Diego Valdes

CSCE 350

Oct 10, 2016

Project 2

Program 2 Analysis

**Naive Versus Sophisticated String Matching**

I wrote a naive string matching algorithm to be tested against three sample texts:

* Dickens’ A Tale of Two Cities (Gutenberg Project version).
* The first 100 thousand digits of π.
* A sequence of DNA letters A, C, T, G, but with skewed frequency distributions.

The question is how badly does naive perform when tested against these three different texts?

**Naive Comparison**

Naive starts by getting a pattern, counting the number of characters in the pattern, and then searches over the string pattern for a match. The pattern follows:

for( select the first letter in the pattern

for( search through the text for a match to the first character in the patterns

if( a match is found

for( search the next character of the pattern against the next character in the text

if( no match -

break the inner loop

if( the pattern count matches the inner loop iteration -1 -

match found

This means that the number of comparisons that will be made are the number of characters in the text (n) minus the number of characters in the pattern (m). As an example, in the DNA text n = 100, m = 4, the minimum number of comparisons will be 96. Anything above n-m are extra comparisons and what a smarter algorithm would look to eliminate.

Naive could be troublesome, as in a worst case scenario, where near matches are a norm the cost of the algorithm could be as high as n\*m. As an example, if searching for the pattern ABCD against a text that is a million characters of ABCZ in a repeating pattern, that cost would come with no hits on a positive result. Just searching through the given example adds 3 extra comparisons with each hit on the first character. In the example described the number of comparisons translates to:

(2n-1)^k, where k is the number of near matches. If the string was 400 characters long, following the example given above, the number of comparisons would be (2\*4-1)^100… That’s terrible!

The goal of the test would be how much text can naive match before it crashes?

**Further Analysis**

It has become clear that to create a smarter algorithm, one needs to build it with the text to be scanned in mind. This may not be necessary, as with the tests provided naive performed well. But in the situation of searching extreme amounts of text, the number of comparisons could cause the system to crash.

In the case of DNA, the break down of characters is 53 A, 25 C, 5 G, 17 T, which breaks down to A 53% of the time, C 25% of the time… etc. Based on this information and the three patterns which where given to match (AACC, CAGT, TACC) the first pattern will get a match on A 53% of the time, the second pattern will get a hit on C 25% of the time, and the third 17%. This means that for each respective pattern, the number of comparisons will be:

* AACC: n + 0.53n + 0.53 \* 0.25n + 0.53 \* 0.25 \* 0.25n ≈ 1.70n
* CAGT: n + 0.53n + 0.53 \* 0.05n + 0.53 \* 0.05 \* 0.17n ≈ 1.56n
* TACC: n + 0.53n + 0.53 \* 0.25n + 0.53 \* 0.25 \* 0.25n ≈ 1.70n

When comparing the results from naive with the estimated results from the numbers above, it is clear that a better algorithm does exist. When looking at AACC, 87 extra comparisons where done. By employing a better algorithm, that number can be brought down to 70. But with CAGT and TACC, due to them not having ass many lead matches, a smarter algorithm might actually do worse. Based on my calculations those comparisons would increase, CAGT would rise from 41 to 60, and TACC would rise from 33 to 74.