Programming Project

Nicholas A Gacharich

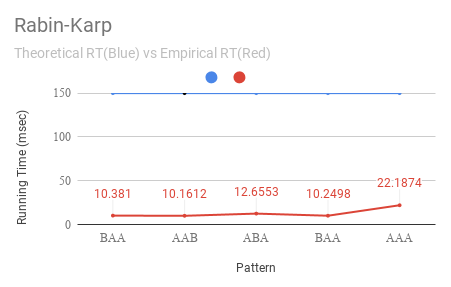
The problem I try to solve is finding where a certain text pattern appears in a long text. I am given the text and the text pattern I am trying to find in the main text. The main text has a size of 50 characters and the text pattern is a size of 3. A real-world application for this problem can be found in spam email. Different email services have a feature which organizes some mail that may be spam in a certain section to avoid the user from falling for them. The program uses pattern recognition to detect common words used in most spam mail like, AMAZING, GUARANTEED, or PROFIT. The first algorithm I used to find patterns in a text was Rabin-Karp. This algorithm is unique because it compares the hash value of the pattern to the hash value of the text in question to see if it is match. This hashing technique allows the algorithm to run quickly. The pseudocode below shows the Rabin-Karp searching technique in the next page.



The running time for Rabin-Karp is O(mn). Where m is the size of the pattern and n is the size of the main text. The programming language I used for the project was C++. For my project I set the size of the text to be 50 characters long and the pattern was 3 characters long. I ran the same number of characters for the text and pattern, but I randomly changed the text and pattern for each run. I ran each algorithm 5 times. The main data structures I used for this algorithm were arrays, characters, integers, and hash values. I generated my input by using rand() function which randomly picks letters A or B for the main text and the pattern. The following chart shows the comparison for the empirical and theoretical running time of the Rabin-Karp algorithm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | N | Empirical RT(msec) | Theoretical RT(mn) | Const c approximation |
| BAA = 3 | BBAABAAAAABBBBBBB  ABABAABAABAA BBABABABBBABB  ABBABBBA = 50 | 10.381 | 150 | C1= 6.92\*10-2 |
| AAB = 3 | ABBABBBBBBABAAAAA  AABBABBAAAAABABABB  AABAAABAAAABBBA = 50 | 10.1612 | 150 | C2=6.77\*10-2 |
| ABA = 3 | BABABBAAABBBBAAAB  ABBBABAAABAAABBBB  BBBBBBBABAAAAABA = 50 | 12.6553 | 150 | C3=8.44\*10-2 |
| BAA = 3 | BABABABBBAABAAAAB  ABAABABBAAAABBABA  BBBABBABABBABBAA = 50 | 10.2498 | 150 | C4=6.83\*10-2 |
| AAA = 3 | BBBBBBAABAAAAAAAA  ABABAAABBABBAAA  AAABAABABBAAABBBBB = 50 | 22.1874 | 150 | C5=1.48\*10-1 |

The graph below shows the results of the empirical and theoretical running time for the Rabin-Karp algorithm. The Y axis is the amount of time the algorithm took to run the function and the X axis shows the pattern used to search in the main text.



The results in the graph show that the empirical time performed faster than the theoretical running time. I think the reason for this is because the theoretical running time takes the worst-case scenario for the algorithm, which is when the pattern matches the text. For example, if pat[]=”AAA” and text[]=”AAAAAA”, however none of the randomly generated letters gave me this worst case scenario.

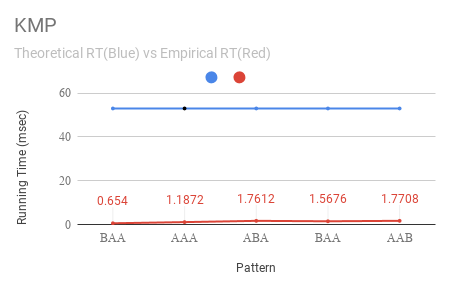
The next algorithm I used was KMP(Knuth Morris Prat) algorithm. This algorithm is similar to Rabin-Karp except that it creates an int array lps[] that is used to compare the characters so that it does not need to look so far back in the text to see if there is a pattern. The problem for this algorithm is the same as the previous one. I am trying to find a pattern in the text. This application is used to detect spam emails by identifying common words used in spam mail. The following pseudocode shows the implementation of this code.



The running time for Rabin-Karp is O(m+n). I used C++ for my programming language of choice. For my project I set the size of the text to be 50 characters long and the pattern was 3 characters long. I ran the same number of characters for the text and pattern, but I randomly changed the text and pattern for each run. I ran the algorithm 5 times. The main data structures I used for this algorithm were arrays, characters, and integers. I generated my input by using rand() function which randomly picks letters A or B for the main text and the pattern. The following chart shows the comparison for the empirical and theoretical running time of the KMP algorithm.

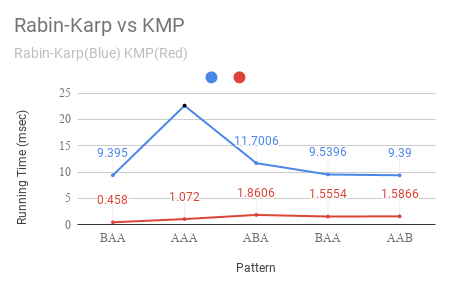
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| m | N | Empirical RT(msec) | Theoretical RT(m+n) | Const c approximation |
| BAA = 3 | BBAABAAAAABBBBBBB  ABABAABAABAABBABA  BABBBABBABBABBBA  = 50 | .654 | 53 | C1= 1.23\*10-2 |
| AAA = 3 | BBBBBBAABAAAAAAAAA  BABAAABBABBAAAAAA  BAABABBAAABBBBB  = 50 | 1.1872 | 53 | C2=2.24\*10-2 |
| ABA = 3 | BABABBAAABBBBAAABA  BBBABAAABAAABBBBBBB  BBBBABAAAAABA  = 50 | 1.7612 | 53 | C3=3.32\*10-2 |
| BAA = 3 | BABABABBBAABAAAABA  BAABABBAAAABBABA  BBBABBABABBABBAA  = 50 | 1.5676 | 53 | C4=2.96\*10-2 |
| AAB = 3 | ABBABBBBBBABAAAAAAA  BBABBAAAAABABABBAA  BAAABAAAABBBA  = 50 | 1.7708 | 53 | C5=3.34\*10-2 |

The graph below shows the results of the empirical and theoretical running time for the KMP algorithm. The Y axis is the amount of time the algorithm took to run the function and the X axis shows the pattern used to search in the main text.



The results in the graph show that the empirical time performed faster than the theoretical running time. I think the reason for this is because the theoretical running time takes the worst-case scenario for the algorithm, which is when the pattern matches the text. For example, if pat[]=”AAA” and text[]=”AAAAAA”, however none of the randomly generated letters gave me this worst case scenario.

I believe that the results I got from the two algorithms is consistent to with the theoretical algorithm because it states the worst-case scenario. In the Rabin-Karp algorithm, I got an empirical running time for the pattern being equal to “AAA” of 22 msecs, which is almost the right condition for the worst-case which would take 150 msecs. It would not be crazy to believe the empirical running time for that pattern would be equal to 150 msecs if the text was also all As. The empirical running times are also consistent because all the running times are really quick for both of these algorithms, therefore it is expected to get a fast empirical running time. I also wanted to compare the two algorithms together using a graph. The following graph shows the empirical running time between KMP and Rabin-Karp algorithm.



As you can see the Rabin-Karp algorithm takes more time to search through the text than KMP. The reason for this is because the running time for Rabin-Karp is O(mn) which is larger than the running time for KMP which is O(m+n).

References

<https://www.youtube.com/watch?v=GTJr8OvyEVQ>

<https://www.youtube.com/watch?v=V5-7GzOfADQ>