Fall 2019

Data Structures Lecture 3

Fang Yu

Software Security Lab.
Dept. Management Information Systems,
National Chengchi University

HWs Review – What you should have learned?

- Calculate your BMI
  - Java Class Library
- Calculator/Matrix manipulation
  - Input format and operation
- Generic Geometric Progression
  - Inheritance
  - Generics
  - Exceptions

Project Announcement

- A Team Project: 30%
  - 3-5 students as a team
  - Send the team list (name and contact) to your TAs before the end of this week
  - Develop your application using Eclipse with SVN
    - TAs will help you set up SVN
    - You will get extra points for having constant code update

Lets Beat Google!

- Goal: On the top of a giant's shoulder, achieve advanced information searching with your expertise!
- Select a topic that you/your team members have interests.
- Make sure your search engine gets better results than a general search engine such as Google.
- Stage 0 (HW3): Keyword Counting
  - Given an URL and a keyword
  - Return how many times the keyword appears in the contents of the URL

Lets Beat Google!

- Stage 1 (30%+): Page Ranking
  - Given a set of keywords and URLs
  - Rank the URLs based on their score
  - Define a score formula based on keyword appearances
  - For each URL (a web page), return its rank, score, and the count on appearance of each keyword

Lets Beat Google!

- Stage 2 (50%+) Site Ranking
  - Multiple level keyword search
  - Given a set of Web sites (URLs) and Keywords
  - Rank the Web sites with their keyword appearances (including **all its sub URLs)**
  - Define a score formula based on keyword appearances in the URL and all its sub URLs
  - For each URL (a web site), return its rank, score, and a tree structure for its sub URLs along with the number of appearance of each keyword in each node

Lets Beat Google!

- Stage 3 (70%+) Refine the rank of Google
  - Given a set of Keywords (No URLs)
  - Use **search engines** to find potential URLs
  - Apply the ranking on Stage 2 to these Web sites
- Stage 4 (80%+) Semantics Analysis
  - Derive **relative keywords** from the discovered Web sites
  - Iteratively do the same analysis on Stage 3
- Stage 5 (90%+) Publish Your Work Online
  - Build a web site/service for your searching engine
- Stage 6 (100%+) Export Your Work to App
  - Wrap your search engine as an iOS/android mobile application

Important Date

Each team needs to

1. Submit the project proposal (4-8 pages) on **Nov. 7**
2. Give a Demo on **Jan. 9**
3. Upload the source code before **Jan. 16**

Text Processing

Strings and Pattern matching

Text Processing

- Due to internet, social networks, web and mobile applications, a lot of documents and contents are online and public available

- Text processing becomes one of the dominant functions of computers

- HTML and XML
  - Primary text formats with added tags for multimedia content
  - Java Applet (embedded Java bytecode in the HTML)

Strings

- A string is a sequence of characters
- An alphabet Σ is the set of possible characters for a family of strings
- Example of alphabets:
  - ASCII
  - Unicode
  - {0, 1}
  - {A, C, G, T}

Strings

- Let P be a string of size m
- A substring $P[i .. j]$ of P is the subsequence of P consisting of the characters with ranks between i and j
- A prefix of P is a substring of the type $P[0 .. i]$
  - "Fan" is a prefix of "Fang Yu, NCCU"
- A suffix of P is a substring of the type $P[i .. m - 1]$
  - "CCU" is a suffix of "Fang Yu, NCCU"

Java String Class

String S;

- Immutable strings: operations simply return information about strings (no modification)

length()	Return the length of S
charAt(i)	Return the ith character
startsWith(Q)	True if Q is a prefix of S
endsWith(Q)	True is Q is a suffix of S
substring(i,j)	Return the substring S[i,j]
concat(Q)	Return S+Q
equals(Q)	True is Q is equal to S
indexOf(Q)	If Q is a substring of S, returns the index of the beginning of the first occurrence of Q in S

Java String Class

String a = "Hello World!";

Operation	Output
a.length()	
a.charAt(1)	
a.startsWith("Hell")	
a.endsWith("rld")	
a.substring(1,2)	
a.concat("rld")	
a.substring(1,2).equals("e")	
indexOf("rld")	

Java String Class

String a = “Hello World!”;

Operation	Output
a.length()	12
a.charAt(1)	e
a.startsWith(“Hell”)	true
a.endsWith(“rld”)	false
a.substring(1,2)	e
a.concat(“rld”)	Hello World!rld
a.substring(1,2).equals(“e”)	true
a.indexOf(“rld”)	8

Java StringBuffer Class

StringBuffer S;

- Mutable strings: operations modify the strings

append(Q)	Replace S with S+Q. Return S.
Insert(i,Q)	Insert Q in S starting at index i. Return S
reverse()	Reverse S. Return S.
setCharAt(i, ch)	Set the character at index i in S to ch
charAt(i)	Return the character at index i in S
toString()	Return a String version of S

Java StringBuffer Class

StringBuffer a = new StringBuffer();

Operation	**a**
a.append(“Hello World!”)	
a.reverse()	
a.reverse()	
a.insert(6,”Fang and the ”)	
a.setCharAt(4, ‘!’)	

Java StringBuffer Class

StringBuffer a = new StringBuffer();

Operation	a
a.append("Hello World!")	Hello World!
a.reverse()	!dlroW olleH
a.reverse()	Hello World!
a.insert(6,"Fang and the ")	Hello Fang and the World!
a.setCharAt(4, '!')	Hell! Fang and the World!

Pattern Matching

- Given a text string T of length n and a pattern string P of length m
- Find whether P is a substring of T
- If so, return the starting index in T of a substring matching P
- The implementation of T.indexOf(P)
- Applications:
  - Text editors, Search engines, Biological research

Brute-Force Pattern Matching

The idea:

- Compare the pattern ***P*** with the text ***T*** for each possible shift of ***P*** relative to ***T***, until
- either a match is found, or
- all placements of the pattern have been tried

```
Algorithm BruteForceMatch(T, P)
    Input text T of size n and pattern
        P of size m
    Output starting index of a
        substring of T equal to P or −1
        if no such substring exists
    for i ← 0 to n − m // test shift i of the pattern
        j ← 0
        while j < m ∧ T[i + j] = P[j]
            j ← j + 1
        if j = m
            return i //match at i
        else
            break while loop //mismatch
    return -1 //no match anywhere
```

Brute-Force Pattern Matching

- Time Complexity:
  - O(mn), where m is the length of T and n is the length of P
- A worst case example:
  - T = aaaaaaaaaaaaaab
  - P = aab
  - Need 39 comparisons to find a match
  - may occur in images and DNA sequences
  - unlikely in English text

Can we do better?

Here are two Heuristics.

1. Backward comparison

- Compare T and P from the end of P and move backward to the front of P

2. Shift as far as you can

- When there is a mismatch of P[j] and T[i]=c, if c does not appear in P, shift P[0] to T[i+1]

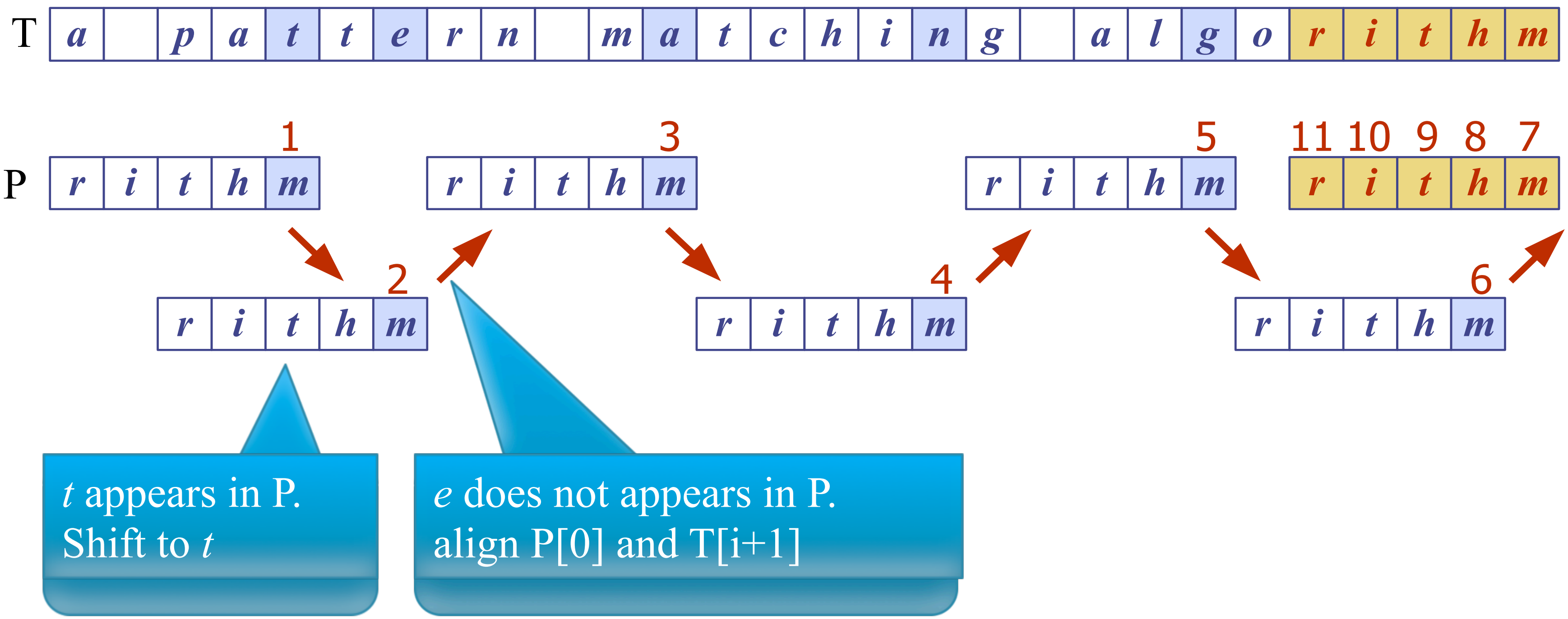
The Boyer-Moore Algorithm

- The Boyer-Moore's pattern matching algorithm is based on these two heuristics:

- The looking-glass heuristic: Compare ***P*** with a subsequence of ***T*** moving backwards

- The character-jump heuristic: When a mismatch occurs at ***T*** [***i***] = ***c***
  - If ***P*** contains ***c***, shift ***P*** to align the last occurrence of ***c*** in ***P*** with ***T***[***i***]
  - Else, shift ***P*** to align ***P***[0] with ***T***[***i*** + 1]

An Example


T
a p a t t e r n m a t c h i n g a l g o r i t h m
P
r i t h m
1 2 3 4 5 6 7 8 9 10 11
t appears in P.
Shift to t
e does not appears in P.
align P[0] and T[i+1]

Last Occurrence Function

- Boyer-Moore's algorithm preprocesses the pattern P and the alphabet Σ to build the last-occurrence function L mapping Σ to integers
- $L(c)$ is defined as (c is a character)
  - the largest index i such that $P[i] = c$ or
  - -1 if no such index exists
- Example:
  - $\Sigma = \{a, b, c, d\}$
  - $P = abacab$

c	a	b	c	d
$L(c)$	4	5	3	-1

Last Occurrence Function

- The last-occurrence function can be represented by an array indexed by the numeric codes of the characters
- The last-occurrence function can be computed in time $O(m + s)$, where m is the size of P and s is the size of Σ

The Boyer-Moore Algorithm

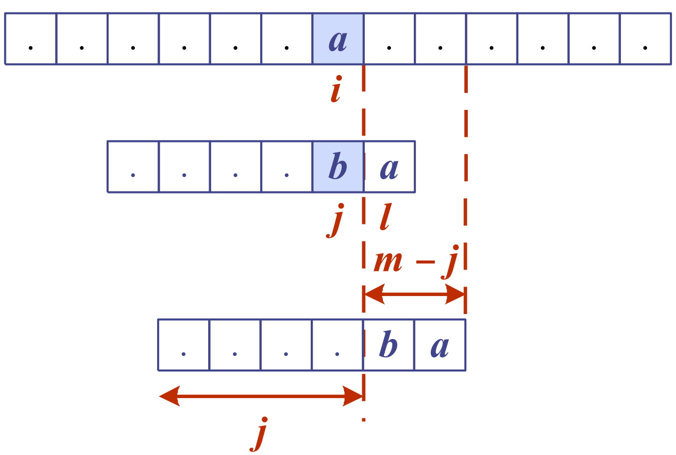
```
Algorithm BoyerMooreMatch(T, P, Σ)
    L ← lastOccurenceFunction(P, Σ)
    i ← m − 1 //backward
    j ← m − 1
    repeat
        if T[i] = P[j]
            if j = 0
                return i  // match at i
            else
                i ← i − 1
                j ← j − 1
        else
            // character-jump
            l ← L[T[i]]
            i ← i + m – min(j, 1 + l)
            j ← m − 1
    until i > n − 1
    return −1 { no match }
```

How to shift i?

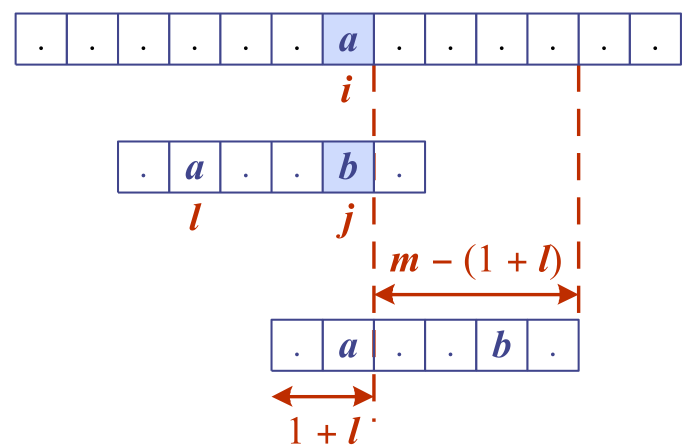
How to shift i after mismatching characters?

- $\boldsymbol{i} \leftarrow \boldsymbol{i} + \boldsymbol{m} - \min(\boldsymbol{j}, 1 + \boldsymbol{l})$
- Don't shift back!

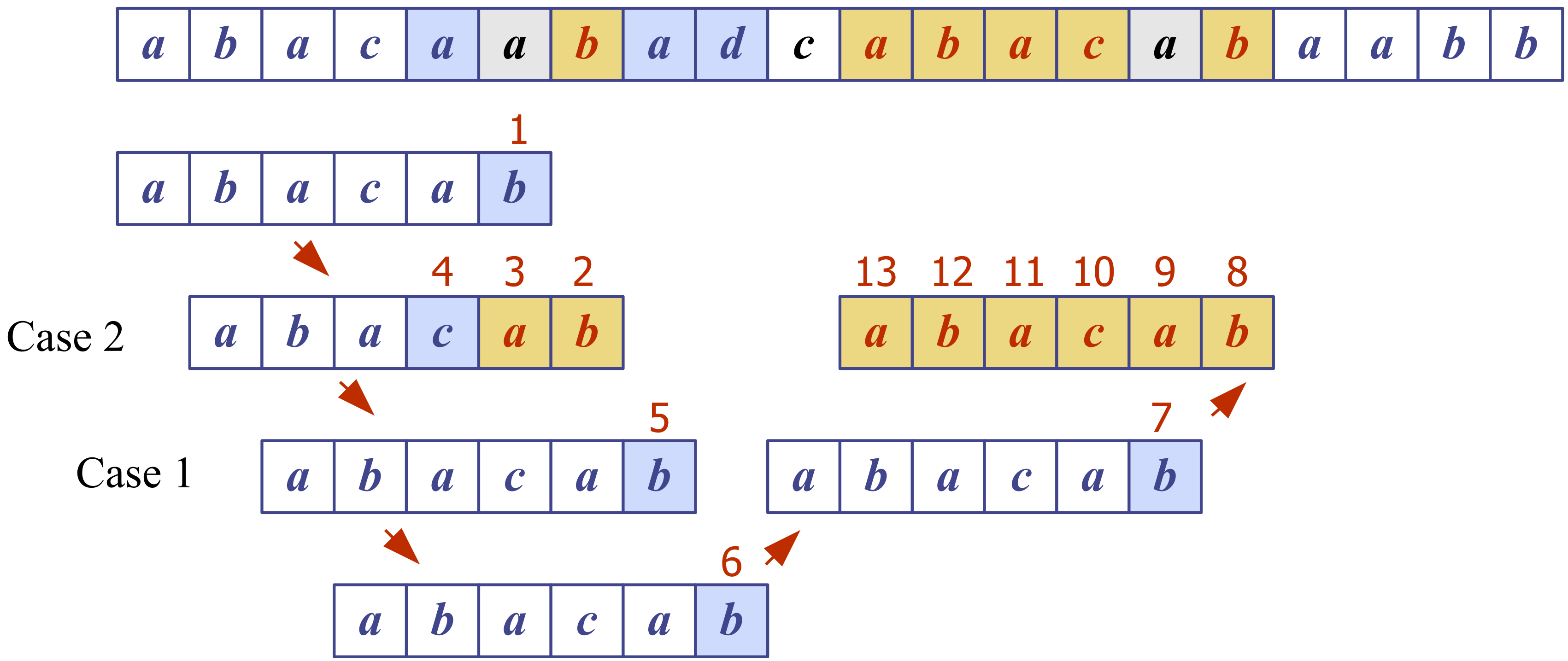
Case 1: $\boldsymbol{j} \leq 1 + \boldsymbol{l}$ ***(a appears after b)***



Case 2: $1 + \boldsymbol{l} \leq \boldsymbol{j}$ ***(a appears before b, jump!)***



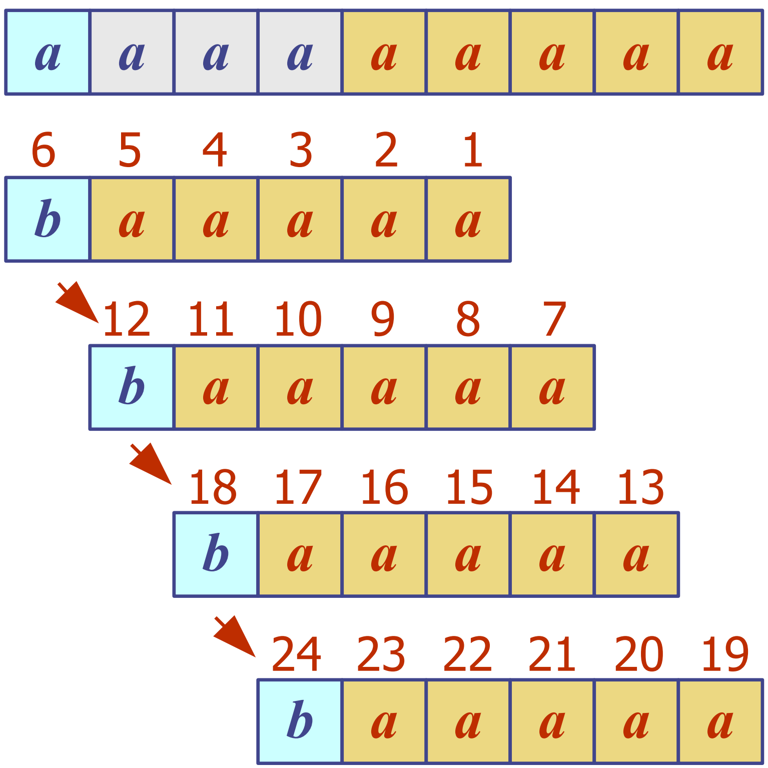
Another Example



Is it a better algorithm?

- Boyer-Moore's algorithm runs in time $\boldsymbol{O}(\boldsymbol{nm} + \boldsymbol{s})$
- An example of the worst case:
  - $\boldsymbol{T = aaa \ldots a}$
  - $\boldsymbol{P = baaa}$
- The worst case may occur in images and DNA sequences but it is unlikely happened in English text
- It has been shown that in practice Boyer-Moore's algorithm is significantly faster than the brute-force algorithm on English text

The Worst-case Example



HW3 (Due on 10/17)

Count A Keyword in a Web Page!

- Get a URL and a keyword from user inputs
- Return how many times the keyword appears in the contents of the URL
- For example:
  - Enter URL: http://soslab.nccu.edu.tw
  - Enter Keyword: Fang
  - Output: Fang appears X times

Hints

Count A Keyword in a Web Page!

- Implement indexOf() with Boyer-Moore’s algorithm
- Use looking-glass and character-jump heuristics

Coming up…

- We will start to discuss fundamental data structures such as arrays and linked lists on October 17 and continue the discussion on queues, stacks, trees, and heaps in the following weeks.
- Read TB Chapter 3