Announcement
A Make Up Course on Sep. 30

- We will have a make-up course on Sep. 30.
  - 9-12am (Mandarin) and 1-4pm (English)
  - Room 312, College of Commerce.
- We will not have courses on Oct. 5 and Oct. 12.
- Lab on Oct. 2 for HW3 (Submit your team members)
- Lab on Oct. 16 for HW4 (and tutorial on SVN)
HWs Review – What you should have learned?

- Calculate your BMI
  - Java Class Library

- 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
Let's 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. 2
2. Give a Demo on Jan. 11
3. Upload the source code before Jan. 18
Text Processing

Strings and Pattern matching
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 $\Sigma$ 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)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length()</td>
<td>Return the length of S</td>
</tr>
<tr>
<td>charAt(i)</td>
<td>Return the i'th character</td>
</tr>
<tr>
<td>startsWith(Q)</td>
<td>True if Q is a prefix of S</td>
</tr>
<tr>
<td>endsWith(Q)</td>
<td>True if Q is a suffix of S</td>
</tr>
<tr>
<td>substring(i,j)</td>
<td>Return the substring S[i,j]</td>
</tr>
<tr>
<td>concat(Q)</td>
<td>Return S+Q</td>
</tr>
<tr>
<td>equals(Q)</td>
<td>True if Q is equal to S</td>
</tr>
<tr>
<td>indexOf(Q)</td>
<td>If Q is a substring of S, returns the index of the beginning of the first occurrence of Q in S</td>
</tr>
</tbody>
</table>
Java String Class

String a = “Hello World!”;

<table>
<thead>
<tr>
<th>Operation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.length()</td>
<td></td>
</tr>
<tr>
<td>a.charAt(1)</td>
<td></td>
</tr>
<tr>
<td>a.startsWith(“Hell”)</td>
<td></td>
</tr>
<tr>
<td>a.endsWith(“rld”)</td>
<td></td>
</tr>
<tr>
<td>a.substring(1,2)</td>
<td></td>
</tr>
<tr>
<td>a.concat(“rld”)</td>
<td></td>
</tr>
<tr>
<td>a.substring(1,2).equals(“e”)</td>
<td></td>
</tr>
<tr>
<td>indexOf(“rld”)</td>
<td></td>
</tr>
</tbody>
</table>
Java String Class

String a = “Hello World!”;

<table>
<thead>
<tr>
<th>Operation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.length()</td>
<td>12</td>
</tr>
<tr>
<td>a.charAt(1)</td>
<td>e</td>
</tr>
<tr>
<td>a.startsWith(“Hell”)</td>
<td>true</td>
</tr>
<tr>
<td>a.endsWith(“rld”)</td>
<td>false</td>
</tr>
<tr>
<td>a.substring(1,2)</td>
<td>e</td>
</tr>
<tr>
<td>a.concat(“rld”)</td>
<td>Hello World!rld</td>
</tr>
<tr>
<td>a.substring(1,2).equals(“e”)</td>
<td>true</td>
</tr>
<tr>
<td>a.indexOf(“rld”)</td>
<td>8</td>
</tr>
</tbody>
</table>
Java StringBuffer Class

StringBuffer S;

- Mutable strings: operations modify the strings

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append(Q)</td>
<td>Replace S with S+Q. Return S.</td>
</tr>
<tr>
<td>Insert(i,Q)</td>
<td>Insert Q in S starting at index i. Return S</td>
</tr>
<tr>
<td>reverse()</td>
<td>Reverse S. Return S.</td>
</tr>
<tr>
<td>setCharAt(i, ch)</td>
<td>Set the character at index i in S to ch</td>
</tr>
<tr>
<td>charAt(i)</td>
<td>Return the character at index i in S</td>
</tr>
<tr>
<td>toString()</td>
<td>Return a String version of S</td>
</tr>
</tbody>
</table>
Java StringBuffer Class

StringBuffer a = new StringBuffer();

<table>
<thead>
<tr>
<th>Operation</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.append(&quot;Hello World!&quot;)</td>
<td></td>
</tr>
<tr>
<td>a.reverse()</td>
<td></td>
</tr>
<tr>
<td>a.reverse()</td>
<td></td>
</tr>
<tr>
<td>a.reverse()</td>
<td></td>
</tr>
<tr>
<td>a.insert(6,&quot;Fang and the &quot;)</td>
<td></td>
</tr>
<tr>
<td>a.setCharAt(4, '!')</td>
<td></td>
</tr>
</tbody>
</table>
Java StringBuffer Class

StringBuffer a = new StringBuffer();

<table>
<thead>
<tr>
<th>Operation</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.append(“Hello World!”)</td>
<td>Hello World!</td>
</tr>
<tr>
<td>a.reverse()</td>
<td>!dlroW olleH</td>
</tr>
<tr>
<td>a.reverse()</td>
<td>Hello World!</td>
</tr>
<tr>
<td>a.insert(6,”Fang and the ”)</td>
<td>Hello Fang and the World!</td>
</tr>
<tr>
<td>a.setCharAt(4, ‘!’)</td>
<td>Hell! Fang and the World!</td>
</tr>
</tbody>
</table>
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:
  - $T =$ aaaaaaaaaaaaaaab
  - $P =$ 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 appears in P.  
Shift to t

e does not appear in P.  
align P[0] and T[i+1]
Boyer-Moore’s algorithm preprocesses the pattern $P$ and the alphabet $\Sigma$ to build the last-occurrence function $L$ mapping $\Sigma$ 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:
- $\Sigma = \{a, b, c, d\}$
- $P = abacab$

<table>
<thead>
<tr>
<th></th>
<th>$c$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L(c)$</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>
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 $\Sigma$. 
The Boyer-Moore Algorithm

Algorithm BoyerMooreMatch($T$, $P$, $\Sigma$)

1. $L \leftarrow \text{lastOccurrenceFunction}(P, \Sigma)$
2. $i \leftarrow m - 1$ //backward
3. $j \leftarrow m - 1$
4. repeat

   4.1. if $T[i] = P[j]$
      4.1.1. if $j = 0$
      4.1.1.1. return $i$ // match at $i$
      4.1.2. else
      4.1.2.1. $i \leftarrow i - 1$
      4.1.2.2. $j \leftarrow j - 1$
   4.2. else // character-jump
      4.2.1. $l \leftarrow L[T[i]]$
      4.2.2. $i \leftarrow i + m - \min(j, 1 + l)$
      4.2.3. $j \leftarrow m - 1$

5. until $i > n - 1$
6. return $-1$ { no match }

How to shift $i$?
How to shift $i$ after mismatching characters?

- $i \leftarrow i + m - \min(j, 1 + l)$
- Don’t shift back!

Case 1: $j \leq 1 + l$ (a appears after b)

Case 2: $1 + l \leq j$ (a appears before b, jump!)
Another Example

Case 2

Case 1

1
2
3
4
5
6
7
8
9
10
11
12
13
Is it a better algorithm?

- Boyer-Moore’s algorithm runs in time $O(nm + s)$

- An example of the worst case:
  - $T = \text{aaa} \ldots \text{a}$
  - $P = \text{baaa}$

- 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

```
a  a  a  a  a  a  a  a  a  a
b  a  a  a  a  a
6  5  4  3  2  1
b  a  a  a  a  a
12 11 10  9  8  7
b  a  a  a  a  a
18 17 16 15 14 13
b  a  a  a  a  a
24 23 22 21 20 19
b  a  a  a  a  a
```
HW3 (Due on 10/5)  
(Lab on 10/2)

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 Sep. 30 (this Saturday)
- We have NO classes on Oct. 5 and Oct. 12
- We will then continue the basic data structure discussion on Oct. 19.
- Read TB Chapter 3