Ian Buitrago
John Richter
4-30-2012

CS 337 project 3: string matching Report

In our project, we implemented the four pattern matching algorithms, Brute Force, Rabin-Karp, Kunth-Morris-Pratt, and Boyer-Moore. We ran a variety of test cases, large and small, to push our algorithms to our limits. What follow are three test cases, our own and the sample source. Generally, RK performs better than BF, while BM is the fastest algorithm, running in sub-linear time. At times, our BF and RK algorithms. In order to ensure that we reduced I/O, we read in 25MiB chunks at time.

Case 1: “Richard Dawkins” in Bible

In our first case, we wanted to determine the speed at which our algorithms could go through the entire bible looking for an obviously nonexistent string, Richard Dawkins. As expected, BF took the longest, while the following three were successively faster. BM performed the best, at 0.025 s.

As far as number of comparisons, Brute force had the most comparisons, followed by KMP, BM, and finally RK. RK had the fewest because it only does character comparisons if the hash values match (collision).

RK had 8665 collisions, indicating where we had to individual test characters.

BF	0.056 sec
RK	0.049 sec
KMP	0.09 sec
BM	0.025 sec

Times

Number of Comparisons

BF	4459694
RK	8665
KMP	4452069
BM	430589

Case 2: pattern "78:25 Man did eat angels" in Bible

For our second case, we took a pattern we knew existed in the bible. The times were as follows. BM performed at half the speed of the other algorithms. Perplexingly, BF performed faster than KMP and RK, but marginally so. This might be due to the fact that we used string based implementation of KMP and RK.

BF	0.106
RK	0.112
KMP	0.11
BM	0.048

Number of Comparisons

BF	2245977
RK	1662
KMP	2242919
BM	169854

Case 3: Sample Source

The sampleSource.txt and the patterns in samplePattern.txt produced highly varied results depending on the machine that the program ran on. The following is the output for a CS Linux machine. As expected, BM performed the best, typically a fraction of the time as other algorithms. However, in the first case for Pattern 1, KMP outperformed BM, probably due to placement of the pattern within the fuller text.

Pattern1: &The Project Gutenberg EBook of Twelve Years of a Soldier's Life in India, by
W. S. R. Hodson&
Pattern 2: &whom t3he
mutiny&
Pattern 3: &Can you find ThIs?&

Pattern 1		
BF	0.267	sec
RK	0.071	sec
KMP	0.041	sec
BM	0.063	sec

Pattern 2		
BF	34.339	sec
RK	23.547	sec
KMP	40.839	sec
BM	3.475	sec
Pattern 3		
BF	23.733	sec
RK	21.537	sec
KMP	25.649	sec
BM	3.207	sec

Conclusion and Final Notes

Depending on placement of the pattern within the text and the actual pattern, performance of our algorithms varied drastically. Patterns with repeating strings--and hence large cores--for example tended to perform well in KMP. BM generally performed the best, as it combines traits of KMP with modifications, reading backwards and working in sublinear time. Brute Force, generally, took the longest. Naturally, since it only increments a character at a time, this is to be expected. Rabin-Karp using Hash function generally ran the second slowest, but it only required comparisons when hash functions were equal, which means fewer comparisons.