String Search II

Boyer-Moore and Knuth-Morris-Pratt

Nick, Aidan, Bill, and Emily

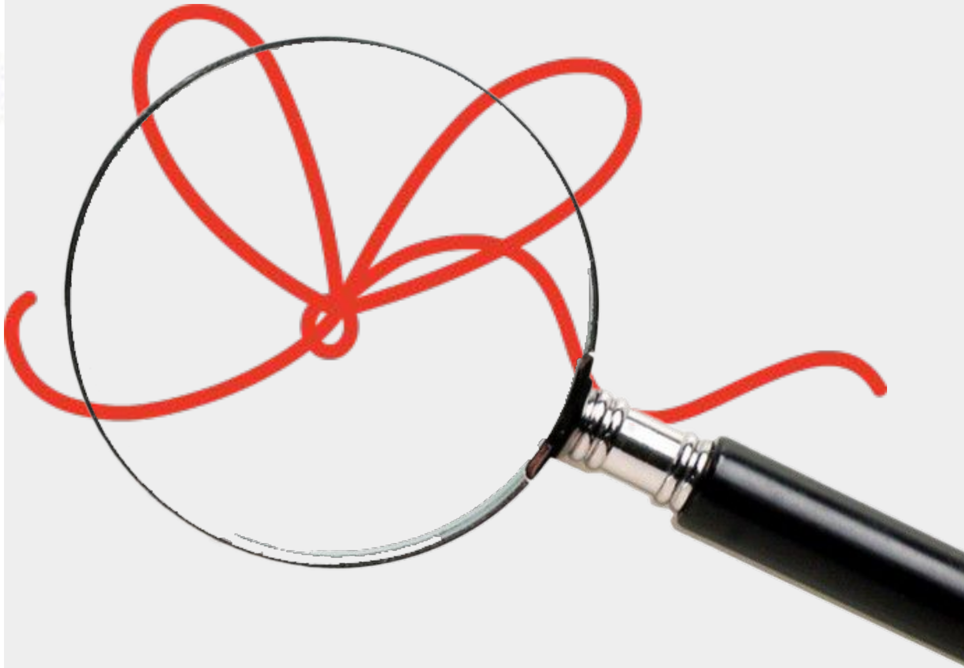
String Search

> Introduction

> Brute Force/Naive

> Boyer-Moore

> Knuth-Morris-Pratt

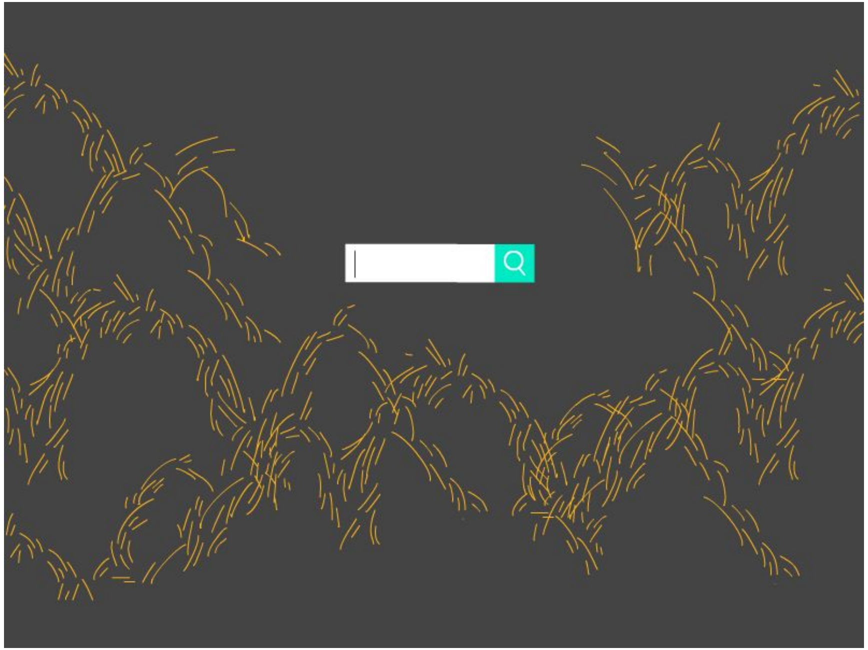


String Search

Goal: Find pattern of length *M* in a text of length *N*.

Pattern: science

Text: Computer science class



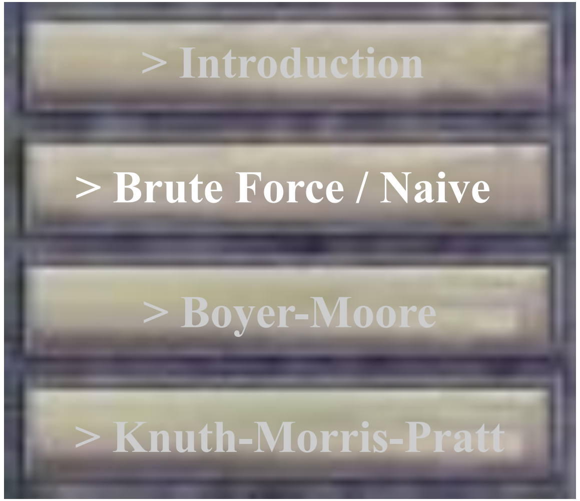
String Search Applications

- Plagiarism detection
- DNA sequencing/bioinformatics
- Spam filter
- Search engines

Bing

- As seen on
- Buy direct
- Meet singles
- Near you
- Additional income
- Kromer

Brute Force Method/Naive



> Introduction

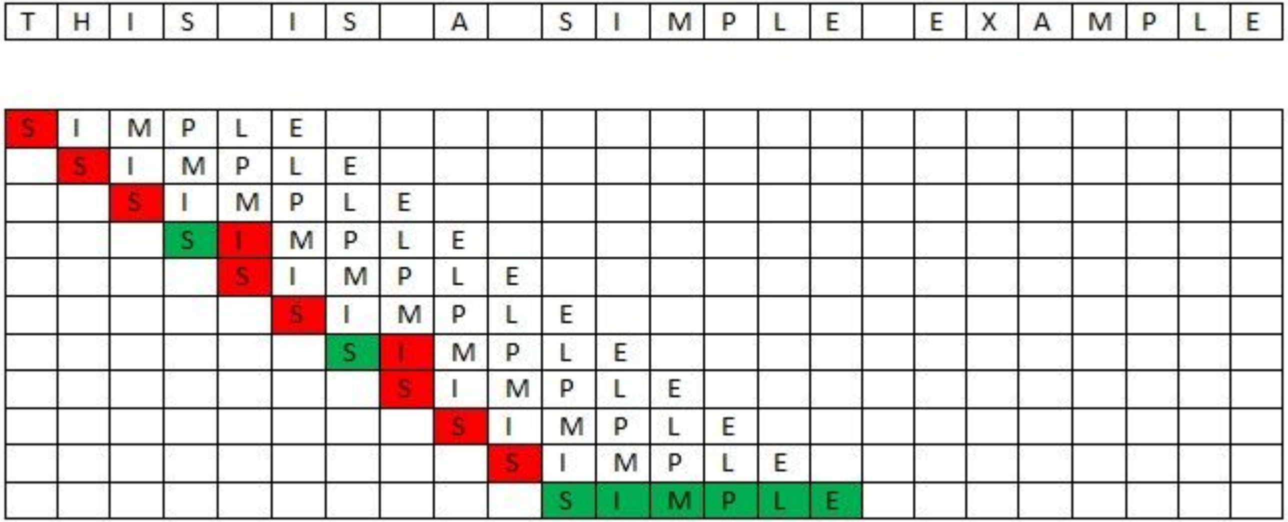
> Brute Force / Naive

> Boyer-Moore

> Knuth-Morris-Pratt

Brute Force String Search

```
do
  if (text letter == pattern letter)
     compare next letter of pattern to next
     letter of text
  else
     move pattern down text by one letter
while (entire pattern found or end of text)
```

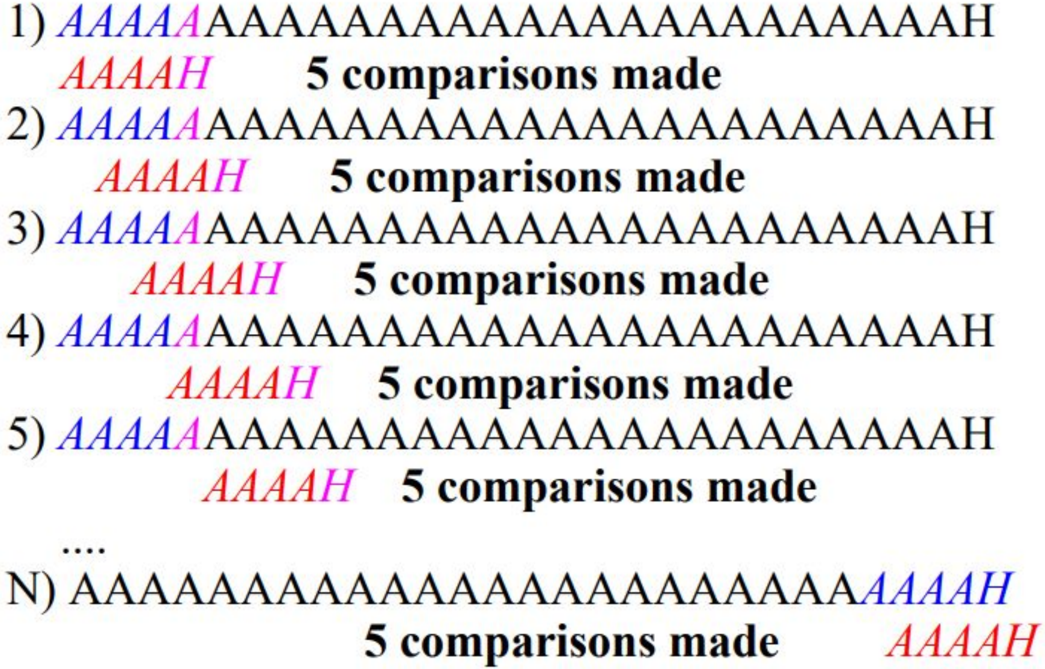


T	H	I	S		I	S		A		S	I	M	P	L	E		E	X	A	M	P	L	E
S	I	M	P	L	E																		
	S	I	M	P	L	E																	
		S	I	M	P	L	E																
			S	I	M	P	L	E															
				S	I	M	P	L	E														
					S	I	M	P	L	E													
						S	I	M	P	L	E												
							S	I	M	P	L	E											
								S	I	M	P	L	E										
									S	I	M	P	L	E									
										S	I	M	P	L	E								

Brute Force String Search: Worst Case

> Too much repetitive text, causes the algorithm to compare M (the length of the input string) times at each index of N

- Total number of comparisons: M (N-M+1)
- Worst case time complexity: O(MN)

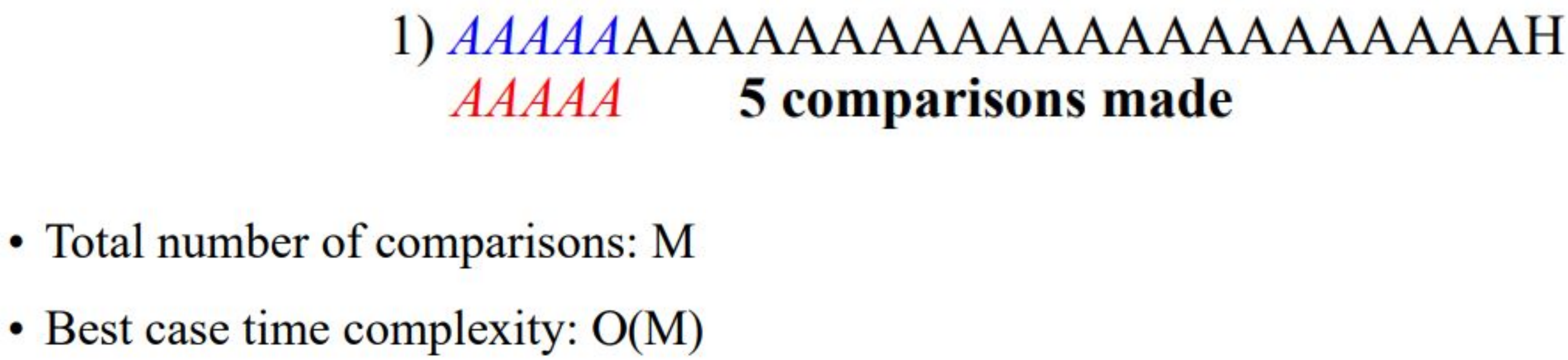


1) *AAAAA*AAAAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**
2) *AAAAA*AAAAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**
3) *AAAAA*AAAAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**
4) *AAAAA*AAAAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**
5) *AAAAA*AAAAAAAAAAAAAAAAAAAAAAAH
AAAAH **5 comparisons made**
....
N) AAAAAAAAAAAAAAAAAAAAAAAA*AAAAH*
5 comparisons made *AAAAH*

So much backup!

Brute Force String Search: Best Case

> Find pattern in first position



1) *AAAAA*AAAAAAAAAAAAAAAAAAAAAAAAH

AAAAA **5 comparisons made**

- Total number of comparisons: M
- Best case time complexity: O(M)

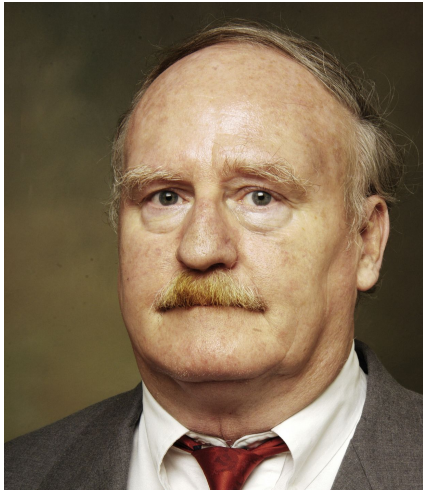
Boyer-Moore

> Introduction

> Brute Force/Naive

> Boyer-Moore

> Knuth-Morris-Pratt



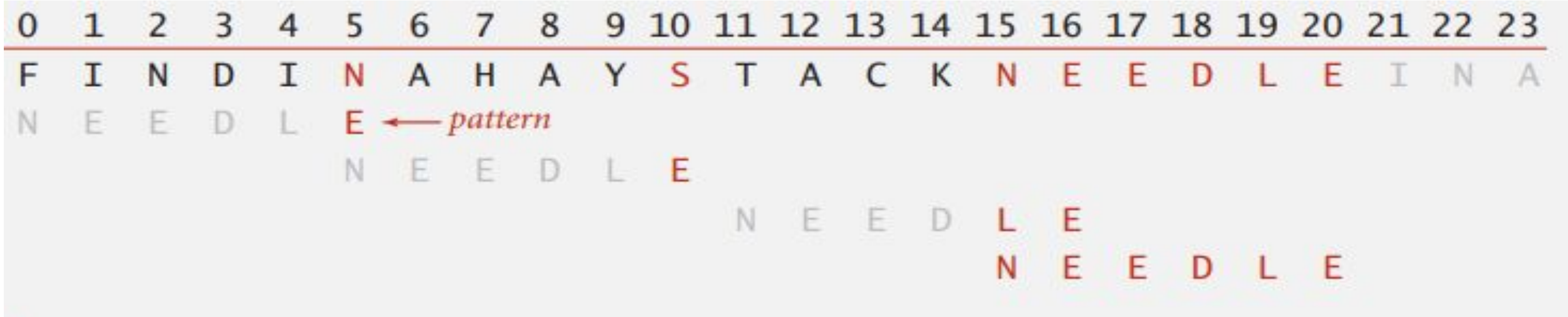
Robert S Boyer

J Strother Moore

Boyer-Moore String Search

> Scans characters in pattern from right to left

> Can skip as many as *M* text characters when not finding one in the pattern

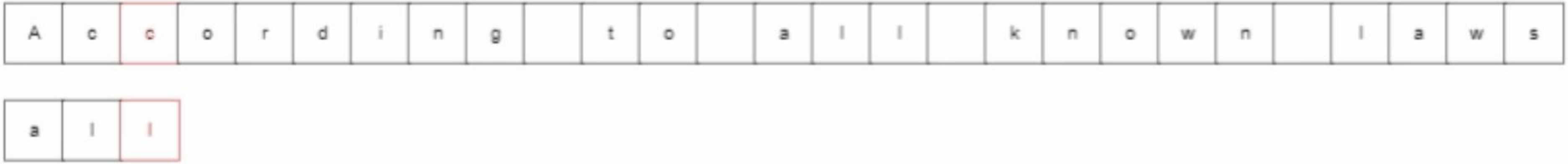


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
F	I	N	D	I	N	A	H	A	Y	S	T	A	C	K	N	E	E	D	L	E	I	N	A
N	E	E	D	L	E																		
					N	E	E	D	L	E													
											N	E	E	D	L	E							
															N	E	E	D	L	E			

Boyer-Moore String Search

> Runs patterns in length *N* until the final char is found, and then will progressively move backwards

> If it runs into an error, it will move forward again based on *N*



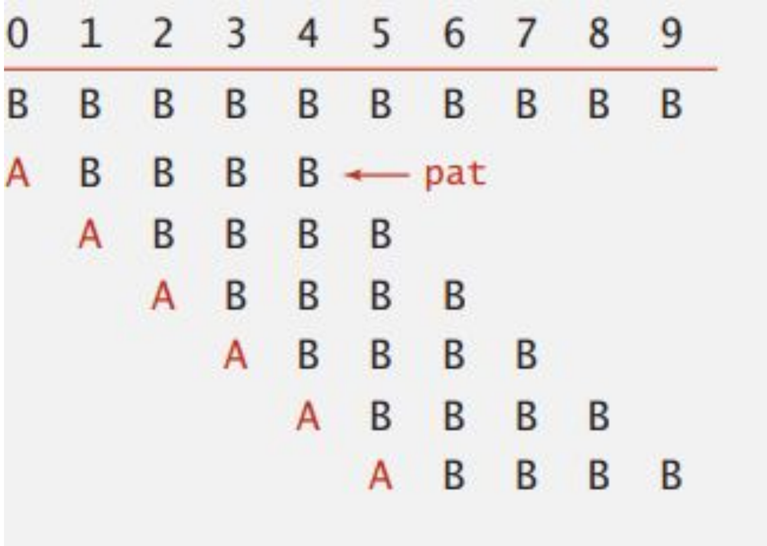
Boyer-Moore String Search: Pseudocode

```
While the end of the text has not been reached
    While the characters in the pattern and text match
        compare the characters in the pattern and text at a certain alignment from right to left
    If a match has been found
        Save the index as being found
        Move to next alignment
    else if a mismatch has been found
    Check to see if the mismatched character in the text exists elsewhere in the in the pattern
    Check to see if the characters that have already been matched exist elsewhere in the pattern
    Calculate the number of alignments which would be skipped by either
        method and apply the method which skips over the most shifts
```

BM String Search: Worst Case and Best Case

> Search with BM can be as bad as ~MN if the pattern M does not actually appear in the main string N

> The other bad case is when all characters of both text and pattern are the same



```
0 1 2 3 4 5 6 7 8 9
B B B B B B B B B B
A B B B B ← pat
  A B B B B
    A B B B B
      A B B B B
        A B B B B
          A B B B B
```

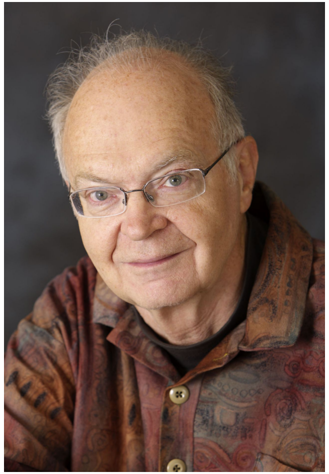
Knuth-Morris-Pratt

> Introduction

> Brute Force/Naive

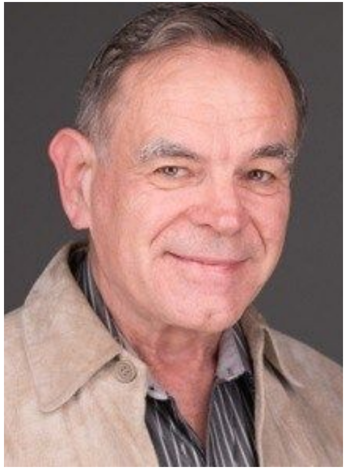
> Boyer-Moore

> Knuth-Morris-Pratt



Donald Knuth

James H. Morris



Vaughan Pratt

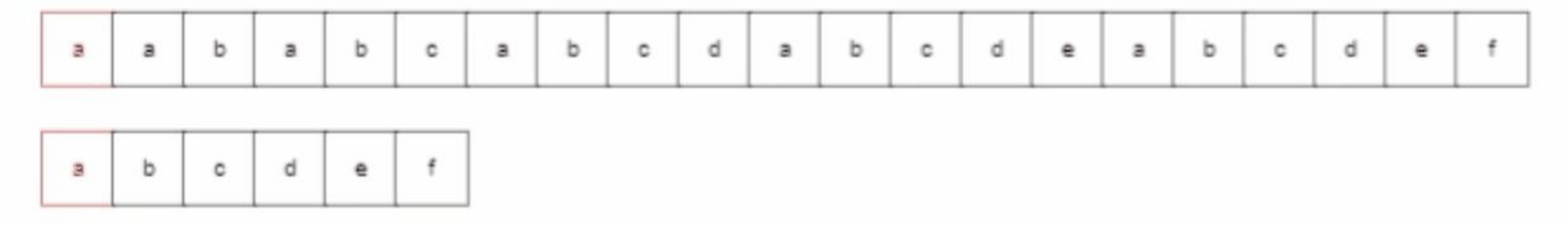
Knuth-Morris-Pratt String Search

The **Knuth-Morris-Pratt** algorithm differs from the naive algorithm by keeping track of information gained from previous comparisons.

A failure function (f) is computed that indicates how much of the last comparison can be used.

This lets us avoid unnecessary backup.

Knuth-Morris-Pratt String Search

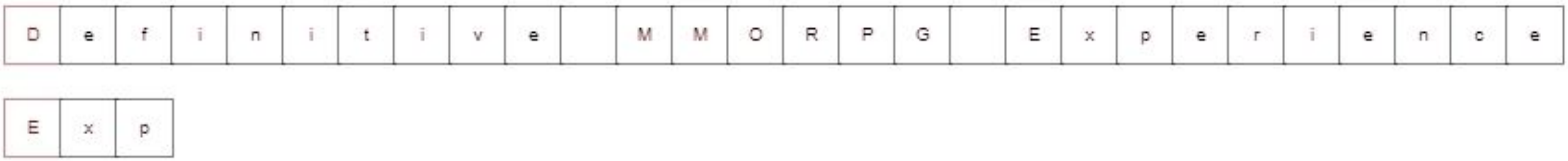


a	a	b	a	b	c	a	b	c	d	a	b	c	d	e	a	b	c	d	e	f

a	b	c	d	e	f

> When used in repetitive patterns, the failure function can help the algorithm jump forward quicker

> KMP struggles with realistic text that is very short in comparison

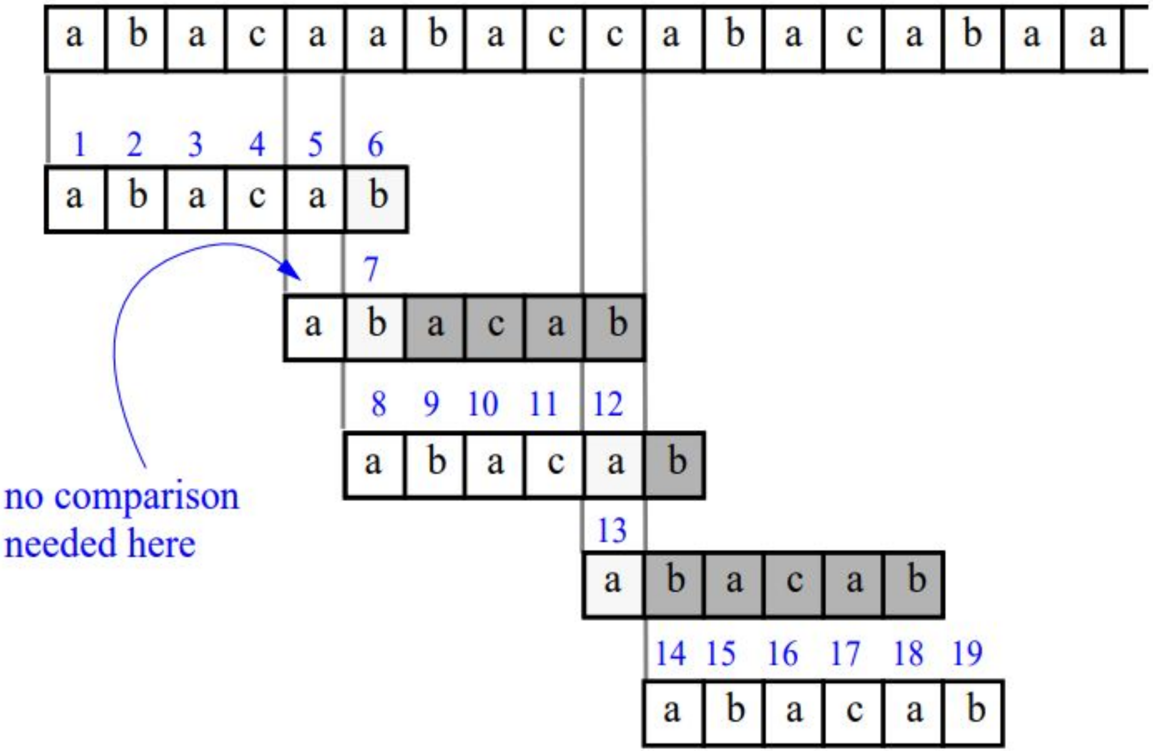


D	e	f	i	n	i	t	i	v	e		M	M	O	R	P	G		E	x	p	e	r	i	e	n	c	e

E	x	p

KMP String Search

> Needs that repetition of characters in the main string *N* to move efficiently



a b a c a a b a c c a b a c a b a a

1 2 3 4 5 6

a b a c a b

7

a b a c a b

8 9 10 11 12

a b a c a b

13

a b a c a b

14 15 16 17 18 19

a b a c a b

no comparison needed here

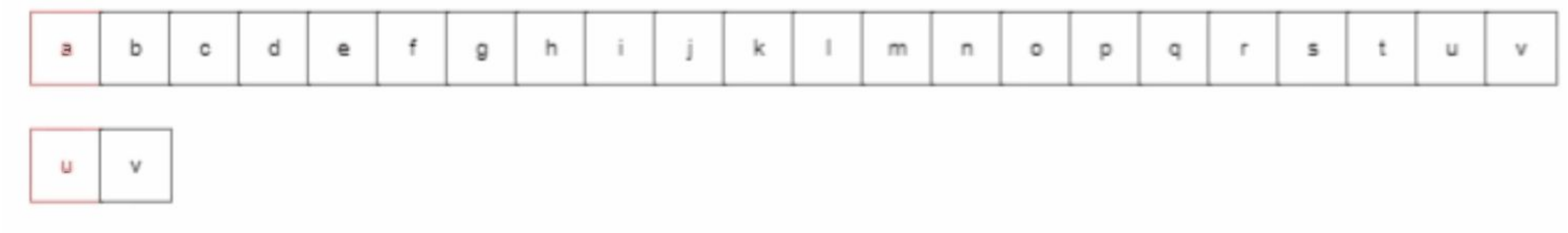
Knuth-Morris-Pratt String Search: Pseudocode

```
n = size of text
m = size of pattern
while i < n, do
   if text[i] = pattern[j], then
      increase i and j by 1
   if j = m, then
      print the location (i-j) as there is the pattern
      j = prefArray[j-1]
   else if i < n AND pattern[j] ≠ text[i] then
      if j ≠ 0 then
         j = prefArray[j - 1]
      else
         increase i by 1
```

KMP String Search: Worst Case

> The worst case will be whenever we have to examine each character in the text and the pattern at least once

> Worst case O(n + m) still, but processing time will still be longer



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
u	v																				

Comparison: When to use which?

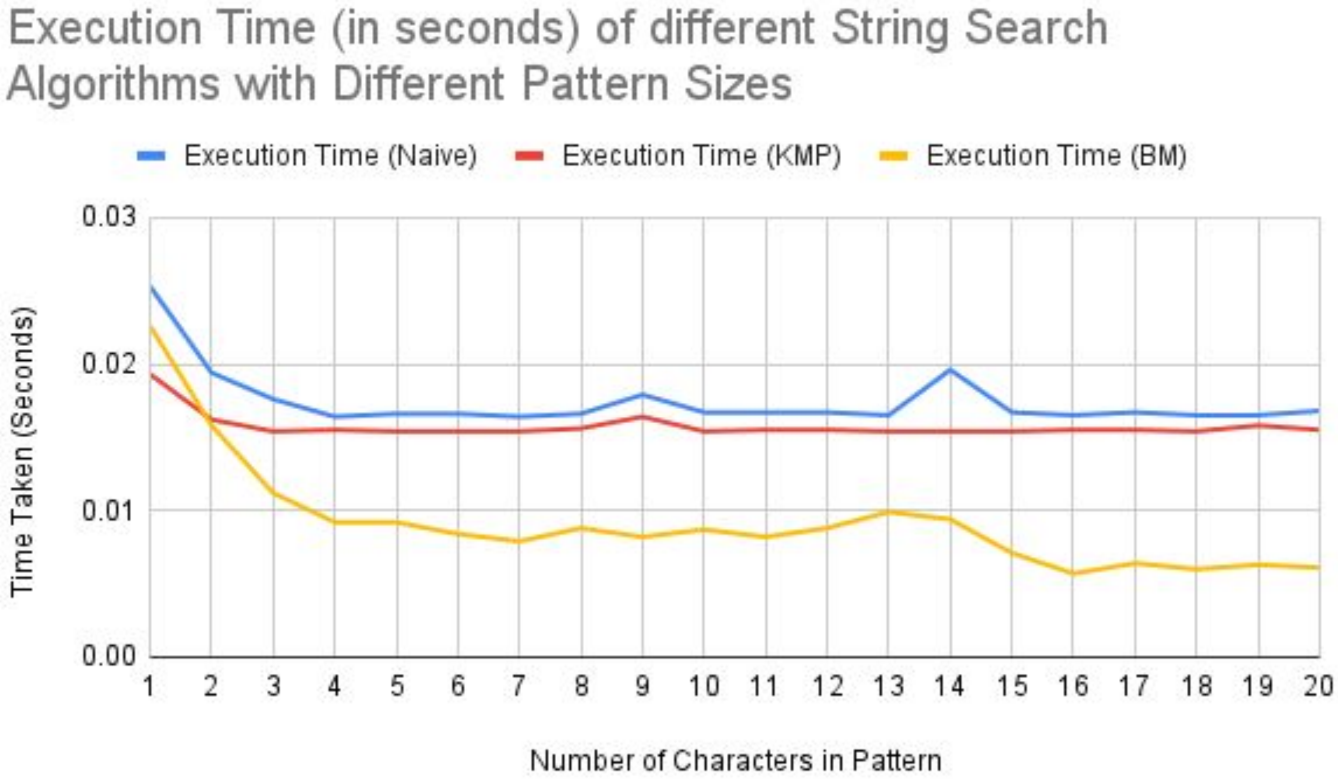
Boyer-Moore

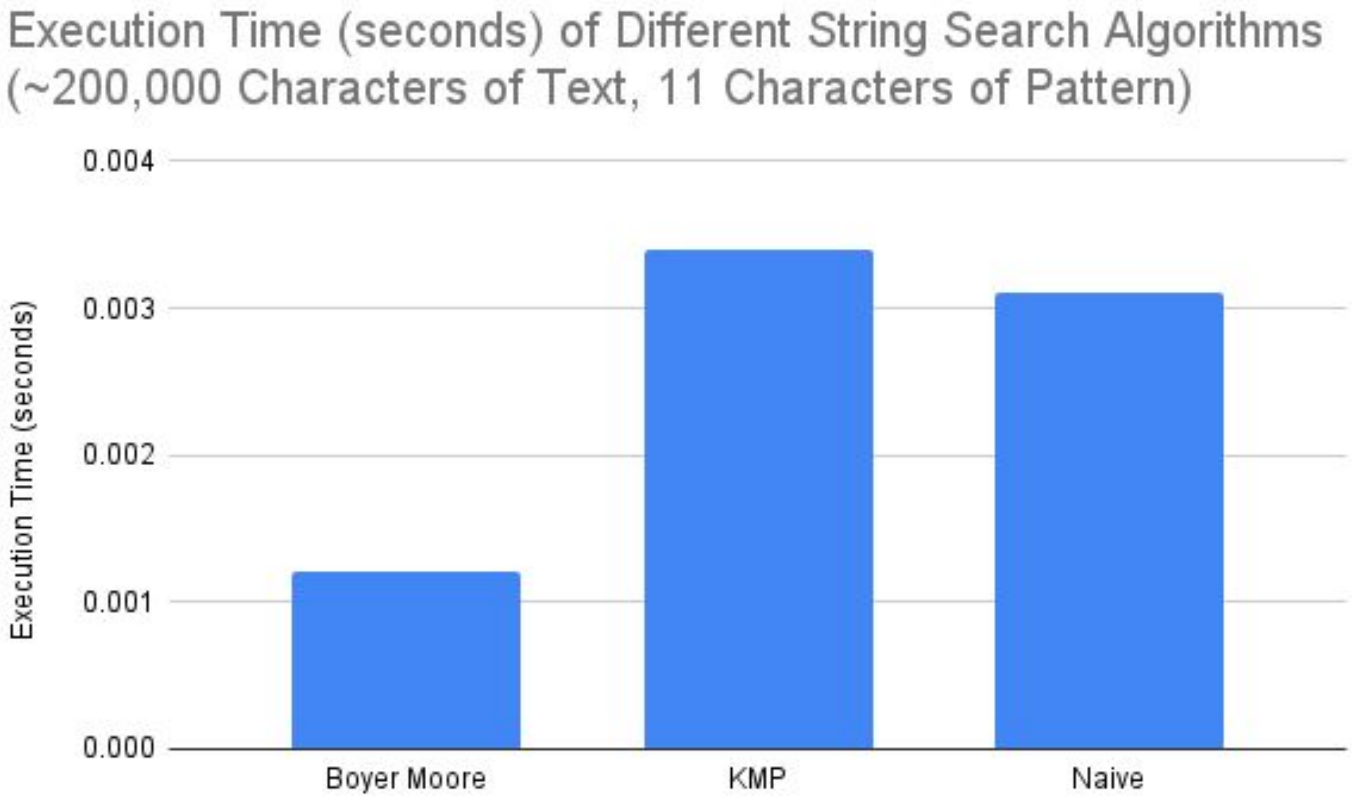
> Long search patterns because you can skip through it much faster

Knuth-Morris-Pratt

> Smaller alphabet (DNA bases) because there will likely be more reusable patterns

The graph compares the algorithms on different DNA sequences





This graph compares the three types of algorithms with randomly generated text

Reference List

B. W. Watson, “Boyer-Moore Algorithm,” Boyer-Moore algorithm. [Online]. Available: https://www-igm.univ-mlv.fr/~lecroq/string/node14.html. [Accessed: 09-Dec-2021].

BAEZA-YATES R., NAVARRO G., RIBEIRO-NETO B., 1999, Indexing and Searching, in Modern Information Retrieval, Chapter 8, p191-228, Addison-Wesley.

M. Abdelpakey, “Knuth-Morris-Pratt String Search Visualization Tool,” Knuth-Morris-Pratt String Search Visualization. [Online]. Available: https://cmps-people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html. [Accessed: 09-Dec-2021].

R. Sedgewick, “Algorithms - Computer Science Department at Princeton ...,” Princeton Computer Science Dept., 14-Apr-2014. [Online]. Available: https://www.cs.princeton.edu/courses/archive/spring18/cos226/lectures/53SubstringSearch.pdf. [Accessed: 09-Dec-2021].

CROCHEMORE, M., RYTTER, W., 1994, Text Algorithms, Oxford University Press.

“Home,” *EverQuest*, 08-Dec-2021. [Online]. Available: https://www.everquest.com/home. [Accessed: 08-Dec-2021].