Algorithms: CZ2001

Iskandar, Andrel, Yong Qiang, Wayne, Jin Han

Overview

Introduction
Choices of Algorithms

The Algorithms
Design & Analysis

Conclusion
Summing it up

Introduction

Brute Force

Sequential Searching

Boyer Moore

Shift Searching

KMP

Pattern Searching

Design of Algorithms

Brute Force Algorithm

Brute Force

- Most Straightforward method
- Compare each letters of pattern and text during iteration
- No Preprocessing is needed

Brute Force

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){
        Int counter = 0;
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        Add counter + 1
                Else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Algorithm

Text (n)	c	a	d	d	b	c	d	c
Pattern (m)	d	d	b					

Brute Force

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){
        Int counter = 0;
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        Add counter + 1
                Else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Algorithm

Text (n)	c	a	d	d	b	c	d	c
Pattern (m)	d	d	b					
		d	d	b				

Brute Force

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){
        Int counter = 0;
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        Add counter + 1
                Else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Algorithm

Text (n)	c	a	d	d	b	c	d	c
Pattern (m)	d	d	b					
		d	d	b				
			d	d	b			

Brute Force

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){
        Int counter = 0;
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        Add counter + 1
                Else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Algorithm

Text (n)	c	a	d	d	b	c	d	c
Pattern (m)	d	d	b					
		d	d	b				
			d	d	b			

Brute Force

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){
        Int counter = 0;
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        Add counter + 1
                Else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Algorithm

Text (n)	c	a	d	d	b	c	d	c
Pattern (m)	d	d	b					
		d	d	b				
			d	d	b			

Brute Force

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){
        Int counter = 0;
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        Add counter + 1
                Else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Algorithm

Text (n)	c	a	d	d	b	c	d	c
Pattern (m)	d	d	b					
		d	d	b				
			d	d	b			
				d	d	b		

Brute Force

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){
        Int counter = 0;
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        Add counter + 1
                Else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Algorithm

Text (n)	c	a	d	d	b	c	d	c
Pattern (m)	d	d	b					
		d	d	b				
			d	d	b			
				d	d	b		
					d	d	b	
						d	d	b

Analysis of Brute Force

Brute Force Analysis

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){ // N Iterations
        int counter = 0;
        for(Loop entire pattern character){ M Iterations
                if(text.letter == pattern.letter)
                        Add counter + 1
                else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Best Case

Text (n)	c	a	b	d	c
Pattern (m)	h	a	b		
		h	a	b	
			h	a	b

Time Complexity

Problem Elements: n - m + 1
Pattern Elements:: 1
Total Comparison: 1(n - m + 1)
Time Complexity: O(n)

Brute Force Analysis

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){ // N Iterations
        int counter = 0;
        for(Loop entire pattern character){ M Iterations
                if(text.letter == pattern.letter)
                        Add counter + 1
                else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Worst Case 1

Text (n)	c	c	c	c	c
Pattern (m)	c	c	a		
		c	c	a	
			c	c	a

Time Complexity

Problem Elements: n - m + 1
Pattern Elements: m
Total Comparison: m(n - m + 1)
Time Complexity: O(mn)

Brute Force Analysis

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){ // N Iterations
        int counter = 0;
        for(Loop entire pattern character){ M Iterations
                if(text.letter == pattern.letter)
                        Add counter + 1
                else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Worst Case 2

Text (n)	c	c	c	c	c
Pattern (m)	c	c	c		
		c	c	c	
			c	c	c

Time Complexity

Problem Elements: n - m + 1
Pattern Elements: m
Total Comparison: m(n - m + 1)
Time Complexity: O(mn)

Brute Force Analysis

Pseudocode

```
var bruteForce = (text, pattern) => {

for(Loop entire text character){ // N Iterations
        int counter = 0;
        for(Loop entire pattern character){ M Iterations
                if(text.letter == pattern.letter)
                        Add counter + 1
                else
                        Exit Pattern Loop
                }
        if(counter == pattern.length)
                Pattern Found in index n!
        }
}
```

Average Case

There are only 4 possible characters - (A, C, G, T) or (A, G U, C)
For each comparison, there is a 1 - m/n% possibility of mismatch.
On average, it will take less than 1 - m/n comparisons for a mismatch to occur.
The upper bound of the average number of comparisons is n/m(n-m+1).

Time Complexity

Problem Elements: n - m + 1
Pattern Elements: n/m
Total Comparison: n/m(n - m + 1)
Time Complexity: O(n)

Boyer-Moore Algorithm

Boyer-Moore Algorithm

- Requires pre-processing method
- Compares characters from the last to the first
- Algorithm run faster on longer patterns

Boyer-Moore Preprocessing

Purpose

- Creates a database to store patterns of the text (Mismatch Shift Table)
- Database contains the bad / good patterns of the text
- Helps to skip sections of the text that has bad pattern during iteration

Bad Patterns

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	c	d	a						

Boyer-Moore Preprocessing

Purpose

- Creates a database to store patterns of the text (Mismatch Shift Table)
- Database contains the bad / good patterns of the text
- Helps to skip sections of the text that has bad pattern during iteration

Bad Patterns

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	c	d	a						
				c	d	a			

Boyer-Moore Preprocessing

Purpose

- Creates a database to store patterns of the text (Mismatch Shift Table)
- Database contains the bad / good patterns of the text
- Helps to skip sections of the text that has bad pattern during iteration

Bad Patterns

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	c	d	a						
				c	d	a			

Good Patterns

Text (n)	d	b	b	d	b	c	d	b	c
Pattern (m) ->	d	b	a	d	b				

Boyer-Moore Preprocessing

Purpose

- Creates a database to store patterns of the text (Mismatch Shift Table)
- Database contains the bad / good patterns of the text
- Helps to skip sections of the text that has bad pattern during iteration

Bad Patterns

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	c	d	a						
				c	d	a			

Good Patterns

Text (n)	d	b	b	d	b	c	d	b	c
Pattern (m) ->	d	b	a	d	b				

Boyer-Moore Preprocessing

Purpose

- Creates a database to store patterns of the text (Mismatch Shift Table)
- Database contains the bad / good patterns of the text
- Helps to skip sections of the text that has bad pattern during iteration

Bad Patterns

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	c	d	a						
				c	d	a			

Good Patterns

Text (n)	d	b	b	d	b	c	d	b	c
Pattern (m) ->	d	b	a	d	b				
				d	b	a	d	b	

Boyer-Moore

PseudoCode

```
var BoyersMoorev2 = (text, pattern) =>{

CreateDatabase(); //Pre-processing

for(Loop entire text character){
        Int NoOfskip = pattern.length
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        if(text.firstletter == pattern.firstletter)
                                NoOfskip -1
                Else
                        Exit Pattern Loop
                }
        if(NoOfskip > 0)
                CheckDatabase();
                SkipString(n, skip);
        Else
                Pattern Found!
        }
}
```

Algorithm

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	d	d	b						

Boyer-Moore

PseudoCode

```
var BoyersMoorev2 = (text, pattern) =>{

CreateDatabase(); //Pre-processing

for(Loop entire text character){
        Int NoOfskip = pattern.length
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        if(text.firstletter == pattern.firstletter)
                                NoOfskip -1
                Else
                        Exit Pattern Loop
                }
        if(NoOfskip > 0)
                CheckDatabase();
                SkipString(n, skip);
        Else
                Pattern Found!
        }
}
```

Algorithm

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	d	d	b						
				d	d	a			

Analysis of Boyer-Moore

Boyer-Moore

Pseudocode

```
var BoyersMoorev2 = (text, pattern) =>{

CreateDatabase(); //Pre-processing

for(Loop entire text character){
        Int NoOfskip = pattern.length
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        if(text.lastetter == pattern.lastletter)
                                NoOfskip -1
                Else
                        Exit Pattern Loop
                }
        if(NoOfskip > 0)
                CheckDatabase();
                SkipString(n, skip);
        Else
                Pattern Found!
        }
}
```

Best Case 1

Text (n)	g	t	c	g	f	h	..
Pattern (m)	c	g	a				
				c	g	a	
							..

Time Complexity

Problem Elements: n
Pattern Elements:: m
Total Comparison: m/n
Time Complexity: O(m/ n)

Boyer-Moore

Pseudocode

```
var BoyersMoorev2 = (text, pattern) =>{

CreateDatabase(); //Pre-processing

for(Loop entire text character){
        Int NoOfskip = pattern.length
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        if(text.lastetter == pattern.lastletter)
                                NoOfskip -1
                Else
                        Exit Pattern Loop
                }
        if(NoOfskip > 0)
                CheckDatabase();
                SkipString(n, skip);
        Else
                Pattern Found!
        }
}
```

Worst Case 1

Text (n)	c	c	c	c	c
Pattern (m)	c	c	c		
		c	c	c	
			c	c	c

Time Complexity

Problem Elements: n
Pattern Elements:: m
Total Comparison: n*m
Time Complexity: O(nm)

Boyer-Moore

Pseudocode

```
var BoyersMoorev2 = (text, pattern) =>{

CreateDatabase(); //Pre-processing

for(Loop entire text character){
        Int NoOfskip = pattern.length
        for(Loop entire pattern character){
                if(text.letter == pattern.letter)
                        if(text.lastetter == pattern.lastletter)
                                NoOfskip -1
                Else
                        Exit Pattern Loop
                }
        if(NoOfskip > 0)
                CheckDatabase();
                SkipString(n, skip);
        Else
                Pattern Found!
        }
}
```

Average Case

Probability:

<u>Assume there's a matching pattern in the text</u>

Matching Pattern: 1 / (nPm) [1 / n * 1/ n - 1 *... * 1/1]

Matching Pattern with repetitions: (1 / [nPm / r!])

m! / (r = No of character repetitions)

Example: 'hello world' = 10! / 3! * 2!

No matching pattern: 1 - (Matching patterns)

Average Case

Problem Elements: n
Pattern Elements:: m

Matching patterns: (1/nPm)*(m/n*[m - 1/n]* [2m+1]/2) + (1/[nPm / r!])*(m/n*[m - 1/n]* [2m+1]/2)
No matching patterns: 1 - (1/nPm)*(m/n*[m - 1/n]* [2m+1]/2) + 1 - (1/[nPm / r!])*(m/n*[m - 1/n]* [2m+1]/2))
Worse case: n*m
Average Case: (Matching patterns + No matching patterns) + Worse Case

(1/nPm)*(n/m*[m - 1/n]* [2m+1]/2) + (1/[nPm / r!])*(n/m*[m - 1/n]* [2m+1]/2) + 1- ((1/nPm)*(n/m*[m - 1/n]* [2m+1]/2) + 1- (1/[nPm / r!])*(n/m*[m - 1/n]* [2m+1]/2)) + m*n

= O(m/n) + O(m/n) + O(m/n) + O(m/n) + O(mn)

Time Complexity: **O(m/n + mn)**

KMP Algorithm

KMP

- Requires pre-processing
- Best suited for when the size of the alphabet is small

KMP

PseudoCode

Building lps[] array

Pattern (m)	a	a	a	b			
		a	a	a	b		

```
// Preprocess the pattern (calculate lps[] array)
FindPrepocessingPattern.Run(pattern, M, lps);

int text_index = 0; // index for txt[]
while (text_index < N) {
    if (pattern.charAt(pattern_index) == file.charAt(text_index)) {
        pattern_index++;
        text_index++;
    }
    if (pattern_index == M) {
        System.out.println("Found pattern " + "at index " + (text_index - pattern_index));
        pattern_index = lps[pattern_index - 1];
    }
    // mismatch after j matches
    else if (text_index < N && pattern.charAt(pattern_index) != file.charAt(text_index)) {
        //Make comparison on mismatched text char with the list
        if (pattern_index != 0 && pattern.indexOf(file.charAt(text_index)) != 1)
            pattern_index = lps[pattern_index - 1];
        else
            text_index = text_index + 1;
    }
```

Index	0	1	2	3
Pattern	a	a	a	b
Match Value	0			

KMP

PseudoCode

```
// Preprocess the pattern (calculate lps[] array)
FindPrepocessingPattern.Run(pattern, M, lps);

int text_index = 0; // index for txt[]
while (text_index < N) {
    if (pattern.charAt(pattern_index) == file.charAt(text_index)) {
        pattern_index++;
        text_index++;
    }
    if (pattern_index == M) {
        System.out.println("Found pattern " + "at index " + (text_index - pattern_index));
        pattern_index = lps[pattern_index - 1];
    }
    // mismatch after j matches
    else if (text_index < N && pattern.charAt(pattern_index) != file.charAt(text_index)) {
        //Make comparison on mismatched text char with the list
        if (pattern_index != 0 && pattern.indexOf(file.charAt(text_index)) != 1)
            pattern_index = lps[pattern_index - 1];
        else
            text_index = text_index + 1;
    }
```

Building lps[] array

Pattern (m)	a	a	a	b			
		a	a	a	b		

Index	0	1	2	3
Pattern	a	a	a	b
Match Value	0	1		

KMP

PseudoCode

Building lps[] array

Pattern (m)	a	a	a	b			
		a	a	a	b		

```
// Preprocess the pattern (calculate lps[] array)
FindPrepocessingPattern.Run(pattern, M, lps);

int text_index = 0; // index for txt[]
while (text_index < N) {
    if (pattern.charAt(pattern_index) == file.charAt(text_index)) {
        pattern_index++;
        text_index++;
    }
    if (pattern_index == M) {
        System.out.println("Found pattern " + "at index " + (text_index - pattern_index));
        pattern_index = lps[pattern_index - 1];
    }
    // mismatch after j matches
    else if (text_index < N && pattern.charAt(pattern_index) != file.charAt(text_index)) {
        //Make comparison on mismatched text char with the list
        if (pattern_index != 0 && pattern.indexOf(file.charAt(text_index)) != 1)
            pattern_index = lps[pattern_index - 1];
        else
            text_index = text_index + 1;
    }
```

Index	0	1	2	3
Pattern	a	a	a	b
Match Value	0	1	2	

KMP

PseudoCode

Building lps[] array

Pattern (m)	a	a	a	b			
		a	a	a	b		

```
// Preprocess the pattern (calculate lps[] array)
FindPrepocessingPattern.Run(pattern, M, lps);

int text_index = 0; // index for txt[]
while (text_index < N) {
    if (pattern.charAt(pattern_index) == file.charAt(text_index)) {
        pattern_index++;
        text_index++;
    }
    if (pattern_index == M) {
        System.out.println("Found pattern " + "at index " + (text_index - pattern_index));
        pattern_index = lps[pattern_index - 1];
    }
    // mismatch after j matches
    else if (text_index < N && pattern.charAt(pattern_index) != file.charAt(text_index)) {
        //Make comparison on mismatched text char with the list
        if (pattern_index != 0 && pattern.indexOf(file.charAt(text_index)) != 1)
            pattern_index = lps[pattern_index - 1];
        else
            text_index = text_index + 1;
    }
```

Index	0	1	2	3
Pattern	a	a	a	b
Match Value	0	1	2	0

KMP

Pseudocode

```
// Preprocess the pattern (calculate lps[] array)
FindPrepocessingPattern.Run(pattern, M, lps);

int text_index = 0; // index for txt[]
while (text_index < N) {
    if (pattern.charAt(pattern_index) == file.charAt(text_index)) {
        pattern_index++;
        text_index++;
    }
    if (pattern_index == M) {
        System.out.println("Found pattern " + "at index " + (text_index - pattern_index));
        pattern_index = lps[pattern_index - 1];
    }
    // mismatch after j matches
    else if (text_index < N && pattern.charAt(pattern_index) != file.charAt(text_index)) {
        //Make comparison on mismatched text char with the list
        if (pattern_index != 0 && pattern.indexOf(file.charAt(text_index)) != 1)
            pattern_index = lps[pattern_index - 1];
        else
            text_index = text_index + 1;
    }
```

Algorithm

Text (n)	a	a	c	a	a	a	b	b	c	c
Pattern (m)	a	a	a	b						

Index	0	1	2	3
Pattern	a	a	a	b
Match Value	0	1	2	0

KMP

Pseudocode

```
// Preprocess the pattern (calculate lps[] array)
FindPrepocessingPattern.Run(pattern, M, lps);

int text_index = 0; // index for txt[]
while (text_index < N) {
    if (pattern.charAt(pattern_index) == file.charAt(text_index)) {
        pattern_index++;
        text_index++;
    }
    if (pattern_index == M) {
        System.out.println("Found pattern " + "at index " + (text_index - pattern_index));
        pattern_index = lps[pattern_index - 1];
    }
    // mismatch after j matches
    else if (text_index < N && pattern.charAt(pattern_index) != file.charAt(text_index)) {
        //Make comparison on mismatched text char with the list
        if (pattern_index != 0 && pattern.indexOf(file.charAt(text_index)) != 1)
            pattern_index = lps[pattern_index - 1];
        else
            text_index = text_index + 1;
    }
```

Algorithm

Text (n)	a	a	c	a	a	a	b	b	c	c
Pattern (m)	a	a	a	b						
		a	a	a	b					

Index	0	1	2	3
Pattern	a	a	a	b
Match Value	0	1	2	0

KMP

Pseudocode

```
// Preprocess the pattern (calculate lps[] array)
FindPrepocessingPattern.Run(pattern, M, lps);

int text_index = 0; // index for txt[]
while (text_index < N) {
    if (pattern.charAt(pattern_index) == file.charAt(text_index)) {
        pattern_index++;
        text_index++;
    }
    if (pattern_index == M) {
        System.out.println("Found pattern " + "at index " + (text_index - pattern_index));
        pattern_index = lps[pattern_index - 1];
    }
    // mismatch after j matches
    else if (text_index < N && pattern.charAt(pattern_index) != file.charAt(text_index)) {
        //Make comparison on mismatched text char with the list
        if (pattern_index != 0 && pattern.indexOf(file.charAt(text_index)) != 1)
            pattern_index = lps[pattern_index - 1];
        else
            text_index = text_index + 1;
    }
```

Algorithm

Text (n)	a	a	c	a	a	a	b	b	c	c
Pattern (m)	a	a	a	b						
		a	a	a	b					
				a	a	a	b			

Index	0	1	2	3
Pattern	a	a	a	b
Match Value	0	1	2	0

Analysis of KMP

Modified KMP

Best Case

Text (n)	g	t	c	g	t
Pattern (m)	a	t	c		
		a	t	c	
			a	t	c

Time Complexity

Problem Elements: n
Pattern Elements:: m
Total Comparison: n - m + 1
Time Complexity: O(n - m)

Modified KMP

Worst Case

Text (n)	a	a	a	a	a
Pattern (m)	a	a	b		
		a	a	$\boxed{b}$	
			a	a	$\boxed{b}$

Time Complexity

Problem Elements: n
Pattern Elements:: m
Total Comparison: 2n - m
Time Complexity: O(n)

Modified KMP

Average case

Index	0	1	2	3	4
Text (n)	a	a	a	a	a
Pattern (m)	a	a	a		
		a	a	a	
			a	a	a

$P(i) = 1/(n-m+1)$

No. of Comparisons = $m + (m+2) + (m+4)+\ldots+(m+2(n-m))$

$$= \sum_{i=0}^{n-m} 2i + m$$

Avg success = $1/(n-m+1) \times \sum_{i=0}^{n-m} 2i + m$

$= n$

Avg failed = $2n-m$

Avg time complexity = $(\frac{1}{4})n + (\frac{3}{4})(2n-m)$

Time Complexity

Problem Elements: n - 1

Pattern Elements: m

Total Comparison: $(7/4)n - (\frac{3}{4})m$

Time Complexity: O(n)

Conclusion

Brute Force

Least Effective Method,
Worst Time Complexity

KMP

Pattern Searching
Rather Effective Method

Boyer Moore

Most Effective Method,
Ideal Time Complexity

Boyers Moore Preprocessing

Purpose

- Creates a database to store patterns of the text (Mismatch Shift Table)
- Database contains the bad / good patterns of the text
- Helps to skip sections of the text that has bad pattern during iteration

Mismatch Shift Table

Index	0	1	2	3	4	5	6
Letters	d	r	i	b	b	l	e

Letters	d	r	i	b	l	e	*
Number of shift	6	5	4	2	2	1	7

KMP

Purpose

- Creates a database to store patterns of the text
- Database contains the bad / good patterns of the text
- Helps to skip sections of the text that has bad pattern during iteration
- Pre-processing is the same concept to Boyers Moore but searching method is different

Bad Patterns

Text (n)	c	a	b	d	d	a	c	d	c
Pattern (m) ->	c	d	a						

Good Patterns

Text (n)	d	b	b	d	b	c	d	b	c
Pattern (m) ->	d	b	a	d	b				
				d	b	a	d	b	

Introduction

01

Genomes and data used

Design

02

Searching Algorithm Methods

Analysis

03

Complexity of the algorithms

Conclusion

04

Summing it up

Introduction

Brute Force

Sequential Searching

Boyers Moore

Shift Searching

KMP

Pattern Searching

Conclusion

Brute Force

Least Effective Method,
Worst Time Complexity

KMP

Rather Effective Search

Boyer Moore

Most Effective Method,
Ideal Time Complexity