Boyer-Moore and Knuth-Morris-Pratt String Search Algorithms

By: Cameron Brickett, Jay Gomes, Ishan Chadha

Ways to use String Search Algorithms

- Search Engines
- Finding a word within a text document
- Plagiarism finder

Boyer-Moore

- Created by Robert Boyer and J Strother Moore in 1977
- Is the benchmark for String search algorithms

Boyer-Moore String Search

- This string search algorithm is considered to be the benchmark for all string searches, but takes a different approach.

- It searches the string from Right to Left

Best-Case

O(N/M)

Worst-Case

O(N*M)

O(N+M)

Boyer-Moore

- The algorithm will decided how many spaces it has to move from right to left because of these two rules.

- **Good Suffix Rule**
- **Bad Character Rule**

Good Suffix Rule

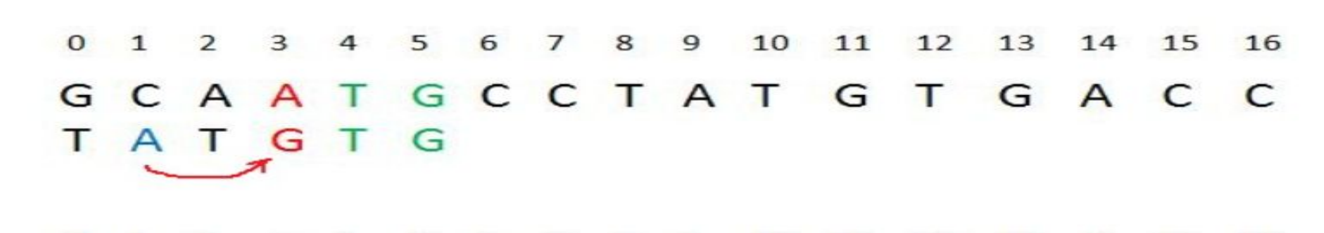
This rule will scan the string you give the program to see if the postfix matches with the prefix of the string.

If a mismatch is found within the text it will

1. Check to see if there is any matches in the suffix
2. If it does find a match it will check to see if the suffix matches the first letter. It will also check to see if the suffix appears somewhere else in the pattern

Bad Characteristic

The text and the pattern we are searching for are lined up both starting at 0 and look for matches on the right of the pattern.



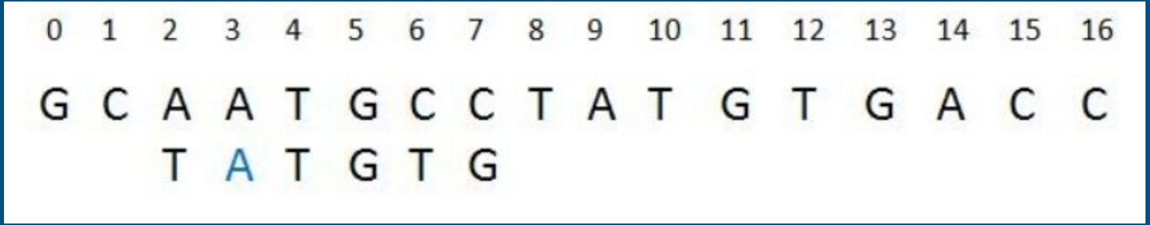
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C
T	A	T	G	T	G											

It will then look from the right to the left trying to find matches.

Here it finds a mismatch at 3. It take the letter at position 3 and scan the pattern to see if there is a match.

Bad Characteristic Continued

In this example it finds a match at position 1. It will take the mismatch number and subtract the matching position to get the amount of shift needed.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G	C	A	A	T	G	C	C	T	A	T	G	T	G	A	C	C
		T	A	T	G	T	G									

If there isn’t a match found then it will move the entire string past position 5

It will continue this process until the match is found.

Combination

To find the amount to shift over by we must combined the max of the Bad Characteristic and Good Suffix rules.

Knuth-Morris-Pratt (KMP) Algorithm

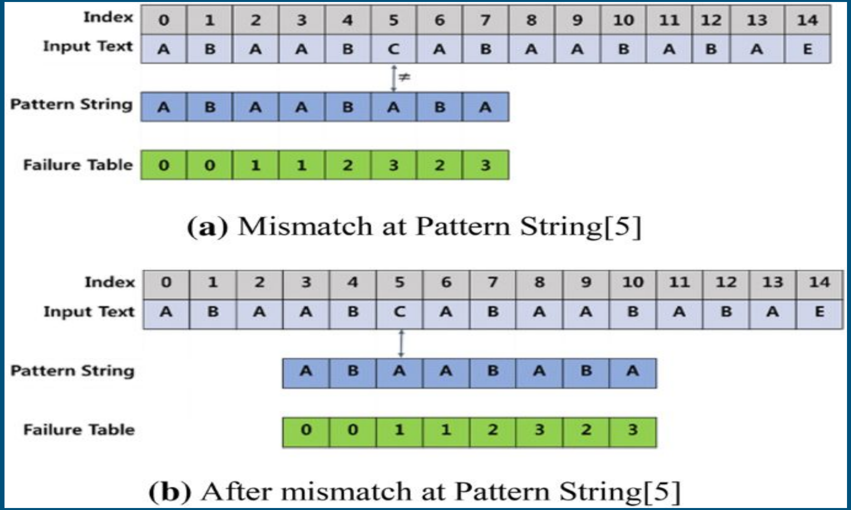
- Discovered by Donald Knuth, Vaughan Pratt, and James H. Morris
  - Published by the three in 1977
- Created by analyzing the Naive string search algorithm

Knuth-Morris-Pratt (KMP) Algorithm

- Time Complexity
  - Always O(N+M)
    - N: Length of Pattern
    - M: Length of Source String
- Similar to Naive Algorithm, but retains more information

KMP Algorithm: How It Works

- Algorithm loops through string, looking for a match with the pattern
  - This is just like the Naive string search algorithm
- Creates a failure table, and uses that to “skip” rematching



Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Input Text	A	B	A	A	B	C	A	B	A	A	B	A	B	A	E
						≠									
Pattern String	A	B	A	A	B	A	B	A							
Failure Table	0	0	1	1	2	3	2	3							

(a) Mismatch at Pattern String[5]

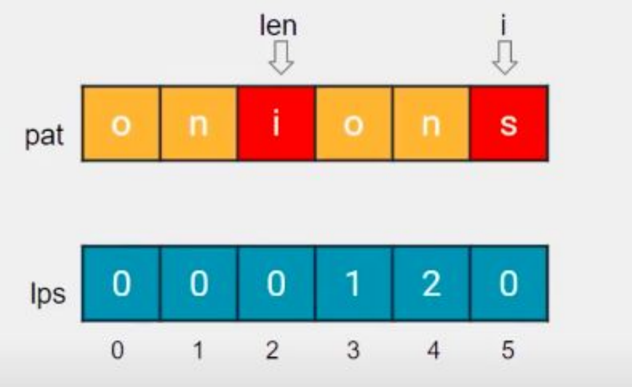
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Input Text	A	B	A	A	B	C	A	B	A	A	B	A	B	A	E
Pattern String				A	B	A	A	B	A	B	A				
Failure Table				0	0	1	1	2	3	2	3				

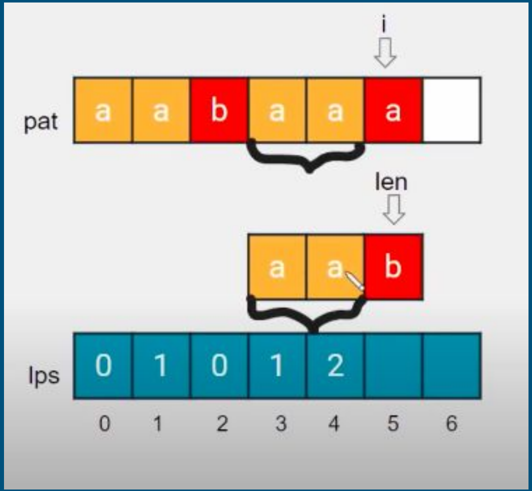
(b) After mismatch at Pattern String[5]

Solve lps array:

- Starting from the first two positions if there are a match increment both pointers.
- If not a match set to 0

- In the case that len is not 0 reference the previous longest prefix that is also a suffix to advance.



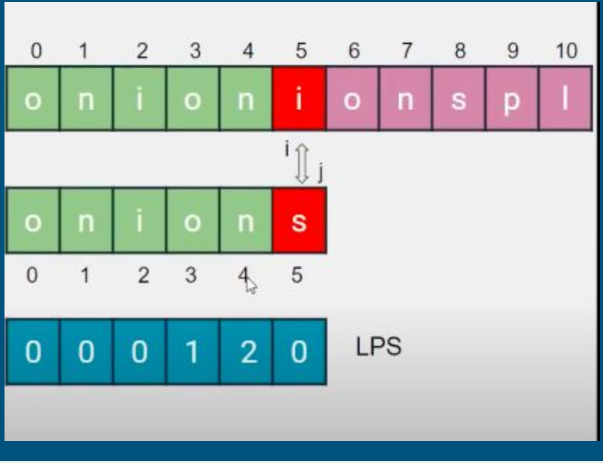


KMP search:

- When a match is made both pointers will advance
- When there is a mismatch the lps array will be reference to find the next position for the pattern pointer

- In the case a mismatch occurs when j is equal to 0 we will advance the original array only

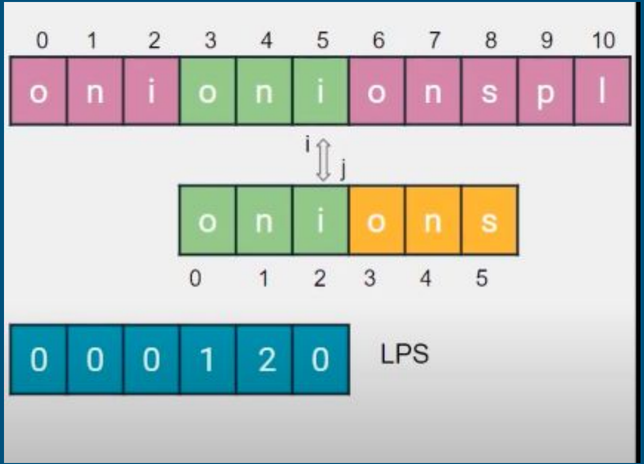
- If j is equal to the size of the pattern, this means we have made a match



0	1	2	3	4	5	6	7	8	9	10
o	n	i	o	n	i	o	n	s	p	l

o	n	i	o	n	s
0	1	2	3	4	5

0	0	0	1	2	0	LPS



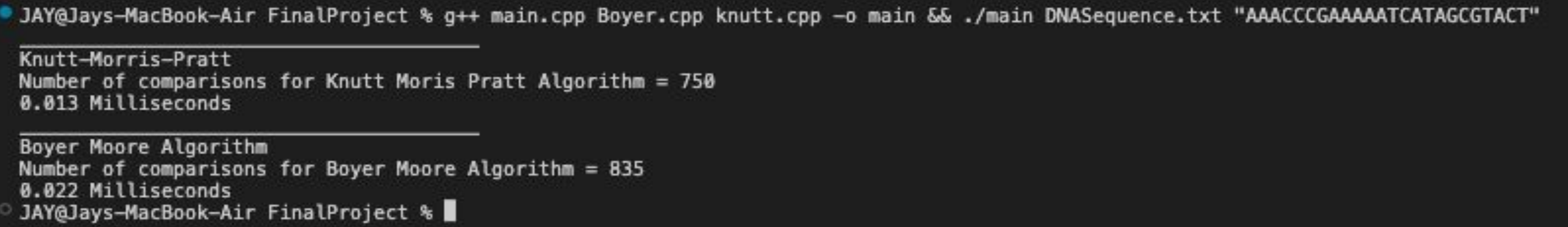
0	1	2	3	4	5	6	7	8	9	10
o	n	i	o	n	i	o	n	s	p	l

o	n	i	o	n	s
0	1	2	3	4	5

0	0	0	1	2	0	LPS

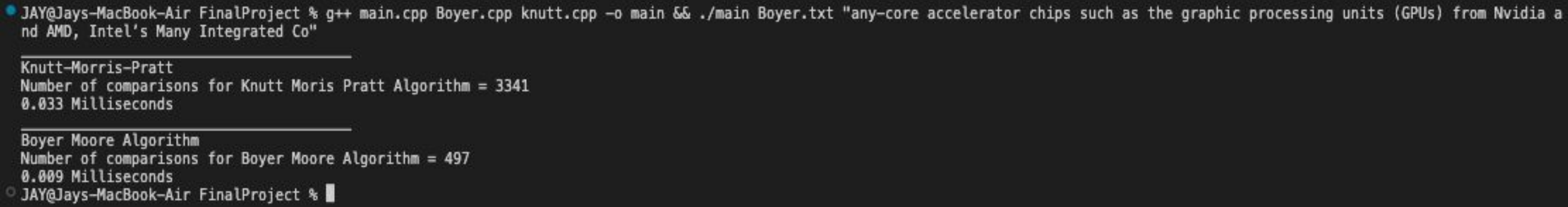
Findings

- The performance will depend on the type of search we want to perform.
    - KMP will be more efficient when the data is a small alphabet (Gene sequence)
        - This will allow more chance of a substring that can be skipped
    - BMH is more efficient when the pattern and data is long.
        - With each unsuccessful attempt to find a match the algorithm uses the bad match table to rule out positions where the pattern cannot match



```
JAY@Jays-MacBook-Air FinalProject % g++ main.cpp Boyer.cpp knutt.cpp -o main && ./main DNASequence.txt "AAACCCGAAAAATCATAGCGTACT"
_______________________________________
Knutt-Morris-Pratt
Number of comparisons for Knutt Moris Pratt Algorithm = 750
0.013 Milliseconds

_______________________________________
Boyer Moore Algorithm
Number of comparisons for Boyer Moore Algorithm = 835
0.022 Milliseconds
JAY@Jays-MacBook-Air FinalProject % 
```



```
JAY@Jays-MacBook-Air FinalProject % g++ main.cpp Boyer.cpp knutt.cpp -o main && ./main Boyer.txt "any-core accelerator chips such as the graphic processing units (GPUs) from Nvidia a
nd AMD, Intel's Many Integrated Co"
_______________________________________
Knutt-Morris-Pratt
Number of comparisons for Knutt Moris Pratt Algorithm = 3341
0.033 Milliseconds

_______________________________________
Boyer Moore Algorithm
Number of comparisons for Boyer Moore Algorithm = 497
0.009 Milliseconds
JAY@Jays-MacBook-Air FinalProject % 
```