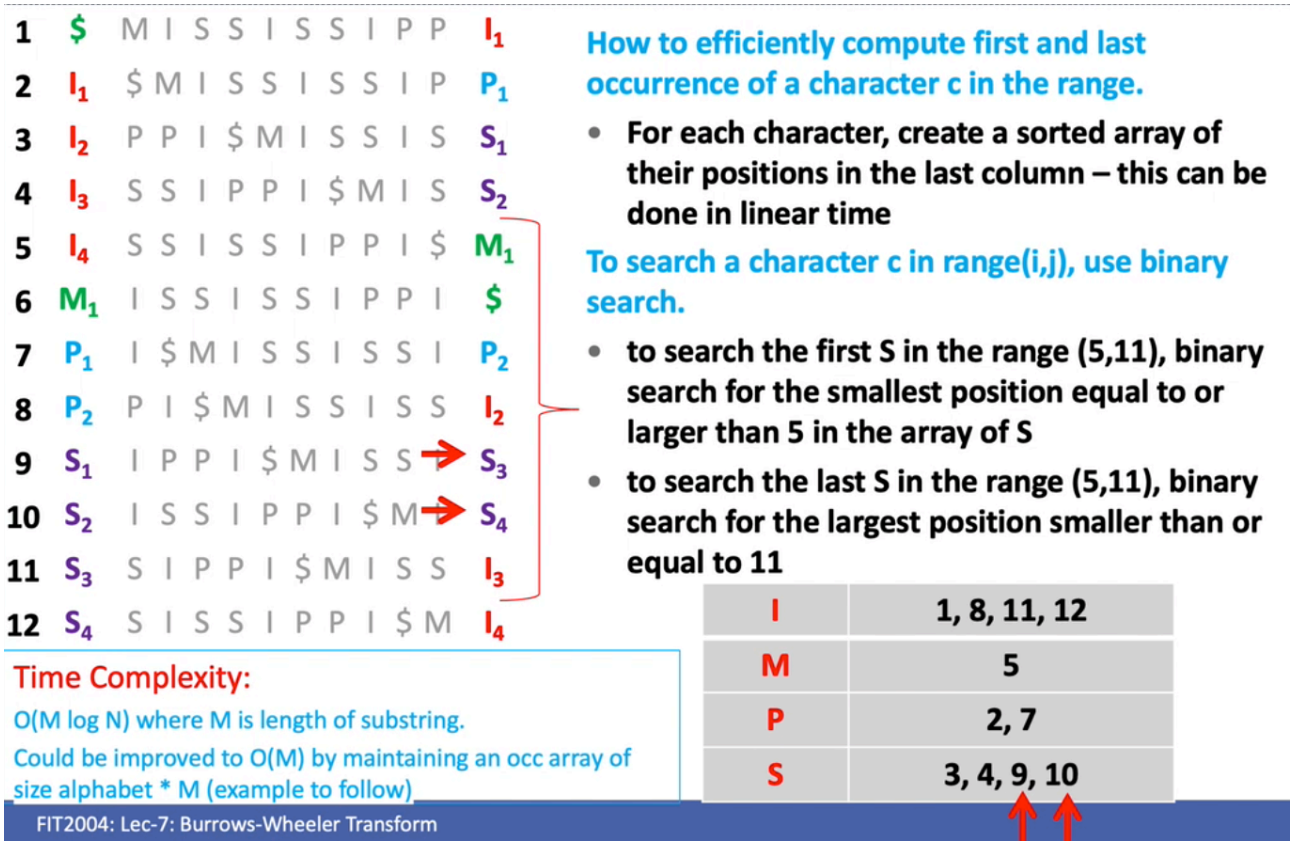
Rank_M is 6: because Rank_I is 2 and there are 4 Is.

End of Row_2 ---> P ---> Occ is 0 ---> 1st P ---> Occ_0 + Rank_P_7 ---> Row_7

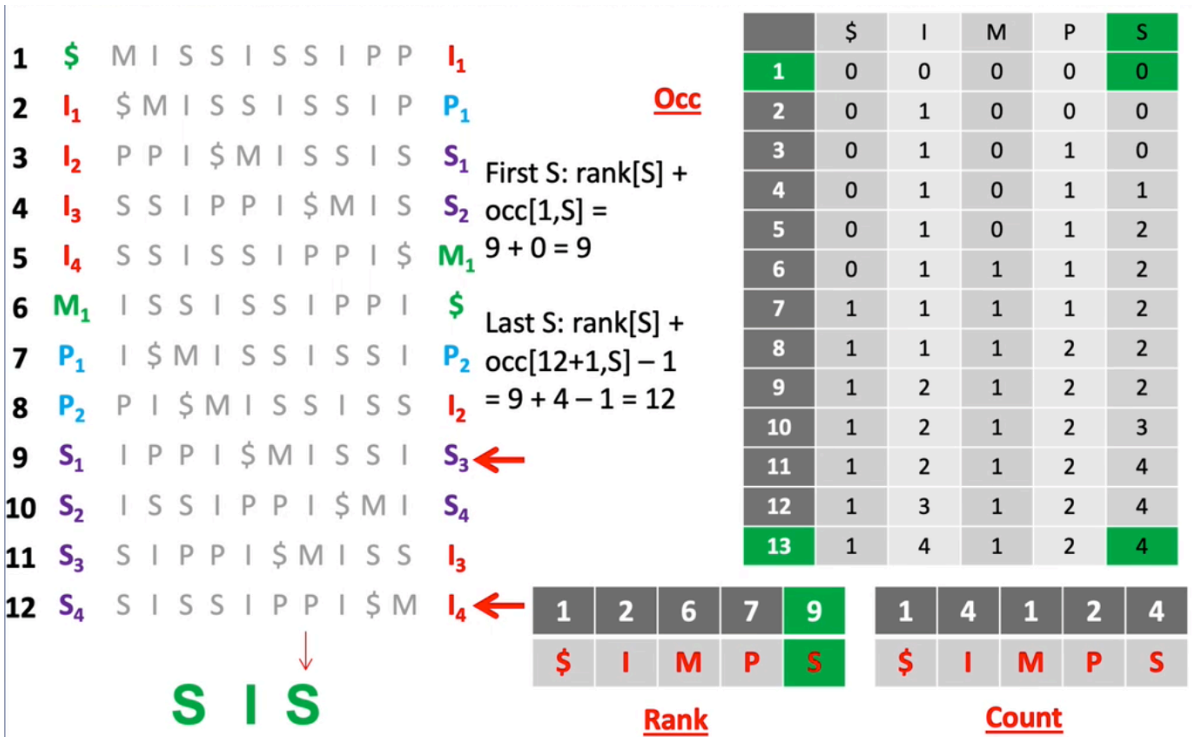
End of Row_7 ---> P ---> Occ is 1 ---> 2nd P ---> Occ_1 + Rank_P_7 ---> Row_8

Substring Search in BWT

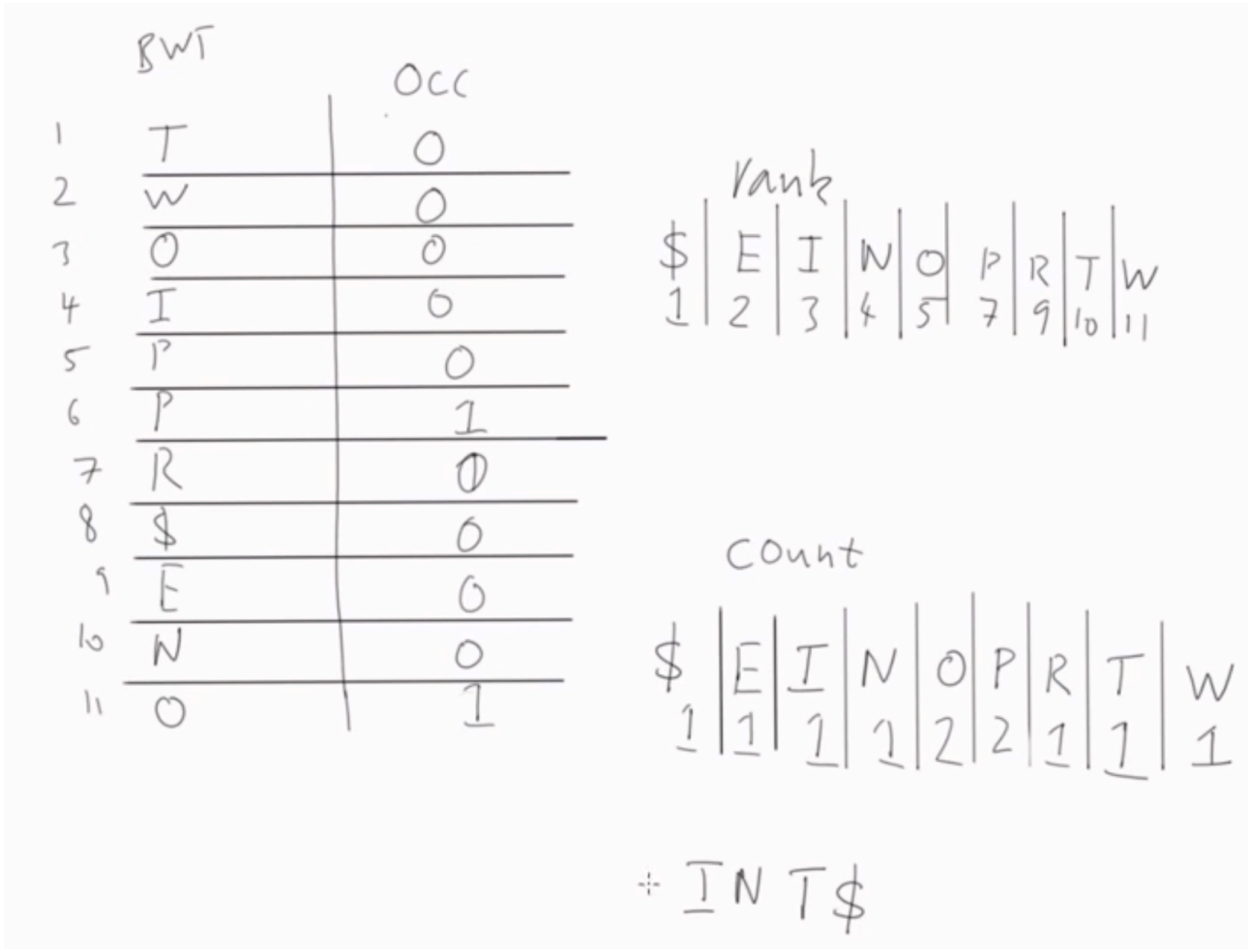
Naive Way: O(M*log(N))



Better: Time in O(M)



Faster Reversion of BWT



Problem 7: Minimum Lexicographical Rotation: Add Corresponding Prefix On

Problem 7. The *minimum lexicographical rotation* problem is the problem of finding, for a given string, its cyclic rotation that is lexicographically least. For example, given the string `banana`, its cyclic rotations are `banana`, `ananab`, `nanaba`, `anaban`, `nabana`, `abanan`. The lexicographically (alphabetically) least one is `abanan`. Describe how to solve the minimum lexicographical rotation problem using a suffix array.

Solution

A cyclic rotation of a string is a suffix of that string followed by a prefix of that string. Since a suffix array stores the suffixes in sorted order, the lexicographically least suffix will be the first element of the arrray (ignoring the "$" which will always be first.) However, we can run into problems such as the string `XAA$`:

i	SA[i]	Suffix
0	3	$
1	2	a$
2	1	aa$
3	0	xaa$

Source: `https://visualgo.net/bn/suffixarray`

Here, the suffix "`A$`" would be followed by `XA` in its cyclic rotation, giving `AXA`, wheras the suffix `AA$` would be followed by `X`, giving `AAX`, which is lexicographically earlier than `AXA`. So it is not enough to just construct the suffix array, we need to account for the elements of the corresponding prefix.

We do this by first appending a copy of the string in question to itself, and then constructing the suffix array. This gaurantees that the corresponding prefix for each suffix is taken into account in the order of the suffix array. Note that we should only consider suffixes which are strictly longer than the original string, since only these suffixes contain characters from the appended copy.

i	SA[i]	Suffix
0	6	$
1	5	a$
2	4	aa$
3	1	aaxaa$
4	2	axaa$
5	3	xaa$
6	0	xaaxaa$

Source: `https://visualgo.net/bn/suffixarray`

Now `AAXAA$X` is the first such string in the suffix array, and hence `AAX` is the lexicographically least rotation.

Problem 9: Another Example of Prefix Doubling

Problem 8. Consider the following rank array, obtained during the prefix doubling algorithm applied to the strings `ABBAABABBA$`. At this point in the algorithm, we have just finished updating the ranks for some value of k.

String	A	B	B	A	A	B	A	B	B	A	$
Suffix ID	1	2	3	4	5	6	7	8	9	10	11
Rank	4	6	5	3	4	5	4	6	5	2	1

(a) Determine what the current value of k is. In other words, what lengths of strings are the ranks currently comparing?

(b) Describe in detail how the prefix doubling algorithm would compare the following pairs of suffixes during the next iteration (for strings of length $2k$):

- 1 and 2
- 1 and 5
- 2 and 8
- 3 and 9
- 1 and 7

Solution

(a) k cannot be 1, since suffix 1 and 4 are different. k could be 2, since all identical 2 letter strings have the same rank. k cannot be 4, since ID 2 and ID 8 have the same rank, but are different 4 character strings. So k is 2.

(b)

- Rank[1] = 4. Rank[2] = 6. Since Rank[1] < Rank[2], we can determine that suffix 1 should come before suffix 2 during sorting.
- Rank[1] = 4. Rank[5]=4. Since they are equal, we need to compare the following 2 characters. Rank[1+2] = 5. Rank[5+2] = 4. Since Rank[5+2] < Rank[1+2], we can determine that suffix 5 is less than suffix 1.
- Rank[2] = 6. Rank[8] = 6. Since they are equal, we need to compare the following 2 characters. Rank[2+2] = 3. Rank[8+2] = 2. Since Rank[8+2] < Rank[2+2], we can determine that suffix 8 is less than 2.
- Rank[3] = 5. Rank[9] = 5. Since they are equal, we need to compare the following 2 characters. Rank[3+2] = 4. Rank[9+2] = 1. Since Rank[9+2] < Rank[3+2], we can determine that suffix 9 is less than suffix 3.
- Rank[1] = 4. Rank[7] = 4. Since they are equal, we need to compare the following 2 characters. Rank[1+2] = 5. Rank[7+2] = 5. Since they are also equal, we cannot differentiate suffixes 1 and 7 on their first 4 characters. As the sort needs to be stable, we will not change their relative order.