Pattern Searching Algorithms

By Jeffrey Chan & Anson Varughese
CS 375 Final Project Report
Fall 2016 Semester

Project Repository located at:
https://github.com/superjeffreyc/CS-375-Fall-2016-Term-Project

Problem

- Given a block of text, what are some various algorithms that can be used to find a pattern or substring within the text?
- Which algorithms provide the best time and space complexities?
- What if there are multiple patterns that need to be searched?
- How will the nature of the data affect the runtime of each algorithm?

Objective

- Implement three different algorithms for pattern searching
    - Brute Force
    - Rabin-Karp
    - Knuth-Morris-Pratt
- Compare time and space complexities
- Calculate running times on various test cases

Algorithm Description

Naive Brute Force Algorithm:

The naive brute force approach to pattern searching is an effortless way to search for patterns. The implementation is very quick and simple and can be implemented quickly as long as the comparison operator for the object is defined. The naive brute force approach for pattern searching is as follows:

```
FOR each character i in the text:
     FOR each character j in the pattern:
          IF (end of text):
               return false
          ELSE:
               IF (next characters in text match pattern):
                    return true
```

The average case time complexity for this algorithm is O(nm) where n and m are the sizes of the text and pattern respectively. The best case time complexity and worst case are also O(nm). Also, this algorithm does not require any pre-processing or any extra space. The space complexity is O(1). The brute force algorithm can be optimized to have less comparisons and on certain data sets, can run as fast as the Rabin-Karp and Knuth-Morris-Pratt algorithms.

Rabin-Karp Algorithm:

The Rabin-Karp approach uses hashing to quickly check multiple patterns in text. There is some negligible pre-processing done before searching the text to compute the hashes but no extra space is needed. The implementation is only slightly more complicated than the naive brute force. The Rabin-Karp algorithm is as follows:

```
patternHash = hash(pattern)
FOR each substring s in the text:
     h = hash(s)
     IF (h == patternHash)and (all characters match):
          return true
return false
```

The Rabin-Karp algorithm has an average time complexity of O(n+m) and a worst time complexity of O(nm). The algorithm preprocesses the pattern and first text window by calculating their hashes in O(m) and uses a rolling hash to compare each substring with the pattern hash in O(n). The time complexity is directly dependent on the implementation of the hashing algorithm and the collisions that take place when searching. It is able to search multiple patterns much faster than the KMP and the brute force pattern searching algorithms. The only extra space it needs is to hold multiple hashes for the multiple patterns.

Knuth Morris Pratt Algorithm

The Knuth-Morris-Pratt pattern searching algorithm is a very fast pattern searching algorithm on certain data sets. It uses a prefix-suffix array to rapidly jump over characters where the pattern is deemed to not exist. Since it needs to compute a prefix-suffix array, there are actually 2 core algorithms involved in the Knuth-Morris-Pratt algorithm. The prefix-suffix array algorithm is implemented as follows:

```
j = 0
FOR each character i in the pattern:
     IF (pattern[i] == pattern[j])
          Increment j and set it to suffixArr[i]
     ELSE IF (J != 0)
          Decrement i to prevent for-loop increment
          Set j to suffixArr[j-1]
     ELSE
          Continue (for-loop increments i automatically)
```

The text searching algorithm is as follows:

```
WHILE(text not ended and pattern not found)
     IF(text character matches pattern character)
          Increment text and pattern index
```

```
        ELSE IF(using pattern index to skip blocks of text)
             Use pattern index to skip characters
        ELSE
             Increment text index
    return (pattern index == pattern size)
```

The Knuth-Morris-Pratt algorithm is very fast if characters are able to be skipped. If the pattern doesn't allow characters to be skipped, then the speed will be around the brute force implementation. The KMP algorithm requires some pre-processing which requires an array the length of the pattern. Therefore, the space complexity is O(n) where n is the size of the pattern. The best, worst, and average time complexity is O(n+m).

Results

In general, the Brute Force approach will be the slowest among the three algorithms. For each of the n substrings, it is going to compare m characters, resulting in O(nm) time complexity. However, it does not require any extra space.

The Rabin-Karp is an improvement on the Brute Force approach and uses hashing to reduce the number of comparisons. By creating hashes for each substring, the average time complexity is O(n+m). However, if multiple hashes collide, the time complexity is no better than Brute Force. The space complexity is similar to Brute Force since it only requires a single integer to hold the hash for single pattern searching (Figure 1). However, for multiple pattern searching, it requires O(p) space, where p is the number of patterns (Figure 2).

The Knuth-Morris Pratt Algorithm's speed is directly dependent on the pattern. If the pattern allows multiple characters to be skipped, then the pattern will be very fast. If the algorithm cannot skip characters, then it will only compare the first character and then move on. Therefore, the algorithm's time complexity will always be O(n+m) for one pattern (Figure 1) and O((n+m)*p) for multiple patterns but it will always require a prefix-suffix array for each pattern. Therefore, the space complexity is O(m) for one pattern and O(m*p) for multiple patterns.

	Average Time Complexity	Worst Case Time Complexity	Space Complexity
Brute Force	O(nm)	O(nm)	O(1)
Rabin-Karp	O(n+m)	O(nm)	O(1)
KMP	O(n+m)	O(n+m)	O(m)

Figure 1. Time and space complexities for single pattern searching

	Average Time Complexity	Worst Case Time Complexity	Space Complexity
Brute Force	O(nmp)	O(nmp)	O(1)
Rabin-Karp	O(n+mp)	O(nmp)	O(p)
KMP	O((n+m)*p)	O((n+m)*p)	O(mp)

Figure 2. Time and space complexities for multiple pattern searching, where p is the number of patterns and m is the average length of all patterns

Test Case 1 (Searching for “the” in Harry Potter chapter. See Appendix A)

There were 34 matches.
Brute force took 39.582 microseconds.
Knuth-Morris Pratt took 25.7915 microseconds.
Rabin Karp took 38.9512 microseconds.

Test Case 2 (Searching for 24 patterns in same Harry Potter chapter. See Appendix A)

Brute force took 902.533 microseconds.
Knuth-Morris Pratt took 579.831 microseconds.
Rabin Karp took 474.439 microseconds.

Test Case 3 (Searching for pattern with repeating substring. See Appendix B)

There were 147 matches.
Brute force took 63.2438 microseconds.
Knuth-Morris Pratt took 22.5137 microseconds.
Rabin Karp took 45.7385 microseconds.

Discussion

In all cases, Brute Force performed the worst among all three test cases. The reason for this is that brute force does not make any optimizations and compares every substring to the pattern.

In test case 1, Rabin-Karp performed around the same time as Brute Force because the pattern “the” is a very short word so hashing it would not provide much benefit. An unexpected result was that KMP performed the best out of all three patterns. This might be because if the character did not match, the algorithm quickly and simply checked to see if the pattern index was at 0. If it wasn’t, then the algorithm would continue onto the next character in the text. There also must have been some optimizations done by the compiler to optimize the “if(pattern != 0)” statement.

In test case 2 (multiple pattern searching), Rabin-Karp performed the best because it hashes all the patterns. Then, for each substring in the text, it only needs to do a hash lookup within the set of patterns, which is a constant time operation, allowing for greater performance over Brute Force and KMP.

In test case 3 (repeated substring searching), KMP outperformed Rabin-Karp and Brute Force by a large margin. The reason for this is because of the prefix-suffix array. The algorithm was able to skip large portions of the text since the pattern had a suffix that was also the prefix in the pattern.

References

1) Harry Potter Chapter by J. K. Rowling
2) http://www.geeksforgeeks.org/ for information about pattern searching algorithms

Appendix A

Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be in-volved in anything strange or mysterious, because they just didn't hold with such nonsense.Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, al-though he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors. Thee Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere.The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about thePotters. Mrs. Potter was Mrs. Dursley's sister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that thePotters had a small son, too, but they had never even seen him. He is boy was another good reason for keeping the Potters away; they didn't want Dudley mixing with a child like that.When Mr. and Mrs. Dursley woke up on the dull, gray Tuesday our story starts, there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happen-ing all over the country. Mr. Dursley hummed as he picked out his most boring tie for work, and Mrs. Dursley gossiped away happily as she wrestled a screaming Dudley into his high chair.None of them noticed a large, tawny owl flutter past the window.At half past eight, Mr. Dursley picked up his briefcase, pecked Mrs. Dursley on the cheek, and tried to kiss Dudley good-bye but missed, because Dudley was now having a tantrum and throwing his cereal at the walls. Little tyke, chortled Mr. Dursley as he left the house. He got into his car and backed out of number fours drive.It was

on the corner of the street that he noticed the first sign of something peculiar a cat reading a map.

Appendix B

AAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAA
CAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABA
ACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAAB
AACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAA
BAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABA
ABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAAB
AABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAA
BAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAA
ABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACA
AABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAAC
AAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAA
CAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABA
ACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAAB
AACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAA
BAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABA
ABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAAB
AABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAA
BAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAA
ABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACA
AABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAAC
AAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAA
CAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABA
ACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAAB
AACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAA
BAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAABA
ABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAAB
AABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAAA
BAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACAA

ABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAACA
AABAABAACAAABAABAACAAABAABAACAAABAABAACAAABAABAAC