CS 456 : Advanced Algorithms

Programming Assignment #02

Total Points: 150

Assigned Date : Thursday, October 20, 2016
Due Date : Thursday, November 03, 2016 @ 09:29:59 a.m.

Overview

For your second **programming assignment**, you will implement three versions of a **Pattern Matching Algorithm** – Naive, Rabin-Karp, and KMP – and empirically validate their asymptotic runtime behaviors using **computer generated results**. Capitalization and spacing matter. These pattern matching algorithms run at different complexities and range from worst to best cases. These algorithms are discussed in Ch. 32 of CLRS textbook as well as several quality youtube videos. Specifically, you are expected to think about and address the following questions:

a. What form must the input take to fully explore the different time complexities of each individual algorithm? What input files must you generate to demonstrate the best and worst complexities for each algorithm?
b. What conditions favor or disfavor a particular algorithm being chosen over another?
c. At what size text files do the algorithms begin to show the difference that their individual complexities should demonstrate?
d. How does the measured run time correspond to the theoretical complexity analysis?

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!!**
- **Take backups of your code often!!**.
- You may use any programming language of your choice. However, you **must** make sure that your code compiles and runs on a typical Linux machine. Absolutely **DO NOT** include executables with your submissions. You **MUST** submit a makefile.
- The report part of your solution must be produced using a word processor. I highly recommend **Latex**. Any figures, graphs, plots, etc., should also be produced using appropriate computer applications. Graphs/plots should be properly labeled. Your final report should be in **PDF** format. No exceptions.
- Follow a good coding standard. Use the Google C++ coding standard found here `http://goo.gl/1rC1o`, if you don't already follow one.
- For input, have the user enter a text file that will constitute the data being searched. Next, offer the user the option to input their own pattern or to use a file containing the patterns. The formats of the two files can be found below under sampleInput.txt and samplePattern.txt.

- All three algorithms should be ran on the text file to determine how long each algorithm takes.
- The output should consist of the times for all algorithms, as well as the number of occurrences for each pattern and their line number location in the text. It should be written to a file named **[inputFileName]Out.txt**. A sample output file can be found below.
- Be sure to test enough different size files to accurately graph behavior.
- Total points: [**150 points**]

Deliverables

The due date of this assignment is **Thursday, November 03, 2016 @ 09:29:59 a.m.** A dropbox will be opened for submission on Moodle before the due date. A complete solution comprises of:

- [**64 points**] A report that includes the followings:
  - Motivation and background of the experiment. Explain your learning objectives of this experiment. [**5 points**]
  - Pseudocode of your algorithms appropriately annotated with the theoretical runtime analysis. It is advised to add a code walkthrough of the algorithms that explains why they have the time complexity that they have. [**12 points**]
  - Testing Plan and Test Results including explanations of the sample input files used to test your algorithms and how you derived best, worst and average cases. [**12 points**]
  - Graphs displaying the timing results of your algorithms. Be sure to discuss the complexity of the algorithms and how closely your program follows the theoretical complexities. A minimum of the following graphs are required:
    - A theoretical-actual comparison graph of the timing results of the **Naive Algorithm**. [**4 points**]
    - A theoretical-actual comparison graph of the timing results of the **Rabin-Karp Algorithm**. [**4 points**]
    - Another theoretical-actual comparison graph of the timing results of the **KMP Algorithm**. [**8 points**]
    - A graph comparing the run times of the three algorithms [**4 points**]
  - Justification of your observations. You must be able to justify and/or argue the empirical asymptotic behavior you are observing Be sure to discuss at what set size the algorithms began to display asymptotic behavior, for which properties the different algorithms would be best suited, what characteristics of the text gives worst case results, and any text sizes that produced irregular results. [**10 points**]
  - Conclusion and problems encountered/key insights. Explain on your own words what you learned from this experiment. [**5 points**]
- [**86 points**] A compressed tarball of the directory containing your source codes and a **Makefile**. **Important:** Be sure to test your code on a linux machine. No exceptions! Do not include executables in this tarball; we will do a fresh compile of your code using your Makefile. To create a compressed tarball of the directory source, use the following command: `tar -zcvf name-pr2.tar.gz source/`. Obviously, change the name to your last name.

Tentative Grading Rubric

- Styling: Easy to read, properly indented, etc..., [8 points]

- Input specifications [8 points]
- Output Specifications [10 points]
- Algorithmic Implementation Correctness[60 points]
  - Naive String Search [20 points]
  - Rabin-Karp Algorithm[20 points]
  - Knuth-Morris-Pratt Algorithm [20 points]
- Report Specifications[64 points]

sampleInput.txt

"It seems probable that once the machine thinking method had started, it would not take long to outstrip our feeble powers. They would be able to converse with each other to sharpen their wits. At some stage therefore, we should have to expect the machines to take control."

samplePattern.txt

not take
would be able
have to expect

sampleInputOut.txt

Naive: .0125s
Rabin-Karp: .0109s
KMP: .0723s

pattern: "not take"
occurrences: 1
line : 2

pattern: "would be able"
occurrences: 1
line : 2

pattern: "the"
occurrences: 4
line: 1, 3, 4, 4