IE 170 Laboratory 9: String Matching

Dr. T.K. Ralphs
Scott Denegre
Ashutosh Mahajan

Due April 17, 2006

1 Laboratory Description and Procedures

1.1 Learning Objectives

You should be able to do the following after completing this laboratory.

1. Understand the basic principles of string matching.
2. Understand how preprocessing works.

1.2 Key Words

1. string matching
2. modular arithmetic
3. preprocessing
4. string
5. prefix function

1.3 Scenario

You just received a summer job working in a genetics laboratory in Philadelphia. Your boss tells you that he has discovered a way to alter yeast to make beer taste even better, while having a stronger effect on your system. Your job is to write a program that your boss can use to find all the instances of a particular pattern in the yeast's gene, so he can more effectively analyze the organism and improve the lives of beer drinkers across the globe. If you can accomplish this, you will receive a large raise, in liquid form, and be able to impress all your friends with your summer endeavors when you return to campus in the fall.

1.4 Designing and Analyzing the Algorithm

In this lab, you must implement three different string matching algorithms in order to perform the type of search described above. The user will have the option of searching for a desired string or giving an input file as an argument. The primary focus of the analysis, in this lab, is to compare the different string matching algorithms on the basis of the time needed to search the input string to find all instances of a specified search string.

1.4.1 Program Function

Your program should perform the following functions.

1. Read in the text file(s).
2. Prompt the user for a pattern to search for (if a search file is not given as an argument).
3. Determine whether the pattern is in the text.
4. Report the shift in the file at which each instance of the pattern is found.
5. When the key pattern submitted is "quit" or "exit", the program should end.

1.4.2 Algorithms

For this lab, you will implement and test three different string matching algorithms as described in the class lecture.

- *Naïve-String-Matcher*. The naïve algorithm is similar to sliding a "template" containing the pattern over the text. It begins by checking for a match of the first character of the Pattern string and, if found, continues to look for a match of the rest of the characters in the Pattern. The matching portion runs $\mathcal{O}((n-m+1)m)$ time. There is no preprocessing.
- *Rabin-Karp-Matcher*. During a preprocessing phase, the pattern $P[1..m]$ is interpreted as an integer base N, where N is the number of characters in the alphabet from which the characters of the string are drawn. This integer is converted into its decimal equivalent p. Similarly, we compute a decimal equivalent t_s with the substring of T of length m beginning at shift s for $s = 0, \ldots, n-m-1$. Because These numbers can be extremely large, we apply a hash function to each of them and then compare the hashed value of t_s to the hashed value of p for each shift s. When these two values are equal, then s may be a valid shift and we explicitly compare the pattern to the substring of length m at shift s. The hash function typically used is $h(x) = x \mod q$ for a prime number q. The preprocessing time takes $\theta(m)$ and the matching time is $\theta((n-m+1)m)$.
- *KMP-Matcher*. The KMP algorithm utilizes an auxiliary prefix function $\pi[1..m]$ that contains information about the pattern that can be used to avoid testing useless shifts. The algorithm is similar to the naïve algorithm, but avoids backtracking and reads each character of the string exactly once. The preprocessing time to compare the pattern to itself takes $\theta(m)$ and the matching running time is $\theta(n)$.

2 Laboratory Test Files

The test files for this laboratory are in the zip archive `Lab9.zip` available on the course Web site. The archive will unpack into a directory called `Lab9` with subdirectories `data`, and `shell`. The shell directory contains the files `main.cpp`, `StringMatch.hpp` and `StringMatch.cpp` that contain parts of the code needed to complete the lab. You must fill in the missing pieces. In the data directory is test data files called `input.txt` and an input file called `search.txt` that you may use for testing the program's ability to match a search string. You must choose both a pattern that exists in the text and one that does not exist.

3 Laboratory Assignments

3.1 Programming (50 points)

1. Write a program that behaves in the specified manner. You may use the files in the the `shell` subdirectory to as a guide.

2. Verify that your program compiles and runs correctly using the test files provided for the lab.

3.2 Analysis (20 points)

- (10 points) Plot and compare the running times of the three algorithms for the given input file `search.txt`. Also plot and compare the rough number of operations performed by each algorithm (arithmetic operations and comparisons). Analyze and explain the results.

- (5 points) Compare the running times of the algorithms for various search string lengths. Remember that the size of `input.txt` is close to 9.8 million characters. Compare for at least the following lengths: 5, 50, 500. Analyze and explain the results.

- (5 points) Run each algorithm for the following strings and compare the results:

  – THISSTRINGSHOULDNOTBEINTHEINPUT
  – CGAATTGAGTGGTATTCCACCAGCTCCAAGA

  Analyze and explain your observations.

3.3 Follow-up Questions (30 points)

- (5 points) CLRS 32.1-2
- (5 points) CLRS 32.4-1
- (10 points) CLRS 32.4-5
- (10 points) CLRS 32.1-4