Chapter 7
Indexing and Searching

Introduction

- How to retrieval information?
- A simple alternative is to search the whole text sequentially
- Another option is to build data structures over the text (called ***indices***) to speed up the search

Introduction

Indexing techniques:

- Inverted files
- Suffix arrays
- Signature files

Keywords and Controlled Vocabulary

Keyword:

A term that is used to describe the *subject matter* in a document. It is sometimes called an **index term**. Keywords can be extracted automatically from a document or assigned by a human **cataloguer** or **indexer**.

Controlled vocabulary:

A list of words that can be used as keywords, e.g., in a medical system, a list of medical terms.

Inverted file (more complete definition):

A list of the **keywords that apply to a set of documents**, the documents in which they appear and related information.

Inverted Files

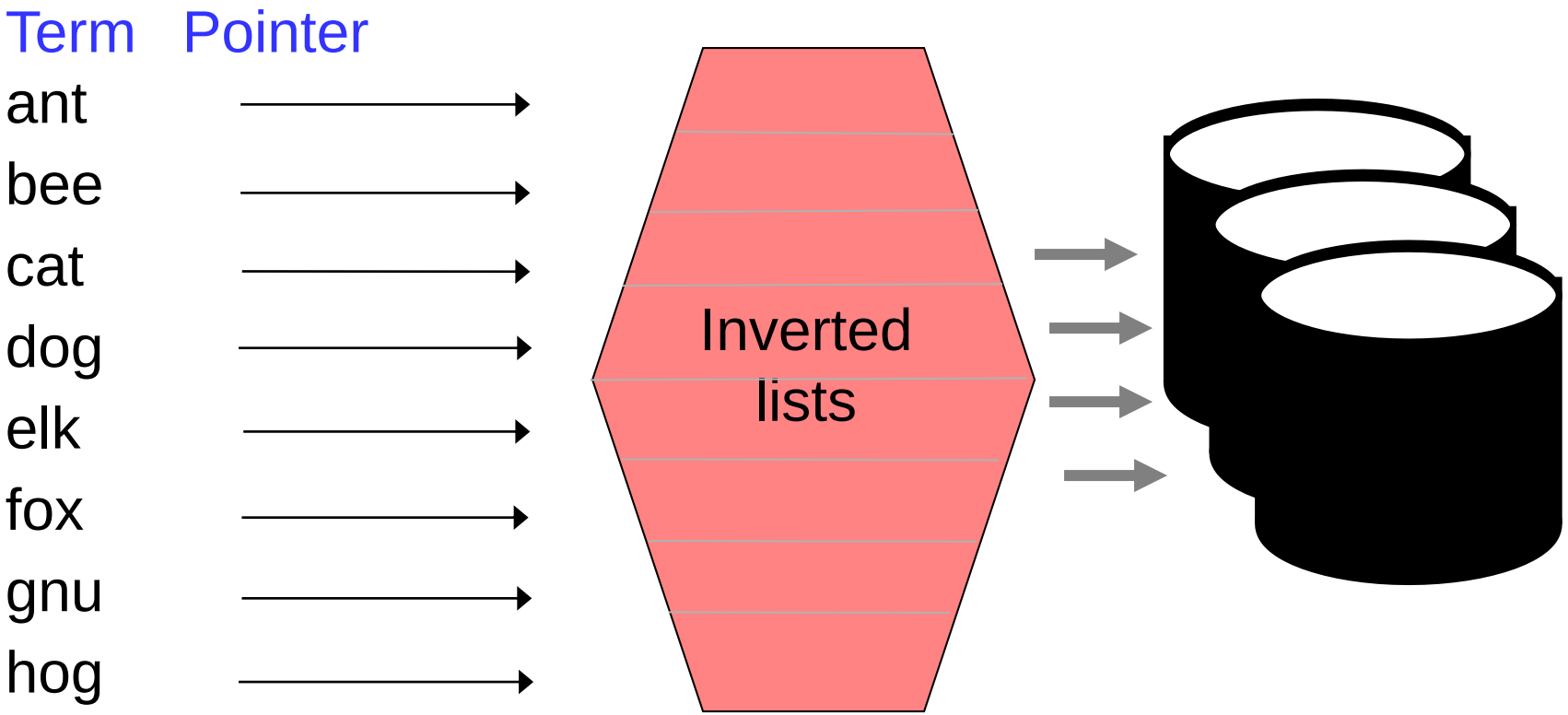
- Definition: an inverted file is a word-oriented mechanism for indexing a text collection in order to speed up the searching task.
- Structure of inverted file:
  - **Vocabulary**: is the set of all distinct words in the text
  - **Occurrences**: lists containing all information necessary for each word of the vocabulary (text position, frequency, documents where the word appears, etc.)

Organization of Inverted Files

Index file

Postings file

Documents file



Example

- Text:

1 6 12 16 18 25 29 36 40 45 54 58 66 70
That house has a garden. The garden has many flowers. The flowers are beautiful

- Inverted file

Vocabulary	Occurrences
beautiful	70
flowers	45, 58
garden	18, 29
house	6

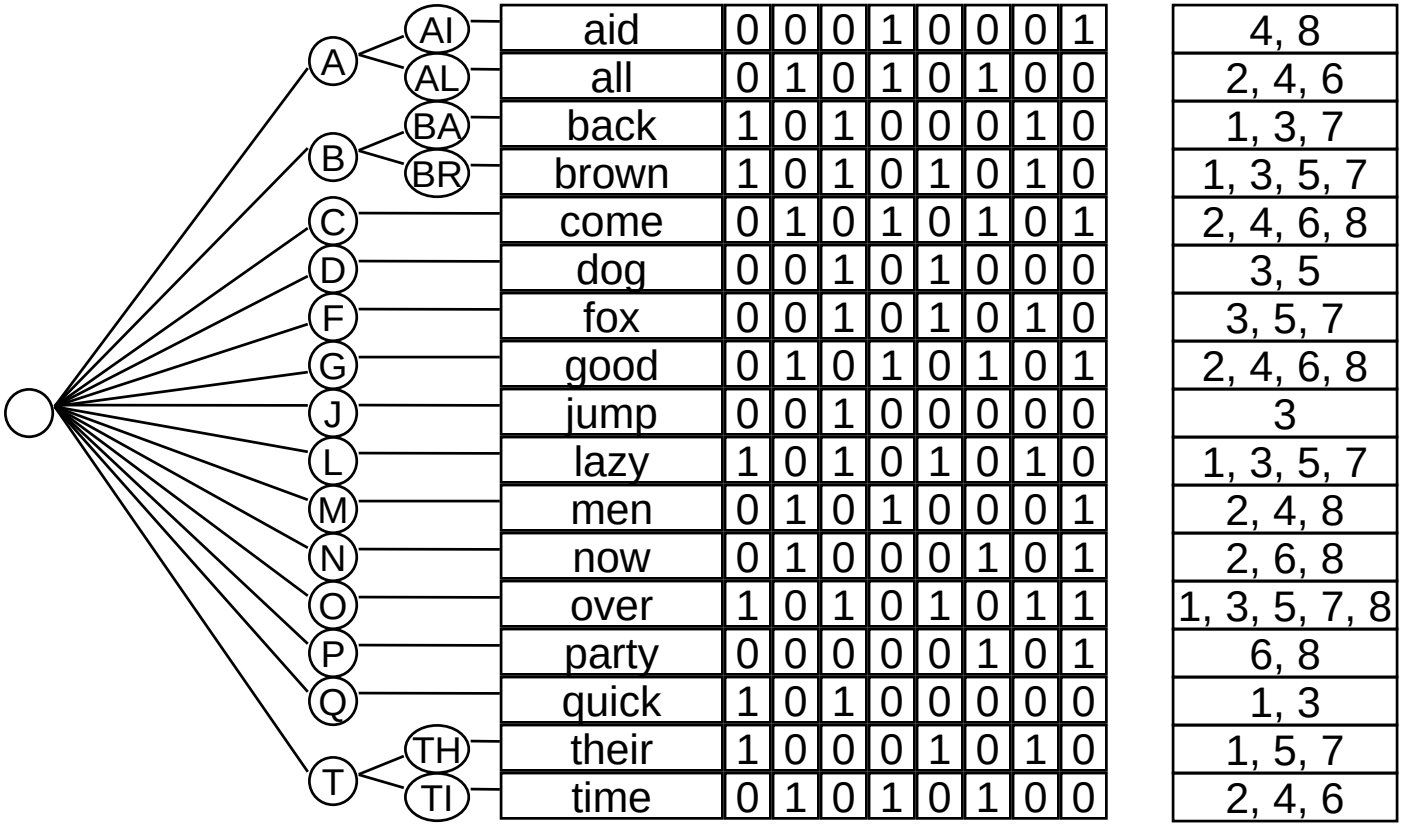
Example

Inverted file: a list of the words in a set of documents and the documents in which they appear.

Word	***Document***
abacus	3
	19
	22
actor	2
	19
	29
aspen	5
atoll	11
	34

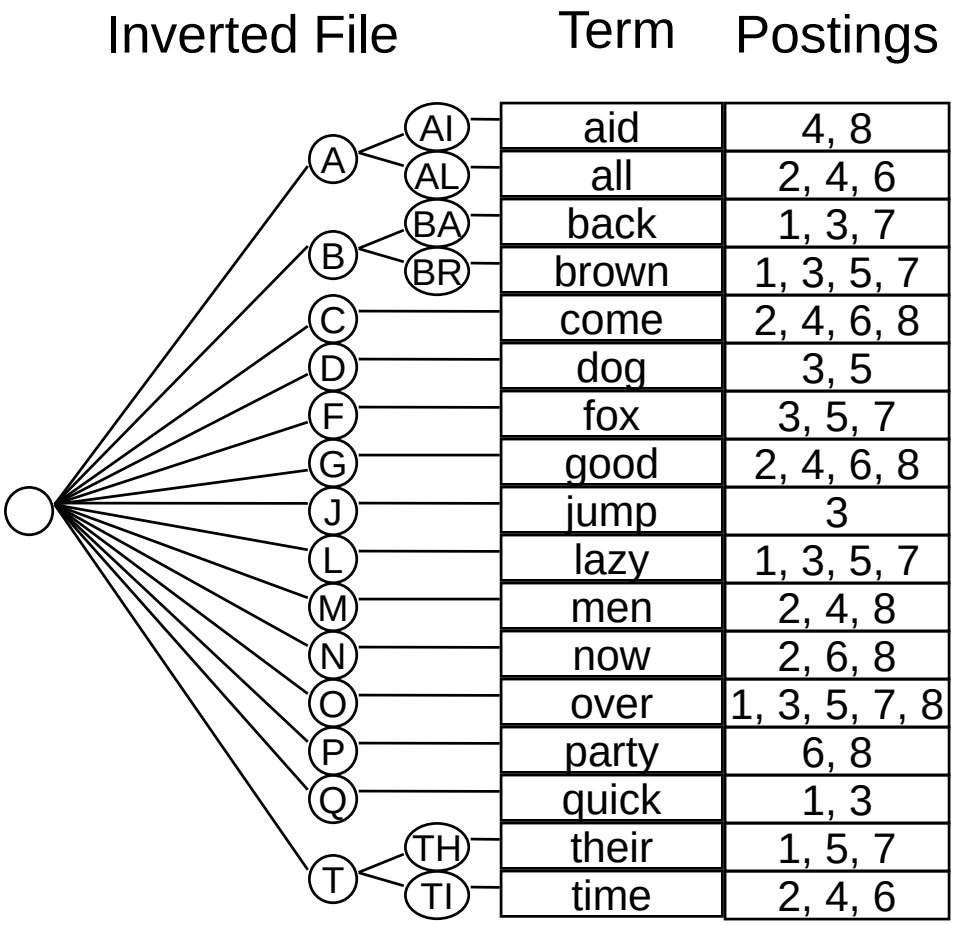
Stop words are removed and stemming carried out before building the index.

An Example



Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8	Postings
aid	0	0	0	1	0	0	0	1	4, 8
all	0	1	0	1	0	1	0	0	2, 4, 6
back	1	0	1	0	0	0	1	0	1, 3, 7
brown	1	0	1	0	1	0	1	0	1, 3, 5, 7
come	0	1	0	1	0	1	0	1	2, 4, 6, 8
dog	0	0	1	0	1	0	0	0	3, 5
fox	0	0	1	0	1	0	1	0	3, 5, 7
good	0	1	0	1	0	1	0	1	2, 4, 6, 8
jump	0	0	1	0	0	0	0	0	3
lazy	1	0	1	0	1	0	1	0	1, 3, 5, 7
men	0	1	0	1	0	0	0	1	2, 4, 8
now	0	1	0	0	0	1	0	1	2, 6, 8
over	1	0	1	0	1	0	1	1	1, 3, 5, 7, 8
party	0	0	0	0	0	1	0	1	6, 8
quick	1	0	1	0	0	0	0	0	1, 3
their	1	0	0	0	1	0	1	0	1, 5, 7
time	0	1	0	1	0	1	0	0	2, 4, 6

The Finished Product



Block Addressing

- The text is divided in blocks
- The occurrences point to the blocks where the word appears
- Advantages:
  - the number of pointers is smaller than positions
  - all the occurrences of a word inside a single block are collapsed to one reference
- Disadvantages:
  - online search over the qualifying blocks if exact positions are required

Example

- Text:

Block 1	Block 2	Block 3	Block 4
That house has a	garden. The garden has	many flowers. The flowers	are beautiful

- Inverted file:

Vocabulary	Occurrences
beautiful	4
flowers	3
garden	2
house	1

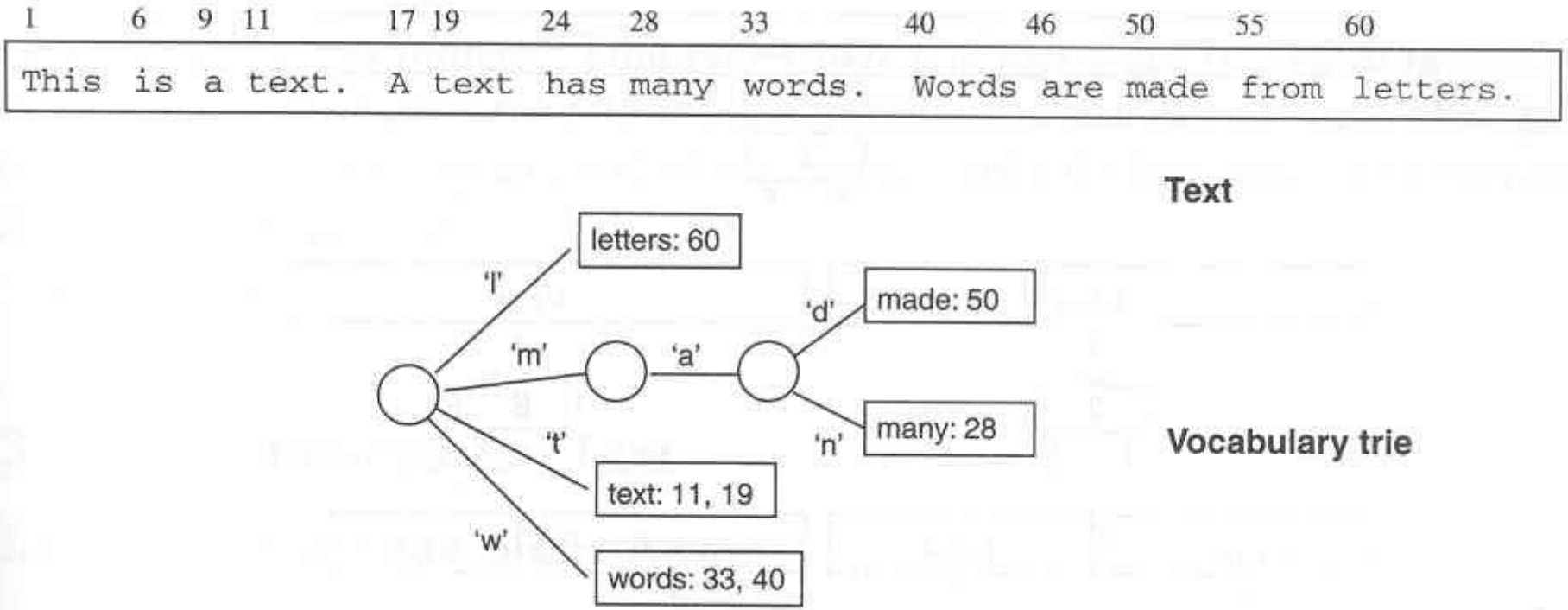
Searching

- The search algorithm on an inverted index follows three steps:
  - **Vocabulary search:** the words present in the query are searched in the vocabulary
  - **Retrieval occurrences:** the lists of the occurrences of all words found are retrieved
  - **Manipulation of occurrences:** the occurrences are processed to solve the query

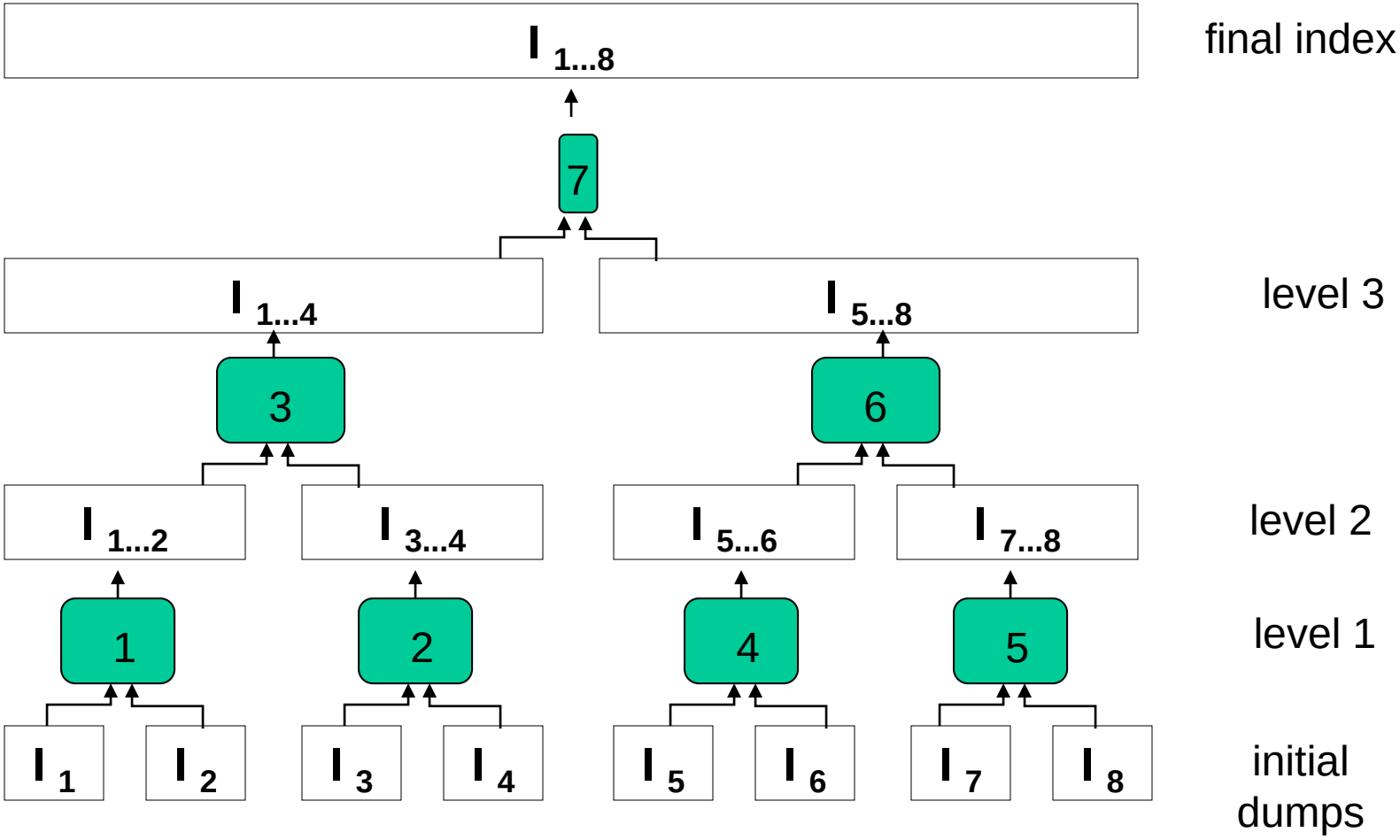
Construction

- All the vocabulary is kept in a **suitable data structure** storing for each word a list of its occurrences
- Each word of the text is **read and searched** in the vocabulary
- If it is not found, it is added to the vocabulary with a empty list of occurrences and the new position is added to the end of its list of occurrences

Example


1 6 9 11 17 19 24 28 33 40 46 50 55 60
This is a text. A text has many words. Words are made from letters.
Text
letters: 60
'l'
'm'
'a'
'd'
made: 50
'n'
many: 28
't'
text: 11, 19
'w'
words: 33, 40
Vocabulary trie

Example


I 1...8
7
I 1...4
I 5...8
3
6
I 1...2
I 3...4
I 5...6
I 7...8
1
2
4
5
I 1
I 2
I 3
I 4
I 5
I 6
I 7
I 8
final index
level 3
level 2
level 1
initial dumps

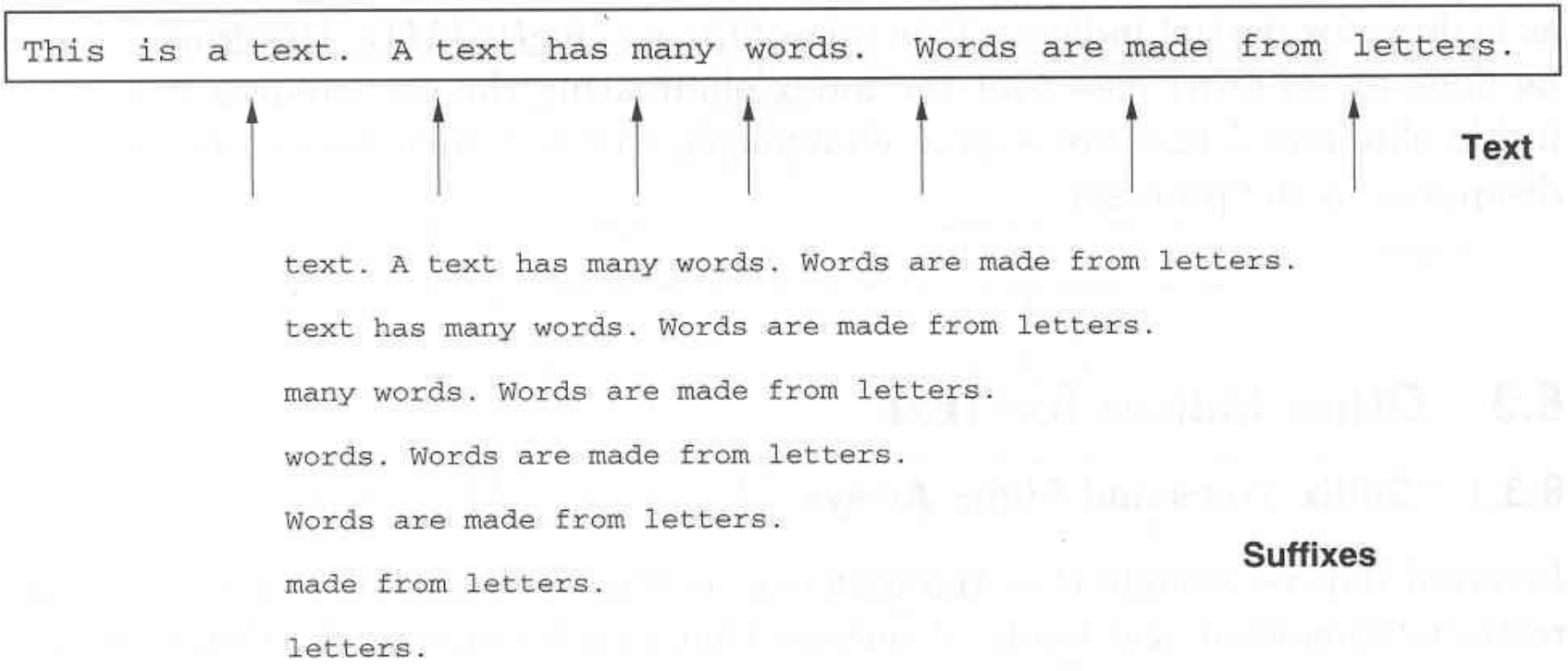
Conclusion

- Inverted file is probably the most adequate indexing technique for database text
- The indices are appropriate when the text collection is **large** and **semi-static**
- Otherwise, if the text collection is **volatile** **online** searching is the only option
- Some techniques combine online and indexed searching

Other Indeice for Text

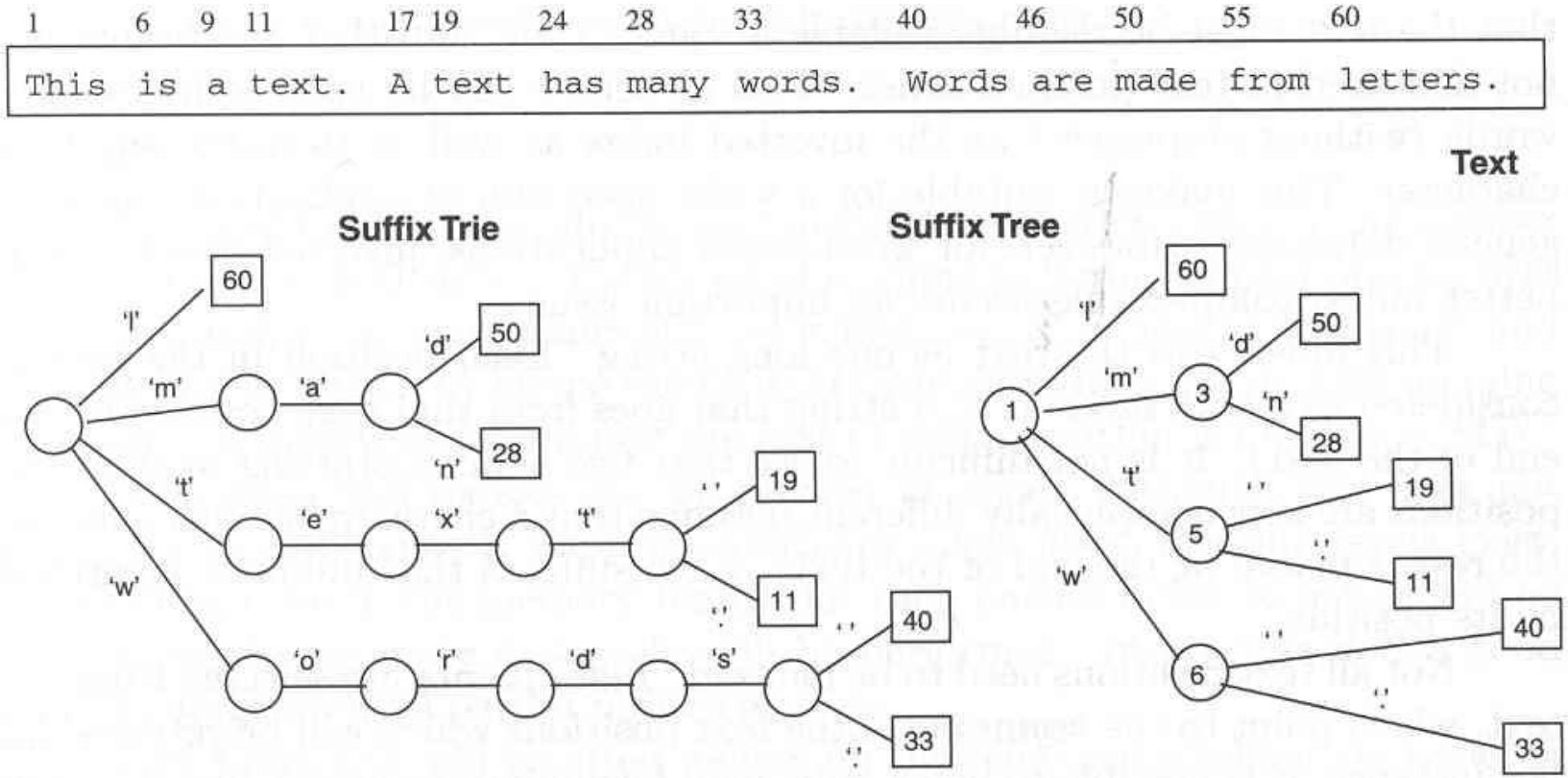
- Suffix Trees and Suffix Arrays
- Signature Files

Suffix Trees and Suffix Arrays



The sample text with index points of interest marked.Below, the suffixes corresponding to those index points

Suffix Trees and Suffix Arrays



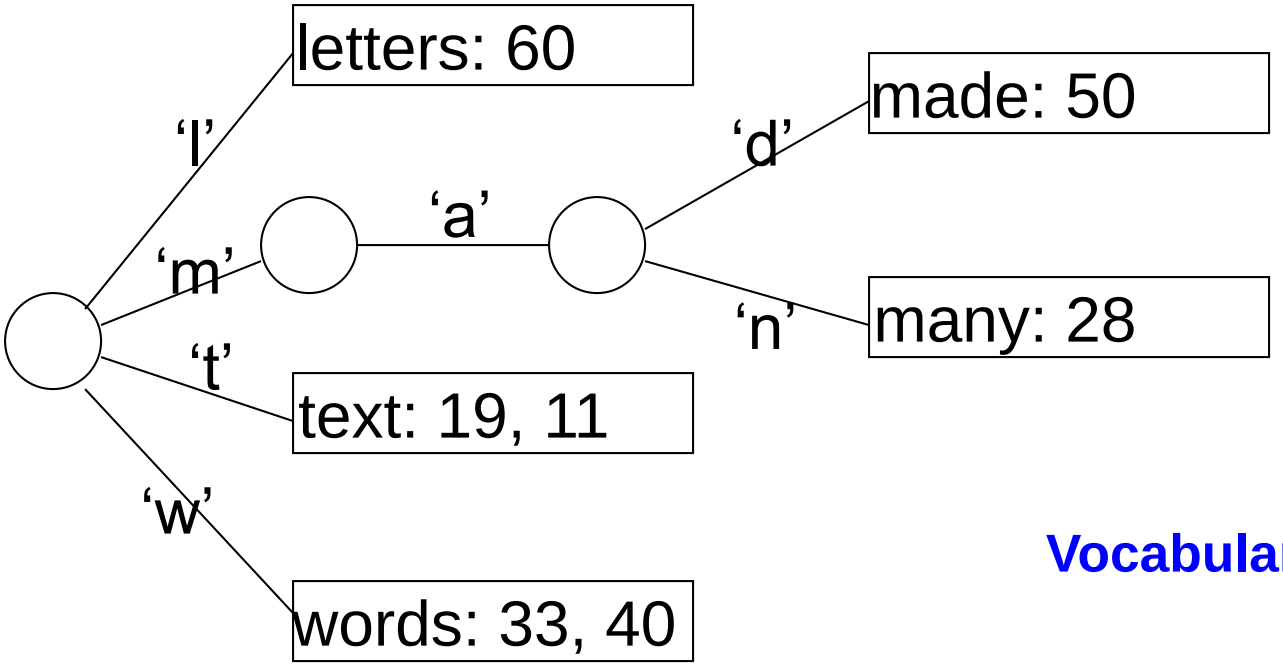
The suffix trie and suffix tree for the sample text.

Trie

1 6 9 11 17 19 24 28 33 40 46 50 55 60

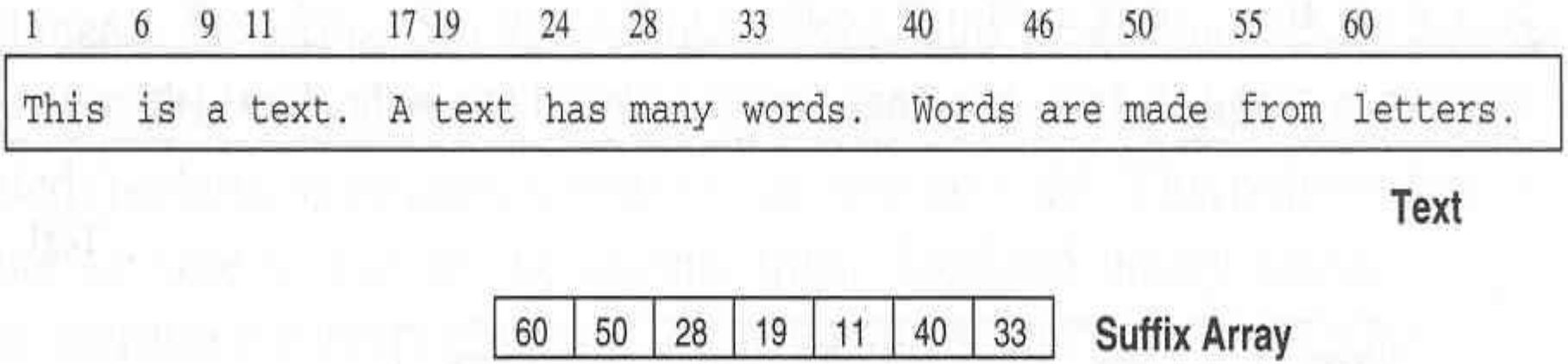
This is a text. A text has many words. Words are made from letters.

Text



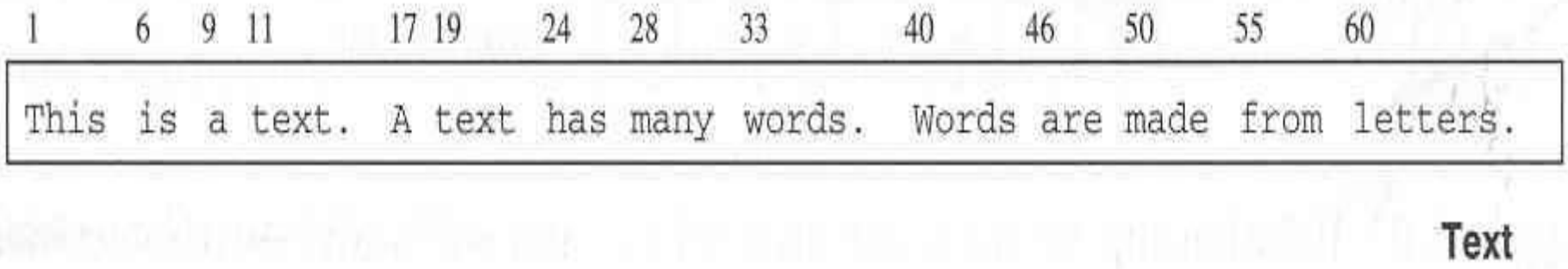
Vocabulary trie

Suffix Trees and Suffix Arrays



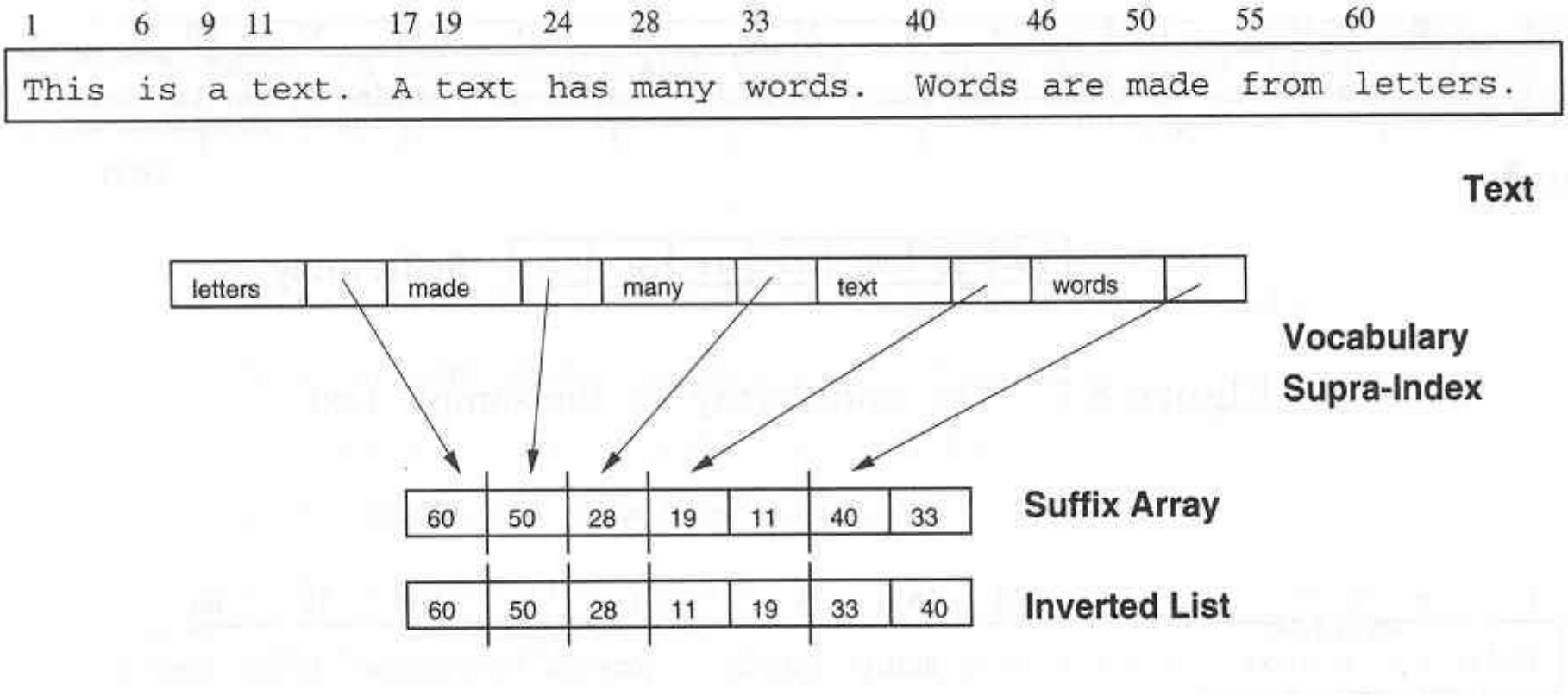
The suffix array for the sample text.

Suffix Trees and Suffix Arrays



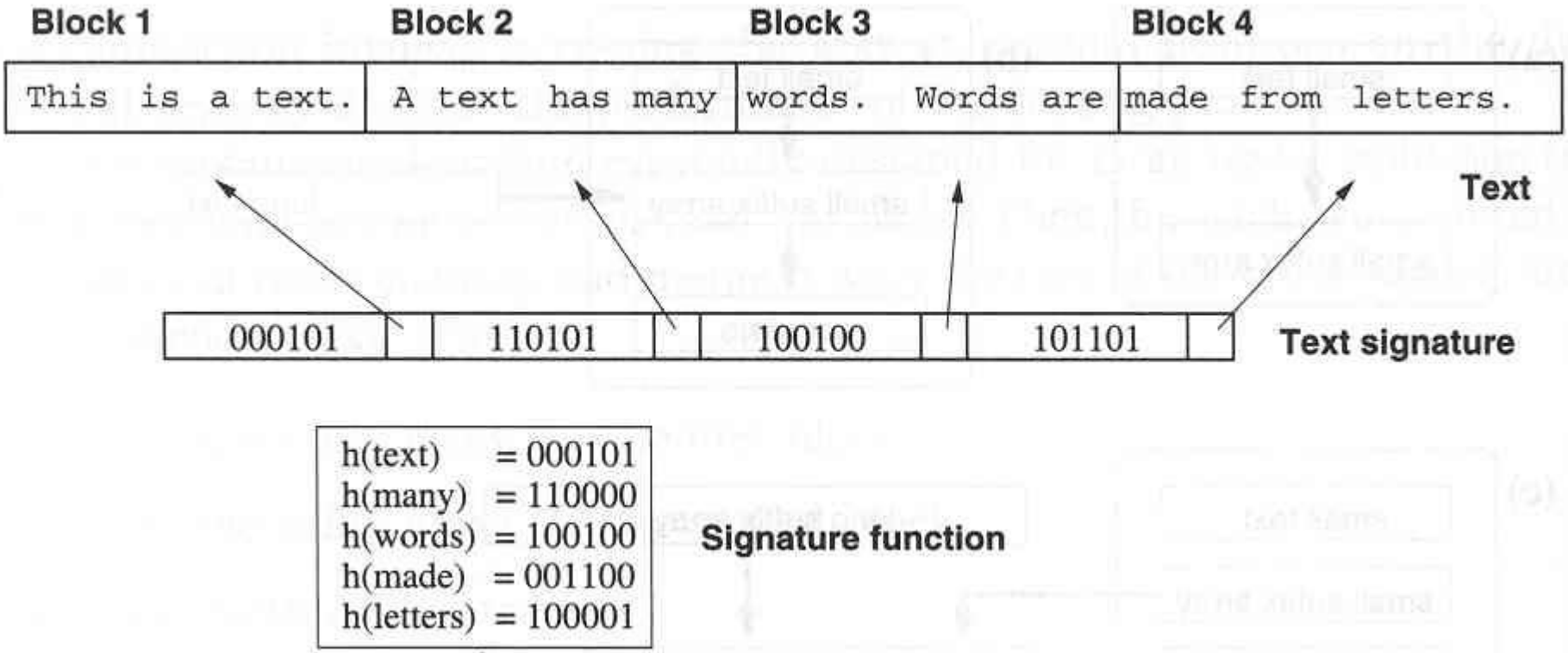
A supra-index over our suffix array. One out of three entries are sampled, keeping their first four characters.

Suffix Trees and Suffix Arrays



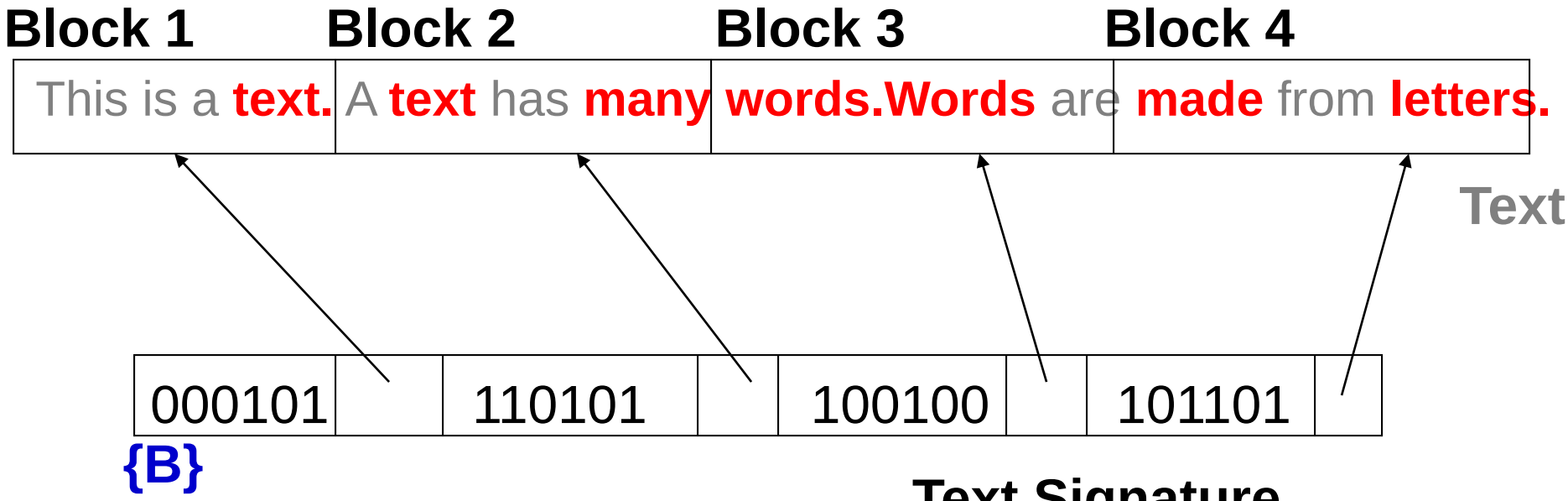
Relationship betweem inverted list and suffix array with vocabulary supra-index.

Signature Files



A signature file for sample text cut into blocks.

A Signature File

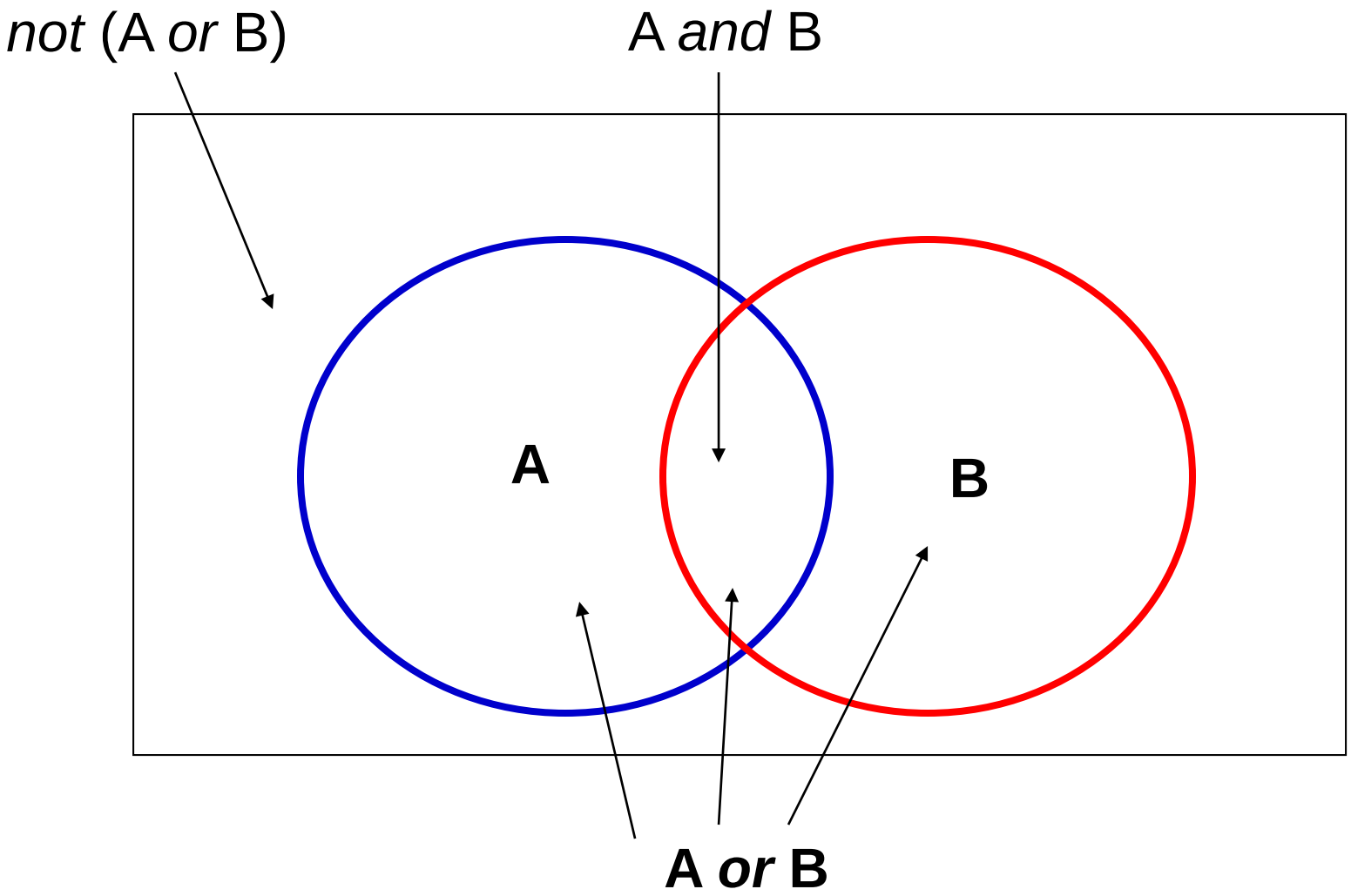


W

h(text)	=000101
h(many)	=110000
h(words)	=100100
h(made)	=001100
h(letters)	=100001

if $W \& B_i = W$ then W in B_i

Boolean Diagram



Evaluating a Boolean Query

Examples: abacus *and* actor

Postings for abacus

3
19
22

Postings for actor

2
19
29

To evaluate the *and* operator, merge the two inverted lists with a logical *AND* operation

Document 19 is the only document that contains both terms, "abacus" and "actor".

Adjacent and Near Operators

abacus *adj* actor

Terms abacus and actor are adjacent to each other as in the string

"abacus actor"

abacus *near 4* actor

Terms abacus and actor are near to each other as in the string

"the actor has an abacus"

Some systems support other operators, such as *with* (two terms in the same sentence) or *same* (two terms in the same paragraph).

Evaluating an Adjacency Operation

Examples: abacus *adj* actor

Postings for abacus

3	94
19	7
19	212
22	56

Postings for actor

2	66
19	213
29	45

Document 19, locations 212 and 213, is the only occurrence of the terms "abacus" and "actor" adjacent.

Evaluation of Boolean Operators

Precedence of operators must be defined:

adj, *near* high

and, *not* ↕

or low

Example

A *and* B *or* C *and* B

is evaluated as

(A *and* B) *or* (C *and* B)

Sequential Search

- Brute Force
- Boyer Moore
- Knuth-Morris-Pratt

Brute Force

- All Cases
- Simplest
- Most Correct
- Standard for other algorithms evaluation

Brute Force Example

i=0

B	r	u	t	e		f	o	r	c	e
f	o	r								

Brute Force Example

i=1

B	r	u	t	e		f	o	r	c	e
	f	o	r							

Brute Force Example

i=2

B	r	u	t	e		f	o	r	c	e
		f	o	r						

Brute Force Example

i=3

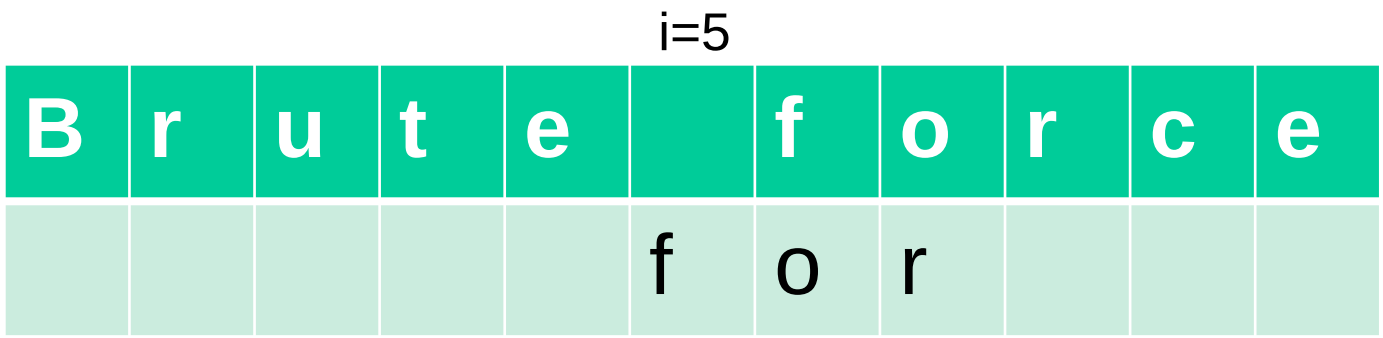
B	r	u	t	e		f	o	r	c	e
			f	o	r					

Brute Force Example

i=4

B	r	u	t	e		f	o	r	c	e
				f	o	r				

Brute Force Example



Brute Force Example

i=6

B	r	u	t	e		f	o	r	c	e
						f	o	r		

Postion Return = 6

ข้อดีของ Brute Force

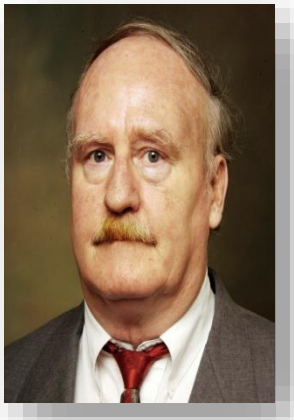
- Simple (Design,Programming)
- Correct Result
- Standard Evaluation

ข้อเสียของ Brute Force

- Small System
- Resouce Using

Boyer Moore

ประวัติของ BOYER และ MOORE



Robert S. Boyer

- นักวิทยาศาสตร์คอมพิวเตอร์ / คณิตศาสตร์ / ปรัชญา
- เกิดที่ประเทศ สหรัฐอเมริกา
- เริ่มคิดค้น Boyer–Moore algorithm ร่วมกับ J Strother Moore ในปี 1977

J Strother Moore

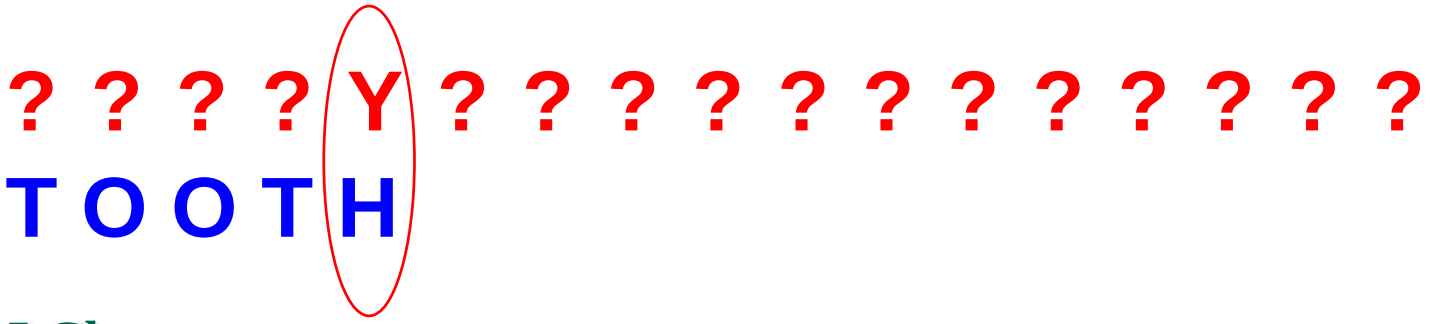
- นักวิทยาศาสตร์คอมพิวเตอร์
- เกิดที่ประเทศ สหรัฐอเมริกา

Boyer Moore

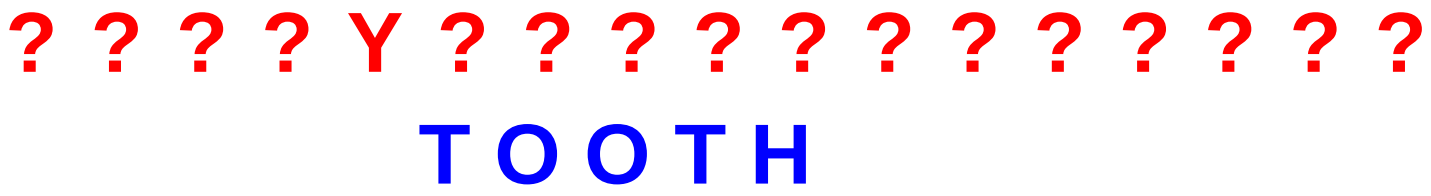
- Two stage algorithm
- Stage 1
  - Build a table that contrain the length to shift when a bad match occurs.
- Stage 2
  - Search string from last character to the first
  - The bad match table is used to skip character

Boyer Moore Example

Rule 1

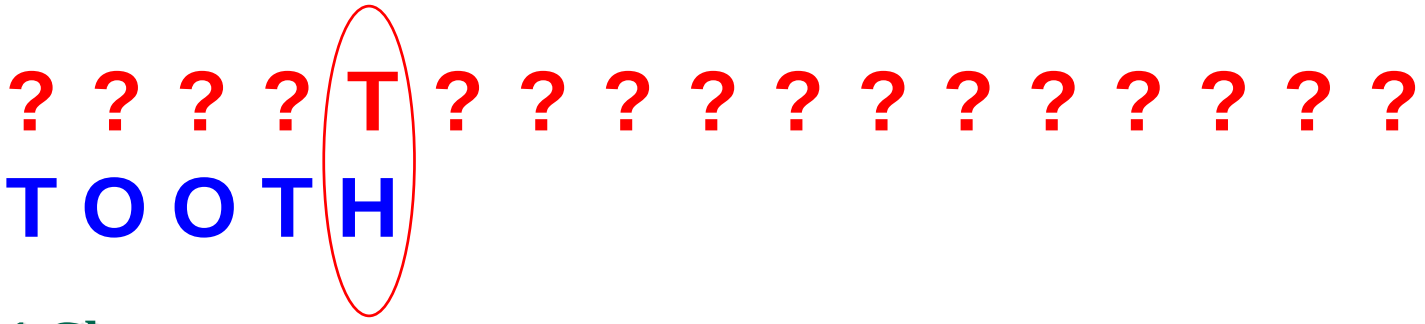


5 Chars move

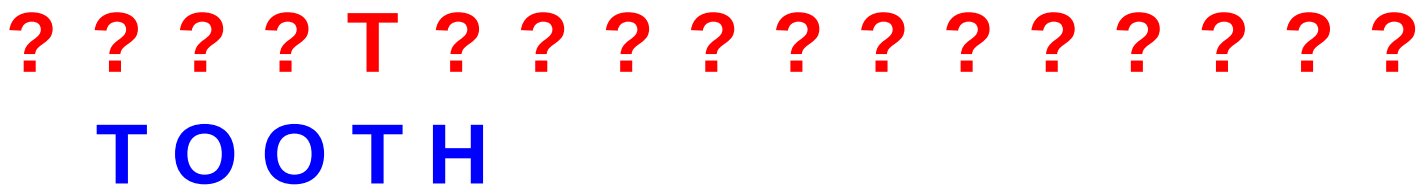


Boyer Moore Example

Rule 2

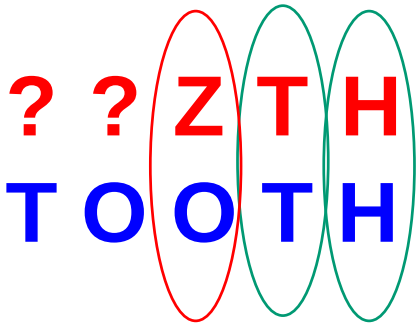


1 Chars move



Boyer Moore Example

Rule 3



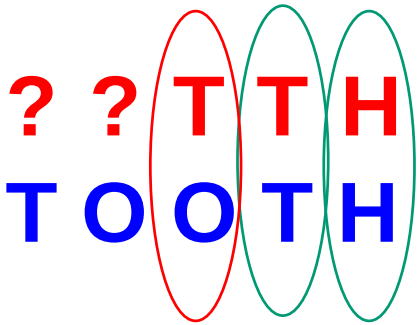
5 Chars move

? ? Z T H ? ? ? ? ? ? ? ? ? ? ? ?

T O O T H

Boyer Moore Example

Rule 4



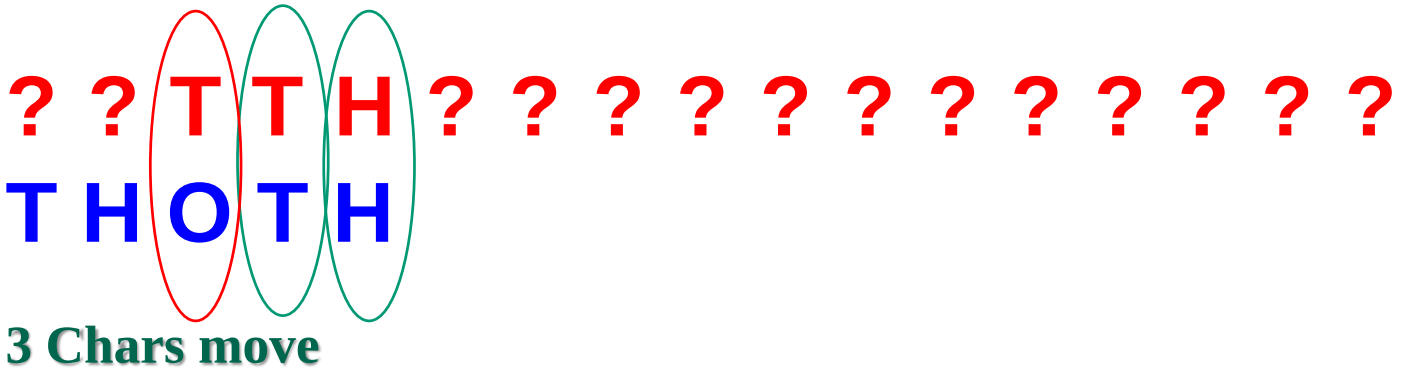
5 Chars move

? ? T T H ? ? ? ? ? ? ? ? ? ? ? ?

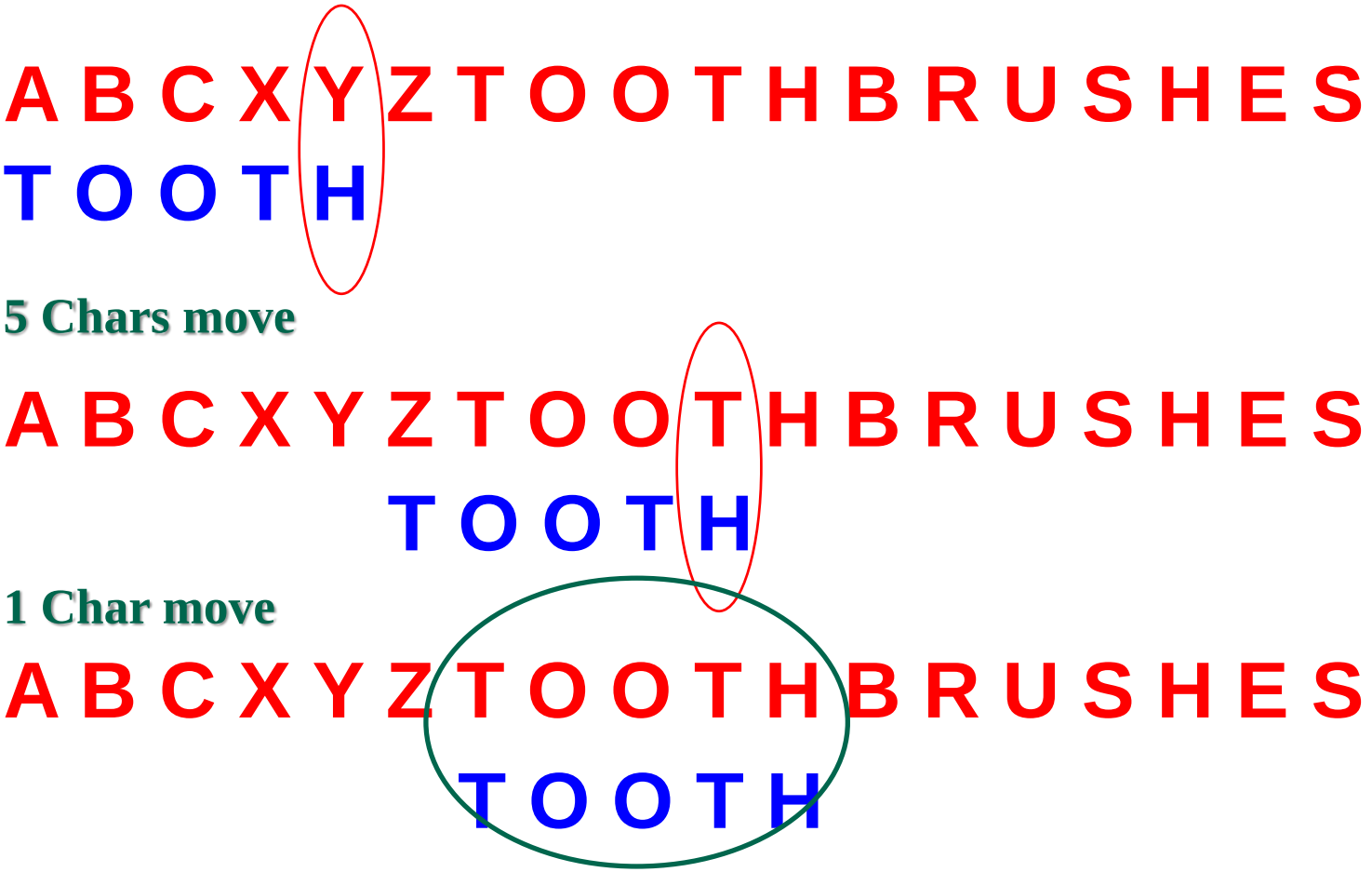
T O O T H

Boyer Moore Example

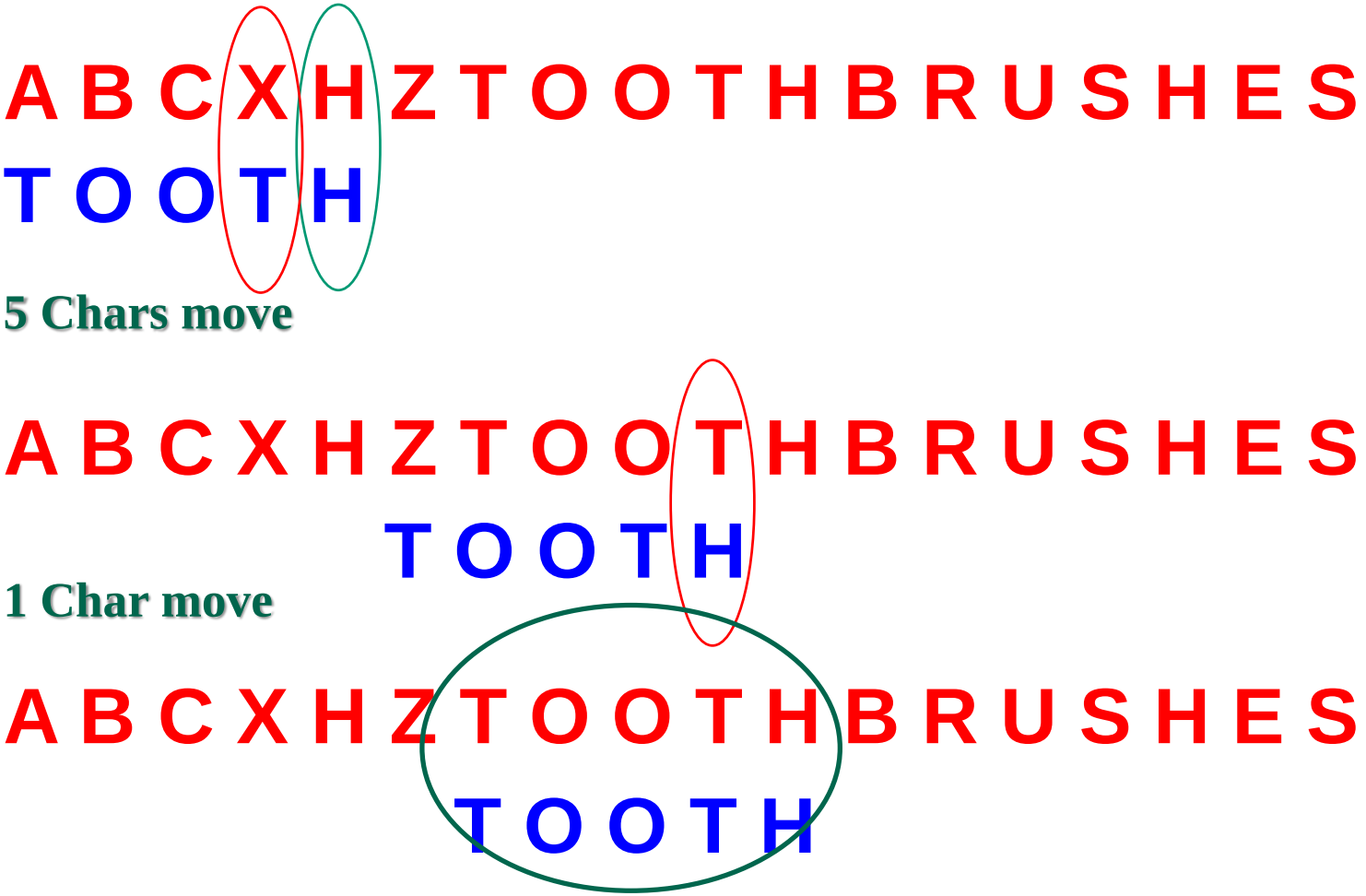
Rule 5



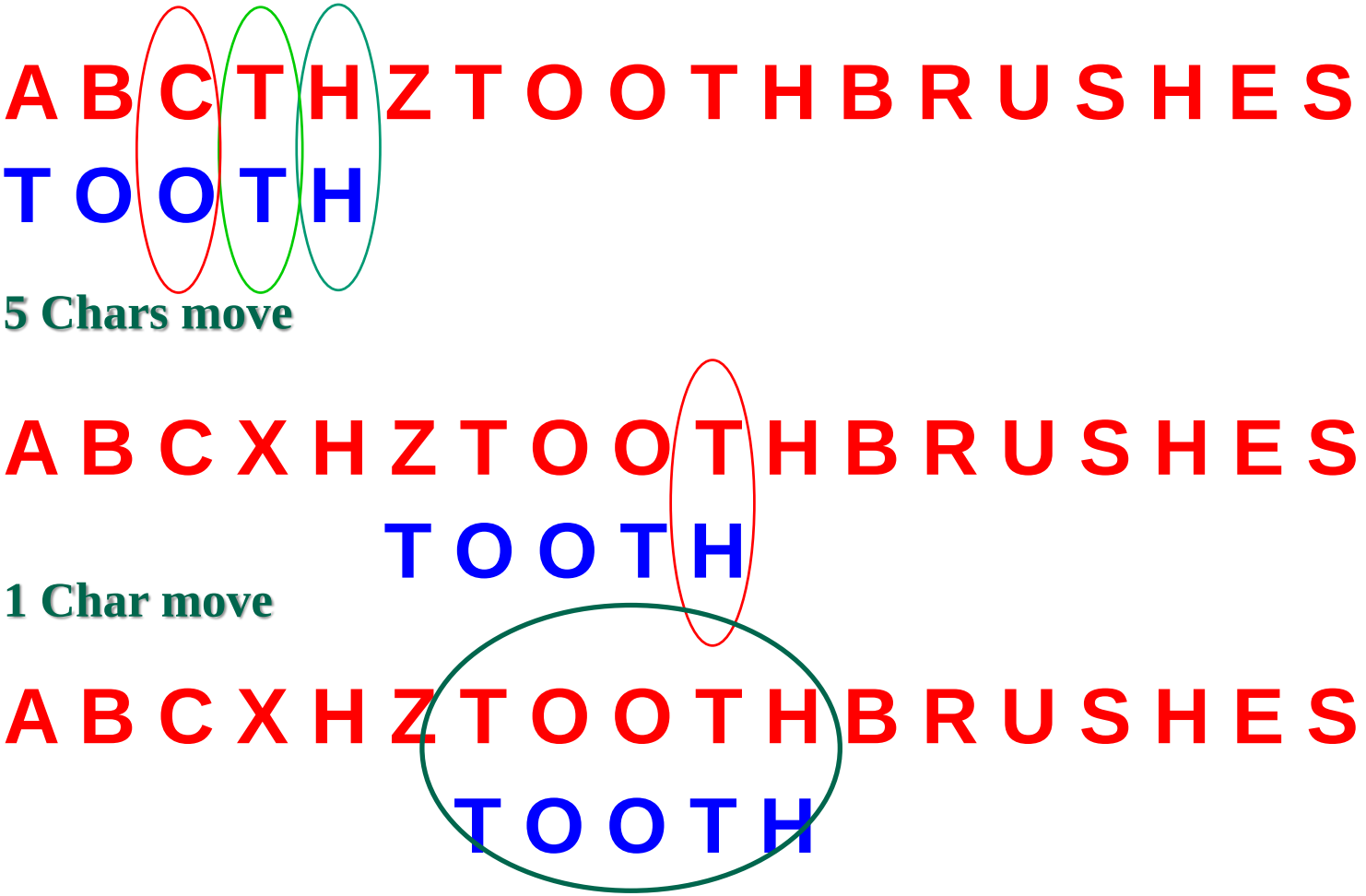
Boyer Moore Example



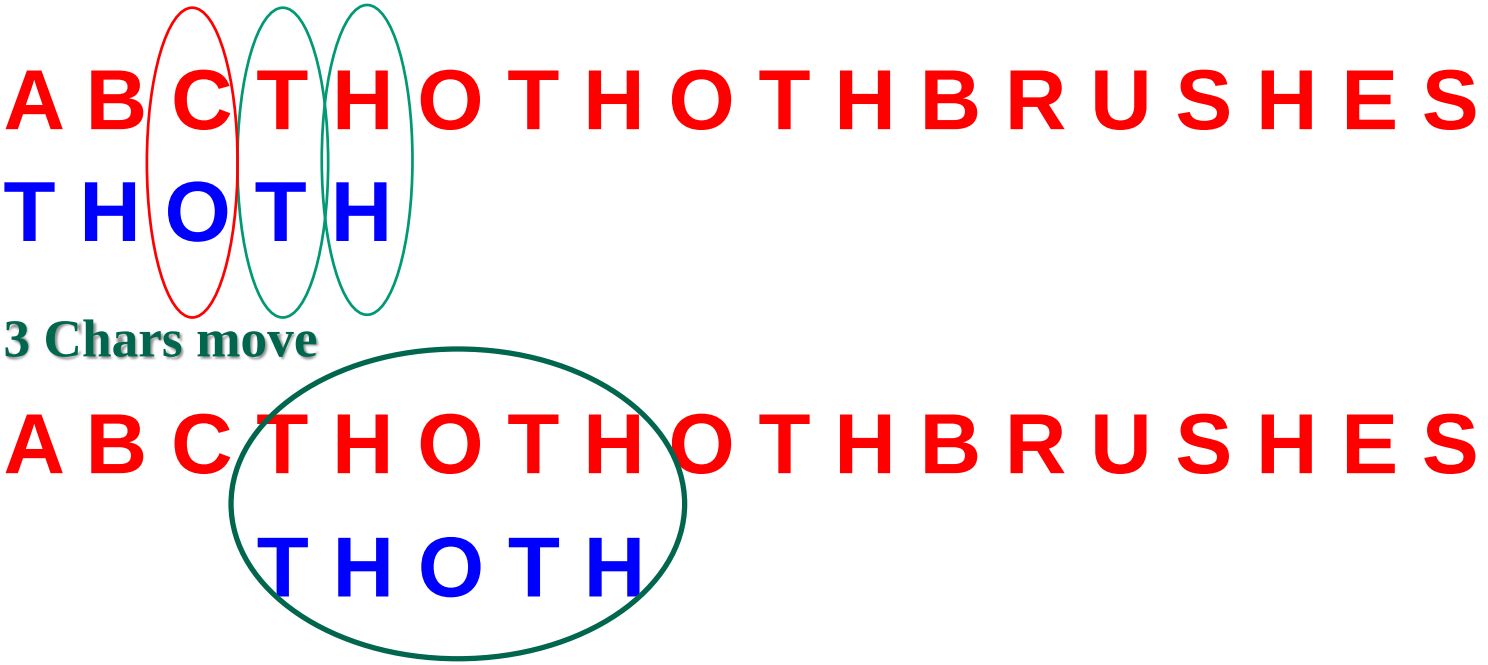
Boyer Moore Example



Boyer Moore Example

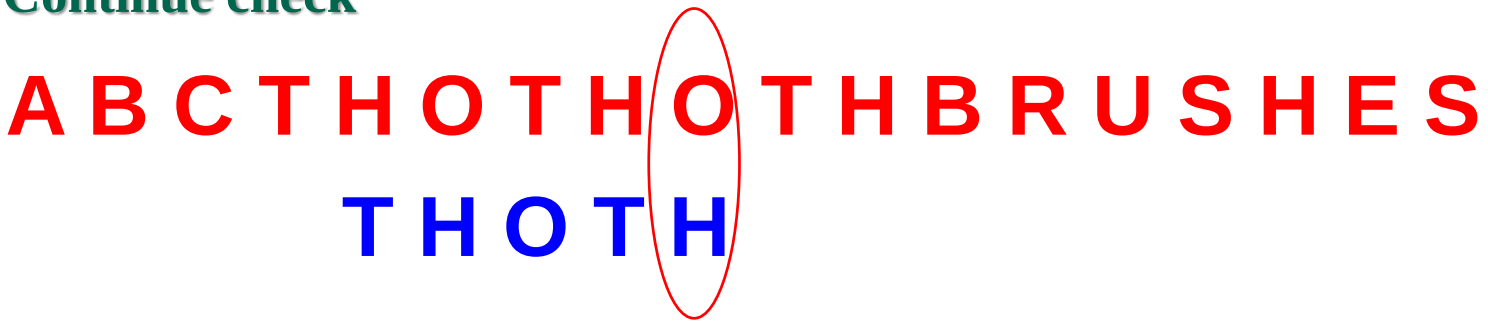


Boyer Moore Example

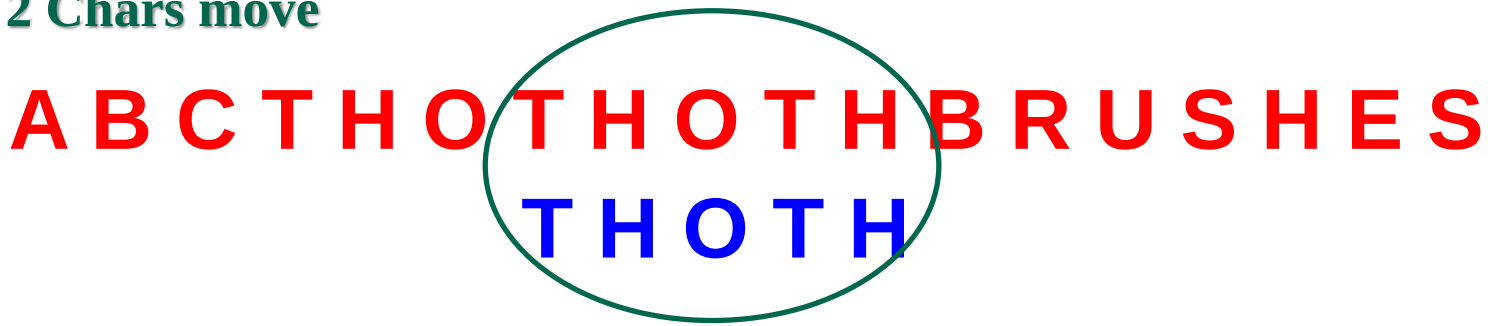


Boyer Moore Example

Continue check



2 Chars move



Boyer Moore Example

Text : **TRUSTHARDTOOTHBRUSHES**

Search Text : **TOOTH**

Boyer Moore Example

- Stage 1
  - Build a table that contrain the length to shift when a bad match occurs.

Boyer Moore Example

Value = length – index – 1

TOOTH Length = 5

Letter	**T**	**O**	**H**	*****
Value				

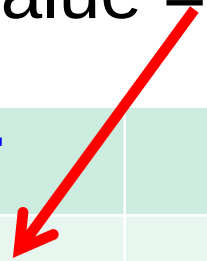
Boyer Moore Example

Index= 0

Value = length – index – 1

TOOTH

Value = 5-0-1=4



Letter	**T**	**O**	**H**	*****
Value	4			

Boyer Moore Example

Index= 1

Value = length – index – 1

TOOTH

Value = 5-1-1=3

Letter	**T**	**O**	**H**	*****
Value	4	3		

Boyer Moore Example

Index= 2 Value = length – index – 1

TOOTH Value = 5-2-1=2

Letter	**T**	**O**	**H**	*****
Value	4	3 2		

Boyer Moore Example

Index= 3 Value = length – index – 1

TOOTH Value = 5-3-1=1

Letter	**T**	**O**	**H**	*****
Value	1	2		

Boyer Moore Example

Last Char Value = length – index – 1

TOOTH

ยังไม่เคยมี H มาก่อน

Value = Length=5

Letter	**T**	**O**	**H**	*****
Value	1	2	5	5

Other char Value = Length=5

Boyer Moore Example

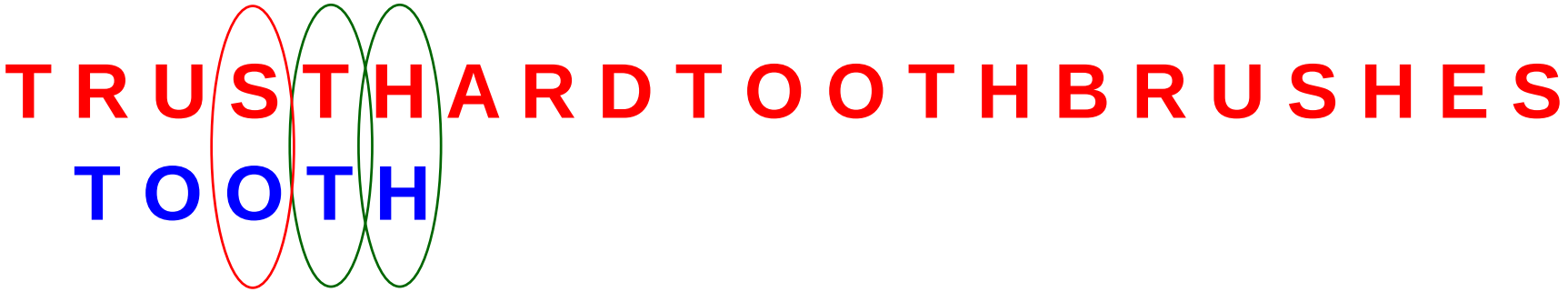
Letter	T	O	H	*
Value	1	2	5	5

T R U S T H A R D T O O T H B R U S H E S

T O O T H

Boyer Moore Example

Letter	T	O	H	*
Value	1	2	5	5



Boyer Moore Example

Letter	T	O	H	*
Value	1	2	5	5

T R U S T H A R D T O O T H B R U S H E S

T O O T H

Boyer Moore Example

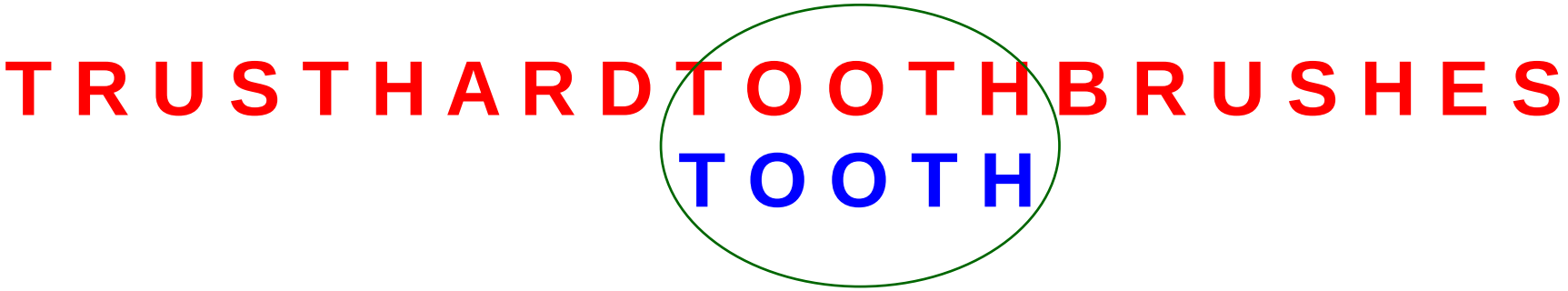
Letter	T	O	H	*
Value	1	2	5	5

T R U S T H A R D T O O T H B R U S H E S

T O O T H

Boyer Moore Example

Letter	T	O	H	*
Value	1	2	5	5



Boyer Moore Example

– Text : **TRUSZHAOTHTHOTHBRUSHES**

Search Text : **THOTH**

Boyer Moore Example

THOTH

Length = 5

Value = length – index – 1

T = 5-3-1=1

H = 5-1-1=3 ← ไม่นำ **H** ตัวสุดท้ายมาคำนวณ

O = 5-2-1=2

Letter	**T**	**H**	**O**	*****
Value	**1**	**3**	**2**	**5**

Boyer Moore Example

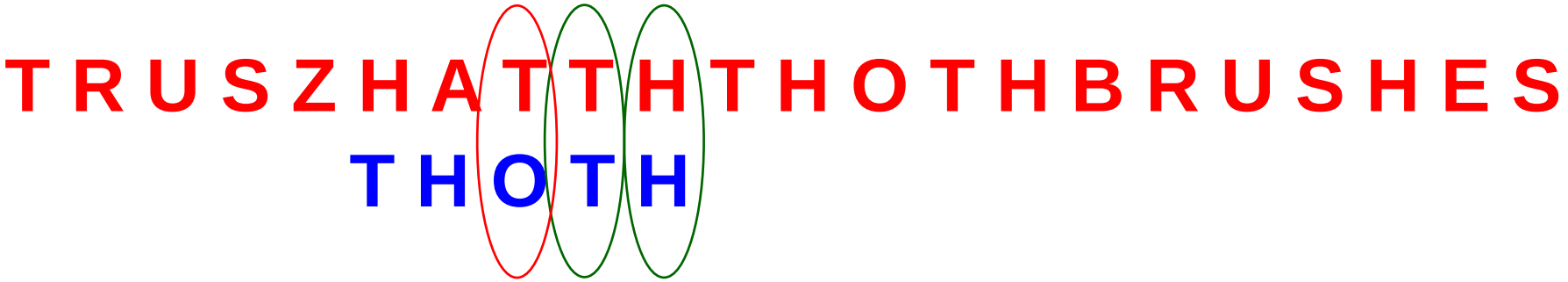
Letter	T	H	O	*
Value	1	3	2	5

T R U S Z H A T T H T H O T H B R U S H E S

T H O T H

Boyer Moore Example

Letter	T	H	O	*
Value	1	3	2	5



Boyer Moore Example

Letter	T	H	O	*
Value	1	3	2	5

T R U S Z H A T T H T H O T H B R U S H E S

T H O T H

Boyer Moore Example

Letter	T	H	O	*
Value	1	3	2	5

T R U S Z H A T T H T H O T H B R U S H E S

T H O T H

Boyer Moore Example

สรุป

1. เทียบตัวอักษรจากท้ายไปต้น
2. หากไม่ตรงกัน

2.1 ไม่ตรงกันที่ตัวอักษรท้ายสุดของข้อมูล หากตัวอักษรท้ายสุดของข้อมูลไม่มีในคำค้น เลื่อนเท่ากับความยาวของคำค้น

2.2 ไม่ตรงกันที่ตัวอักษรท้ายสุดของข้อมูล หากตัวอักษรท้ายสุดของข้อมูลมีในคำค้น เลื่อนตามตารางที่คำนวณไว้ (ใช้ตัวอักษรท้ายสุดของข้อมูล มาพิจารณา)

2.3 ไม่ตรงกันที่ตัวอักษรอื่นของข้อมูล หากตัวอักษรสุดท้ายของคำค้นปรากฏครั้งเดียวในคำค้น เช่น TOOTH เลื่อนเท่ากับความยาวของคำค้น

2.4 ไม่ตรงกันที่ตัวอักษรอื่นของข้อมูล หากตัวอักษรสุดท้ายของคำค้นปรากฏหลายครั้งในคำค้นเช่น THOTH เลื่อนตามตารางที่คำนวณไว้ (ใช้ตัวอักษรท้ายสุดของข้อมูล มาพิจารณา)

Knuth-Morris-Pratt

Knuth-Morris-Pratt

อัลกอริธึมนี้ ได้เริ่มคิดขึ้นในปี 1974 โดย Donald Knuth และ Vaughan Pratt และ James H. Morris และทั้งสามคนได้ตีพิมพ์ผลงานร่วมกันในปี 1977

KMP เป็นอัลกอริธึมที่ใช้ในการค้นหา คำภายในข้อความ โดยการสังเกตว่า เมื่อไรที่พบตัวอักษรที่ไม่ตรงกับคำที่ต้องการ จะมีกระบวนการในการตัดสินใจค้นหาตัวอักษรต่อไปที่ไหน และ จะไม่ตรวจสอบตัวอักษรที่เคยตรวจสอบไปแล้ว

Knuth-Morris-Pratt

- Two stage algorithm
- Stage 1
  - Build a **prefix table** used to shift bad match occurs.
- Stage 2
  - Search string from first character to the Last
  - The prefix table table is used to shift character

Knuth-Morris-Pratt Example

Text : **ACAT ACGACACAGT**

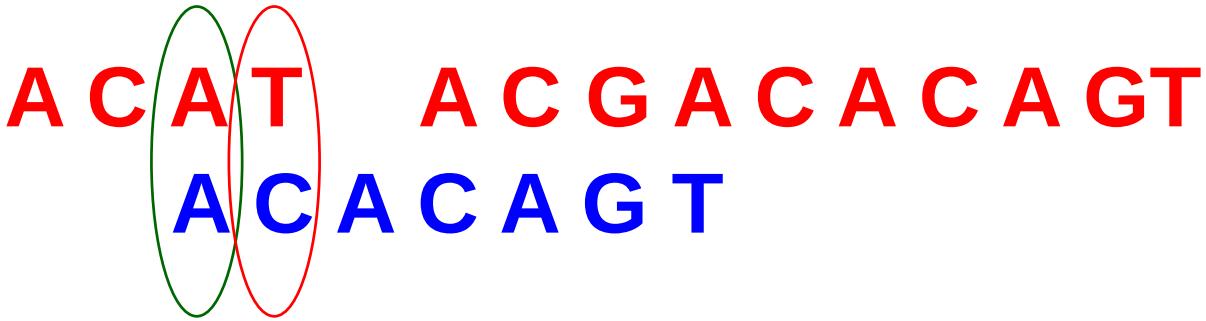
Search Text : **ACACAGT**

Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0

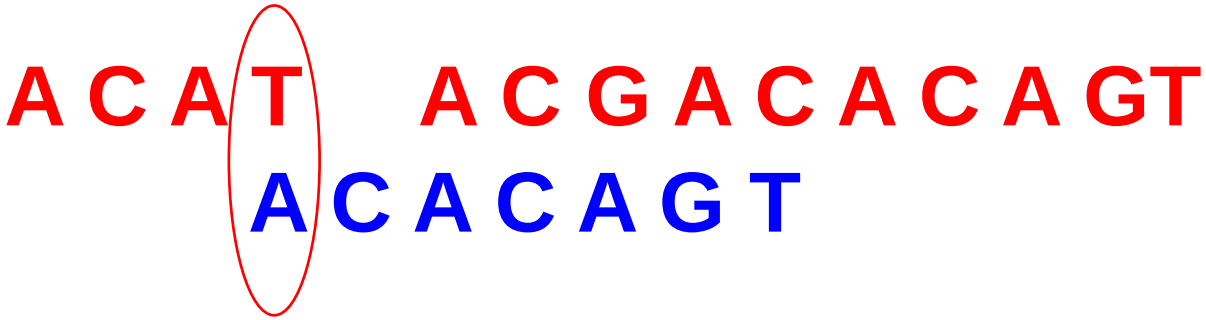
Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0



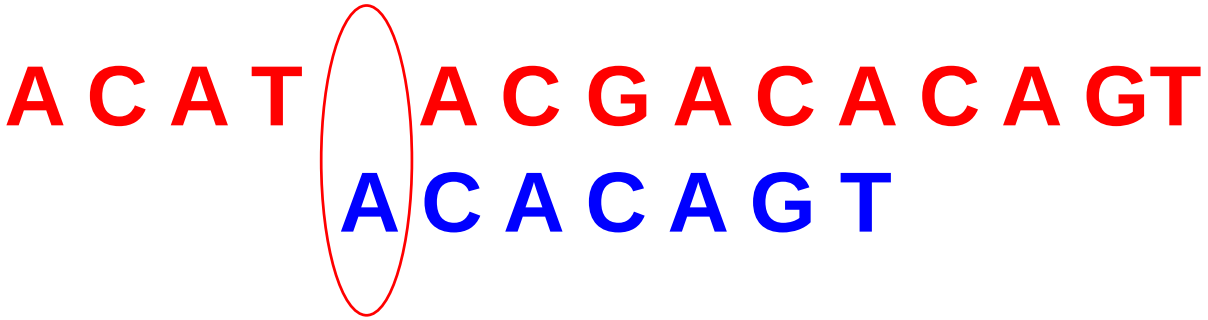
Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0



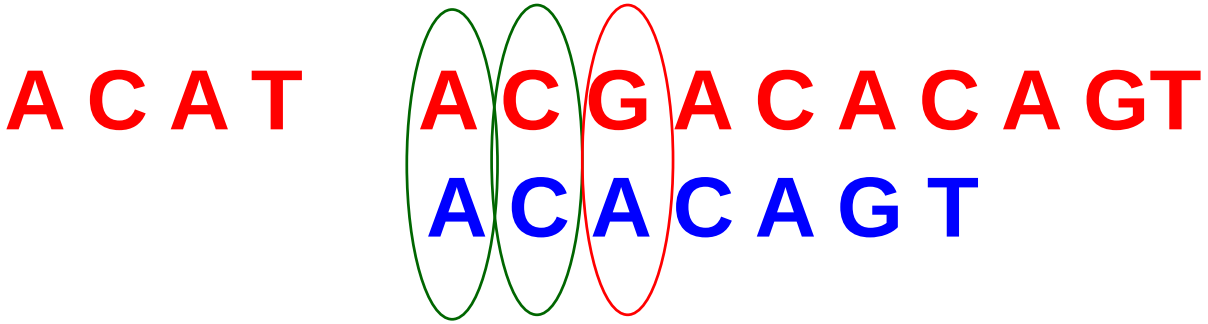
Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0



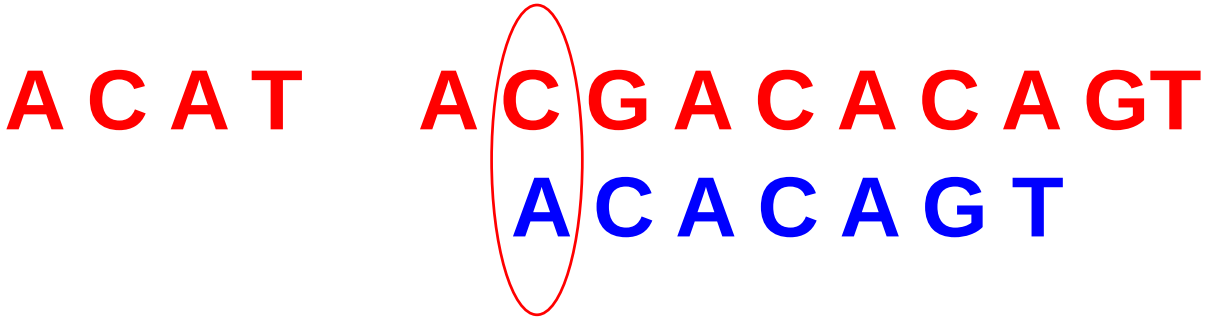
Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0



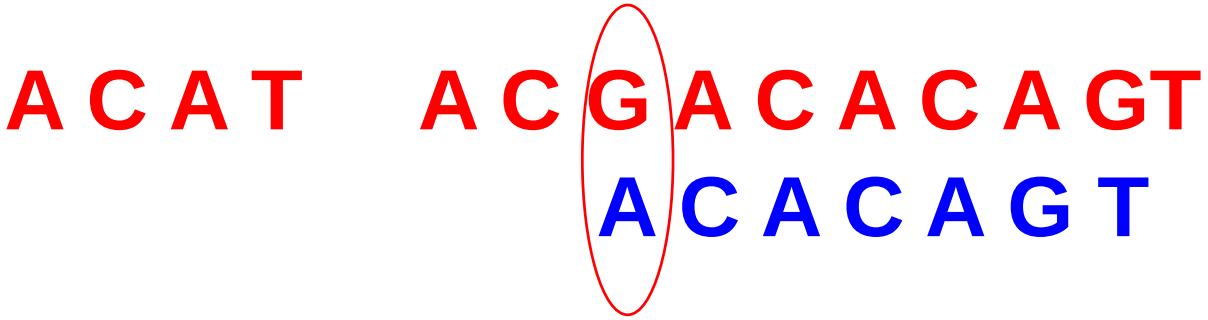
Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0



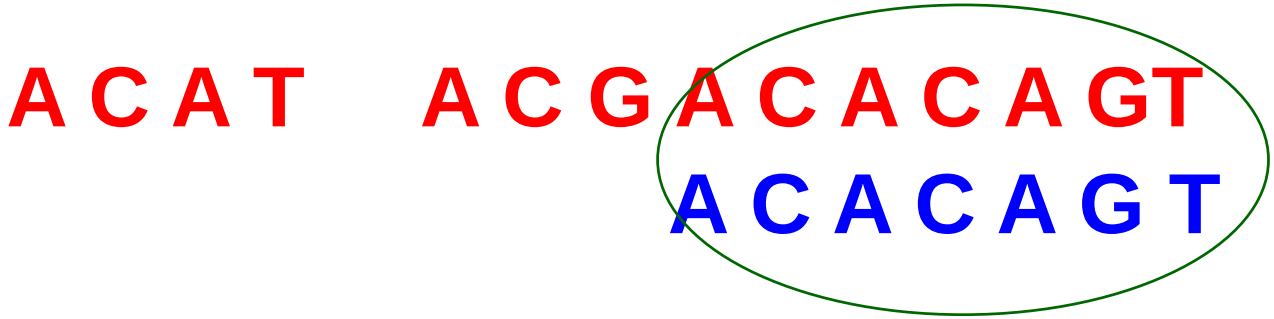
Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0



Knuth-Morris-Pratt Example

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0



Knuth-Morris-Pratt

Prefix Table Creation

No prefix
No suffix

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0						

Knuth-Morris-Pratt

Prefix Table Creation

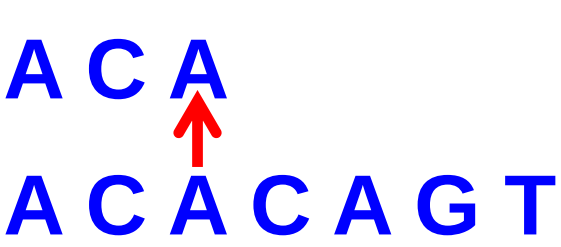
Prefix : **A**
Suffix : **C**

No duplication

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0					

Knuth-Morris-Pratt

Prefix Table Creation



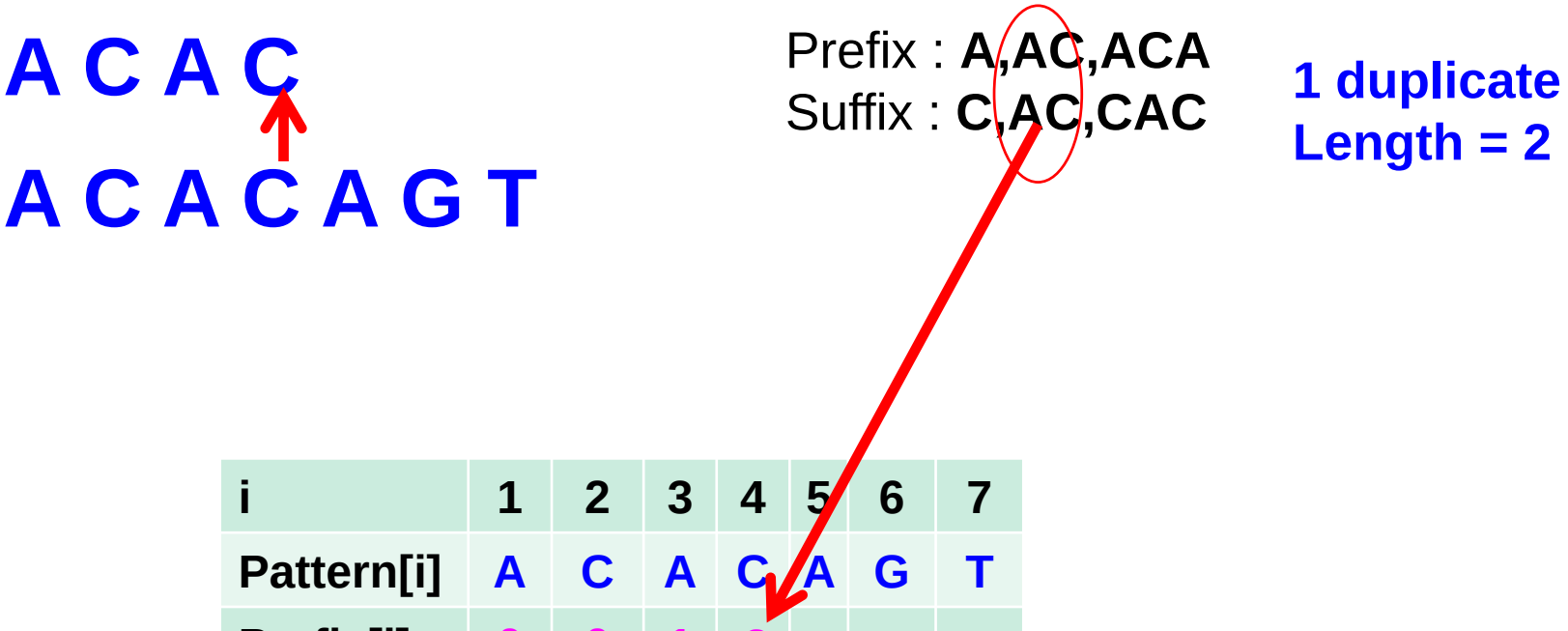
Prefix : **A,AC**
Suffix : **A,CA**

1 duplicate
Length = 1

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1				

Knuth-Morris-Pratt

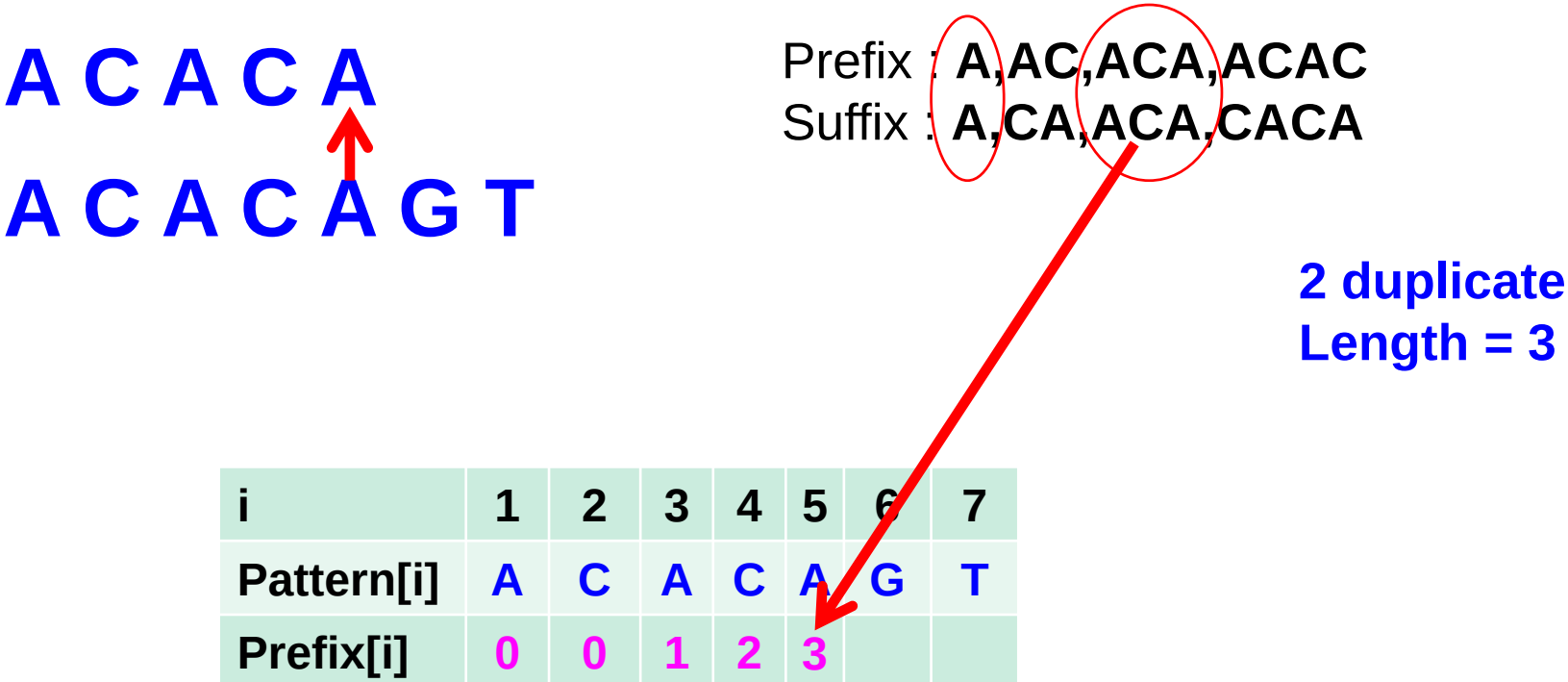
Prefix Table Creation



i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2			

Knuth-Morris-Pratt

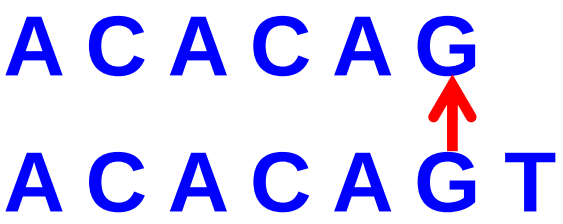
Prefix Table Creation



i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3		

Knuth-Morris-Pratt

Prefix Table Creation



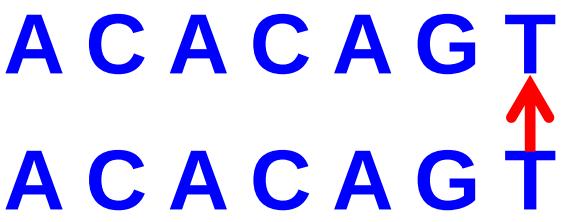
Prefix : **A,AC,ACA,ACAC,ACACA**
Suffix : **G,AG,CAG,ACAG,CACAG**

No duplication

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	

Knuth-Morris-Pratt

Prefix Table Creation



Prefix : **A,AC,ACA,ACAC,ACACA,ACACAG**
Suffix : **T,GT,AGT,CAGT,ACAGT,CACAGT**

No duplication

i	1	2	3	4	5	6	7
Pattern[i]	A	C	A	C	A	G	T
Prefix[i]	0	0	1	2	3	0	0