CS 375 Pattern Searching Algorithms

Jeffrey Chan
Anson Varughese

Pattern Searching Overview

- Brute Force
- Rabin Karp
- Knuth-Morris-Pratt (KMP)
- Time and Space Complexity Comparisons
- Demo

Brute Force

- Given a pattern with length m and text with length n

- For every substring of length m, check if each character matches the corresponding character in the pattern

- Runs in O(nm) time

Rabin Karp

- Calculate hash for the pattern of length m
- Compare to hashes for every substring of length m
- Use a rolling hash

- Average case: O(n+m)
- Worst case: O(nm)

Improve hashing through rolling hash

Text = ABCDEF

Given the hash for ABCD, you can quickly calculate the hash for BCDE in O(1)

A	B	C	D
$base^3$	$base^2$	$base^1$	$base^0$

$$= 65 * base^3 + 66 * base^2 + 67 * base^1 + 68 * base^0$$

Improve hashing through rolling hash

Text = ABCDEF

Given the hash for ABCD, you can quickly calculate the hash for BCDE in O(1)

	B	C	D
base^3	base^2	base^1	base^0

$$= \quad 66 * \text{base}^2 \quad + \quad 67 * \text{base}^1 \quad + \quad 68 * \text{base}^0$$

Improve hashing through rolling hash

Text = ABCDEF

Given the hash for ABCD, you can quickly calculate the hash for BCDE in O(1)

B	C	D	
$base^3$	$base^2$	$base^1$	$base^0$

$$= \quad 66 * base^3 \quad + \quad 67 * base^2 \quad + \quad 68 * base^1$$

Improve hashing through rolling hash

Text = ABCDEF

Given the hash for ABCD, you can quickly calculate the hash for BCDE in O(1)

B	C	D	E
$base^3$	$base^2$	$base^1$	$base^0$

$$= \quad 66 * base^3 \quad + \quad 67 * base^2 \quad + \quad 68 * base^1 \quad + \quad 69 * base^1$$

Rabin Karp Example

Text = <u>ABCD</u>EF

Pattern = CDEF

Pre-process first text window and pattern

Text = <u>ABCD</u>EF Pattern = CDEF Let base = 101

hash(ABCD) = A*base ^{3} + B*base ^{2} + C*base + D

$$= 65*101^3 + 66*101^2 + 67*101 + 68 = 67649666$$

hash(CDEF) = C*base ^{3} + D*base ^{2} + E*base + F

$$= 67*101^3 + 68*101^2 + 69*101 + 70 = 69730874$$

✗

Calculate rolling hash

Text = ABCDEF

Pattern = CDEF

hash(BCDE) = base *(hash(ABCD) - A*101^3) + E

= 101*[67649666 - 65*101^3] + 69

= 68690270

hash(CDEF) = 69730874

✗

Calculate rolling hash

Text = AB<u>CDEF</u> Pattern = CDEF

hash(CDEF) = base *(hash(BCDE) - B*101^3) + F

= 101*[68690270 - 66*101^3] + 70

= 69730874

hash(CDEF) = 69730874 ✓

Now compare each character to verify...

Knuth-Morris-Pratt (KMP)

Searches text using a prefix-suffix array of the pattern to speed up the process

Algorithm:

- Create prefix-suffix array to find repeating substrings in pattern
- Use prefix-suffix array in searching to check repeats in text faster

Runtime :	$\Theta(n+m)$
Space Complexity:	$O(m)$, (where $m = \|pattern\|$)

KMP - Step 1: Create Prefix-Suffix Array

Pattern : AABAAC

A	A	B	A	A	C
0	0	0	0	0	0

Action:

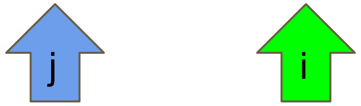
Initialize array elements to 0.

KMP - Step 1: Create Prefix-Suffix Array

Pattern : AABAAC

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

Action:

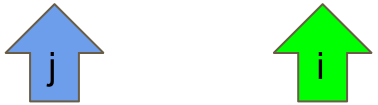


Two pointers to keep track of locations. Since the characters in the pattern match, increment j and set it to arr[i]. Increment i

KMP - Step 1: Create Prefix-Suffix Array

Pattern : AABAAC

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



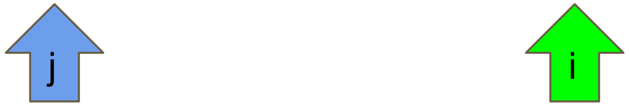
Action:

Since the characters in the pattern DON'T match, we check if j != 0. Since j is 1, we keep i as is and set j = arr[j-1] to go back and check for the prefix.

KMP - Step 1: Create Prefix-Suffix Array

Pattern : AABAAC

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



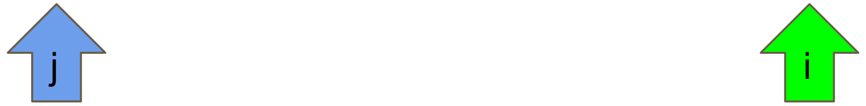
Action:

Since the characters in the pattern DON'T match, we check if j != 0. Since j is 0 now, we increment i. Checking if j == 0 allows us to find the earliest prefix which is also the suffix (substring appearing twice).

KMP - Step 1: Create Prefix-Suffix Array

Pattern : AABAAC

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



Action:

Now i is incremented and we keep repeating that process until we get the final prefix-suffix array.

KMP - Prefix-Suffix Array Complete

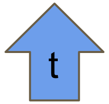
Pattern : AABAAC

A	A	B	A	A	C
0	1	0	1	2	0

KMP - Step 2: Search Text using Prefix-Suffix Array

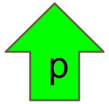
Text: AABAABAAC

A	A	B	A	A	B	A	A	C



Pattern: AABAAC

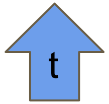
A	A	B	A	A	C
0	1	0	1	2	0



KMP - Step 2: Search Text using Prefix-Suffix Array

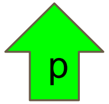
Text: AABAABAAC

A	A	B	A	A	B	A	A	C



Pattern: AABAAC

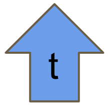
A	A	B	A	A	C
0	1	0	1	2	0



KMP - Step 2: Search Text using Prefix-Suffix Array

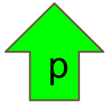
Text: AABAABAAC

A	A	B	A	A	B	A	A	C



Pattern: AABAAC

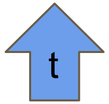
A	A	B	A	A	C
0	1	0	1	2	0



KMP - Step 2: Search Text using Prefix-Suffix Array

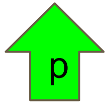
Text: AABAABAAC

A	A	B	A	A	B	A	A	C



Pattern: AABAAC

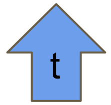
A	A	B	A	A	C
0	1	0	1	2	0



KMP - Step 2: Search Text using Prefix-Suffix Array

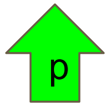
Text: AABAABAAC

A	A	B	A	A	B	A	A	C



Pattern: AABAAC

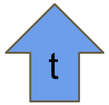
A	A	B	A	A	C
0	1	0	1	2	0



KMP - Step 2: Search Text using Prefix-Suffix Array

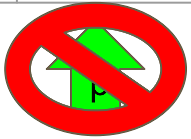
Text: AABAABAAC

A	A	B	A	A	B	A	A	C



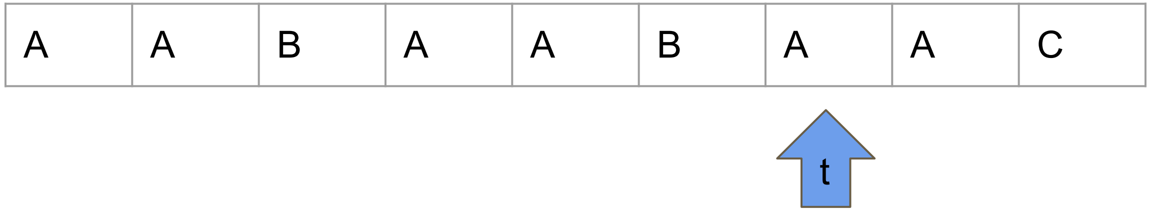
Pattern: AABAAC

A	A	B	A	A	C
0	1	0	1	2	0



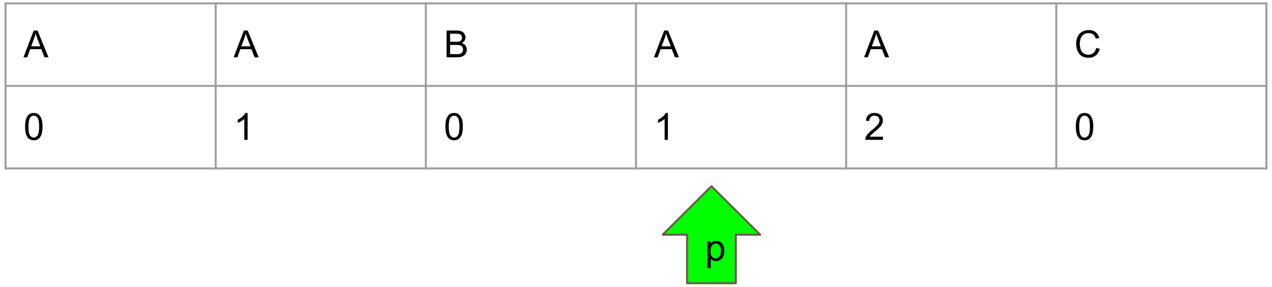
KMP - Step 2: Search Text using Prefix-Suffix Array

Text: AABAABAAC



A	A	B	A	A	B	A	A	C

Pattern: AABAAC

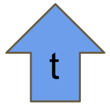


A	A	B	A	A	C
0	1	0	1	2	0

KMP - Step 2: Search Text using Prefix-Suffix Array

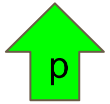
Text: AABAABAAC

A	A	B	A	A	B	A	A	C



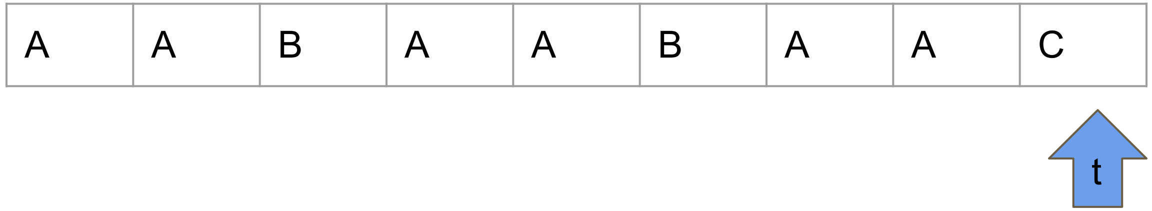
Pattern: AABAAC

A	A	B	A	A	C
0	1	0	1	2	0



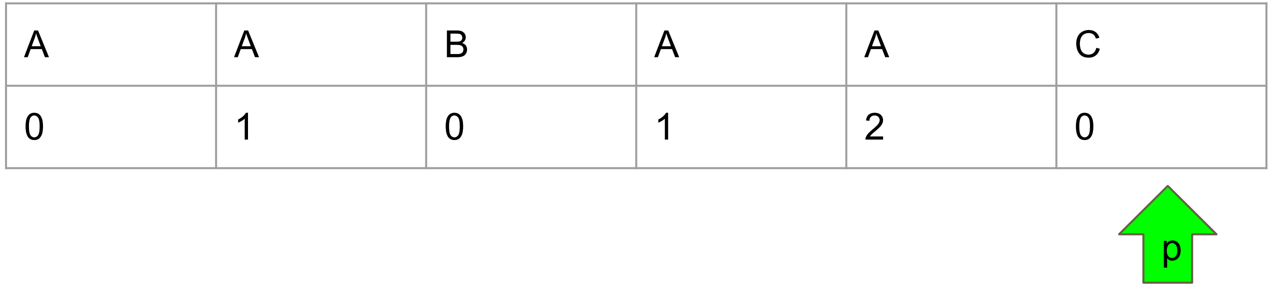
KMP - Step 2: Search Text using Prefix-Suffix Array

Text: AABAABAAC

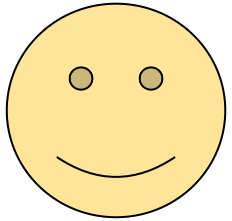


A	A	B	A	A	B	A	A	C

Pattern: AABAAC



A	A	B	A	A	C
0	1	0	1	2	0



Runtime Comparison for Single Pattern Search

	Average Time Complexity	Worst Case Time Complexity	Space Complexity
Brute Force	O(nm)	O(nm)	O(1)
Rabin-Karp	O(n+m)	O(nm)	O(1)
KMP	O(n+m)	O(n+m)	O(m)

Demo

Comparing the pattern searching runtimes of:

- Brute Force
- Rabin-Karp
- KMP

on a random block of text.

Questions?