Project 4 Report

Name: Isha Gajera

Student ID: igajera/ 58766776

Introduction:

For this project, we have been told to implement 5 string matching algorithms which are the Naïve Algorithm, KMP, Rabin Karp, BMH and Bitap.

Algorithms implemented:

1. **Naïve Algorithm:**
   The naive string-matching algorithm is a simple way to find a pattern in a text by comparing each character of the pattern string to the corresponding character in the text string. If there's a match, the algorithm continues to compare the next characters until the end of the pattern is reached. If there's no match, the algorithm moves to the next character in the text and repeats the above process. The naive string matching algorithm has a time complexity of O(mn), where m is the length of the pattern string and n is the length of the text string.
2. **KMP:**
   The KMP string matching algorithm is more efficient than the naive approach. It creates a partial match table that records the length of the longest proper prefix that is also a suffix of each substring in the pattern. The algorithm then compares the pattern string to the text string, and if a character mismatch is found, it uses the partial match table to determine how many characters to skip in the pattern string before the next comparison. The KMP algorithm has a time complexity of O(n+m), where n is the length of the text string and m is the length of the pattern string.
3. **Rabin Karp:**
   The Rabin-Karp algorithm is another efficient string-matching algorithm that uses a hashing function to compare the pattern string to substrings of the text string. It computes a hash value for the pattern and for each substring of the text, and if the hash values match, it checks whether the substrings are identical. If a match is found, the algorithm slides the pattern over by one character and repeats the process. The Rabin-Karp algorithm has a time complexity of O(nm), where n is the length of the text string and m is the length of the pattern string. For the projects in this course, I implemented the Rabin Karp algorithm using the XOR method taught in the class.
4. **BMH:**
   The Boyer-Moore-Horspool (BMH) algorithm is another fast-string matching algorithm that uses a preprocessing step to generate a bad character table, which records the rightmost occurrence of each character in the pattern string. When comparing the pattern string to the text string, the algorithm starts matching from the right end of the

pattern string, and when a character mismatch is found in the text string, it uses the bad character table to determine how many characters to skip in the text string before the next comparison. The BMH algorithm has a time complexity of O(nm + k), where n is the length of the text string and m is the length of the pattern string, k is size of alphabet.

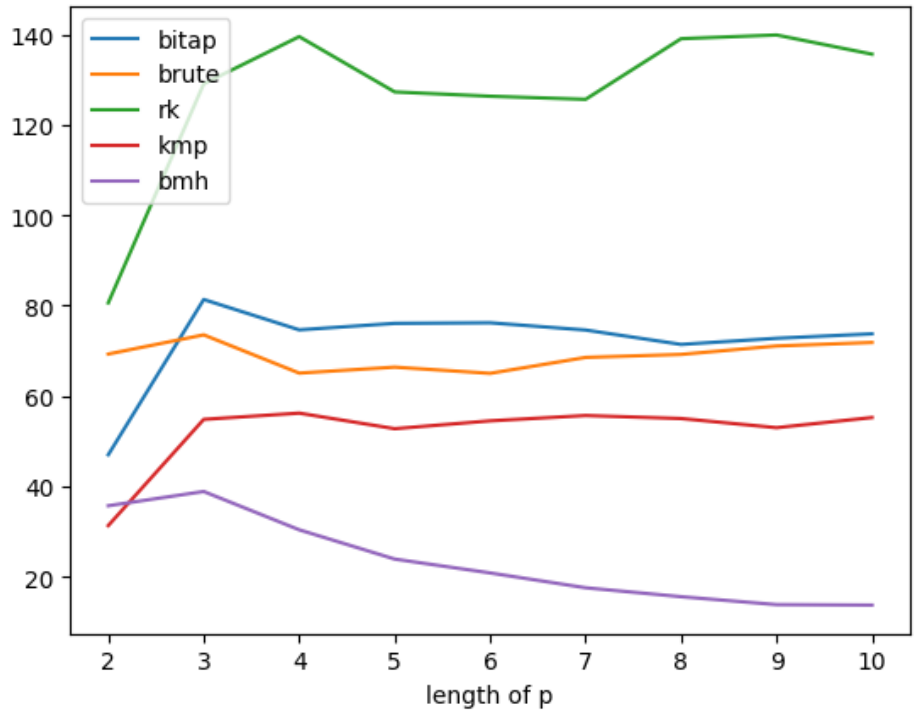
5. **Bitap:**
   The Bitap algorithm, also known as the shift-or algorithm or the Baeza-Yates-Gonnet algorithm, is a string-matching algorithm that uses bitwise operations and a bit array to perform pattern matching. It works by creating a bit mask for each character in the pattern string and then comparing the mask with a shifted version of the text string. If the bitwise AND operation between the two masks produces a non-zero result, the algorithm shifts the mask and continues the comparison. The BMH algorithm has a time complexity of O((n+k)*m/w), where n is the length of the text string and m is the length of the pattern string, k is the alphabet, w is the word size.

Conclusion:

The runtime for the algorithms explained above when I used a **randomly generated string T** of length 10,000 is given by the below table:

Length of P	2	3	4	5	6	7	8	9	10
bitap	46.991	81.371	74.649	76.083	76.215	74.608	71.421	72.782	73.768
rk	80.624	129.115	139.576	127.332	126.375	125.693	139.117	139.939	135.687
bmh	35.701	38.856	30.377	23.893	20.824	17.539	15.564	13.809	13.706
brute	69.29	73.54	65.069	66.391	65.016	68.544	69.209	71.079	71.852
kmp	31.31	54.83	56.207	52.762	54.498	55.678	55.015	52.986	55.236

Following is the graph, I got:



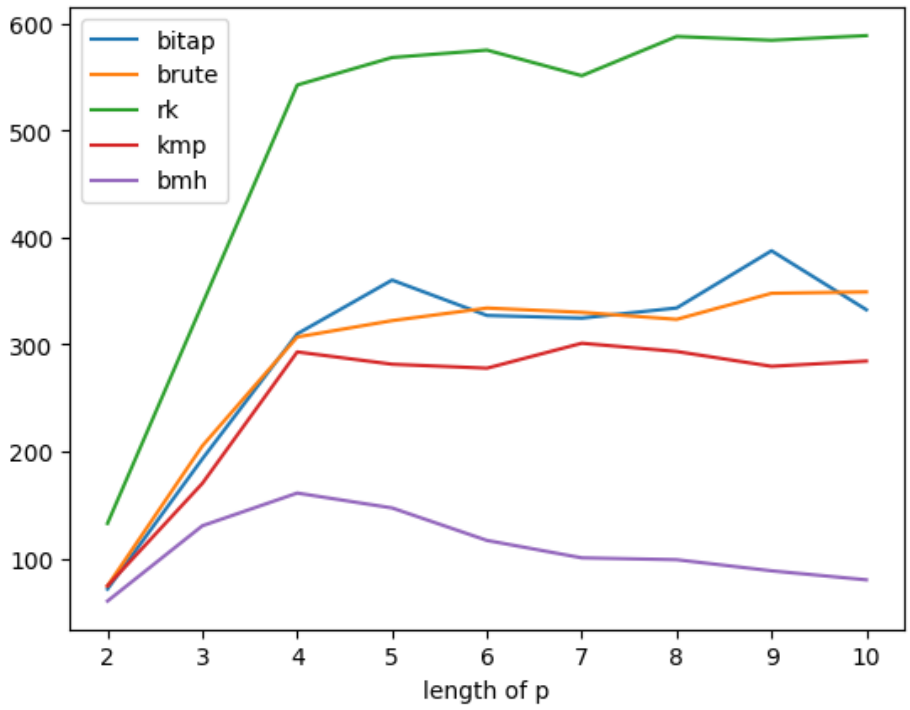
The output is gained by using 1000 pattern strings each of length 2,3,4,5,6,7,8,9,10 and running each string-matching algorithm against them. The time taken by the algorithm is in microseconds and is the Y axis of the graph. We can see that the BMH algorithm has the lowest run time, whereas Rabin Karp took the most time in running the code. For pattern length =10, Rabin Karp performed the worst by taking 135 microseconds and BMH performed the best by taking only 13 microseconds. Furthermore, it can also be observed that Bitap, KMP and Naïve Algorithm had almost comparable times lying between the range of 50-80 microseconds.

As the length of pattern string increases, it can be seen that the difference between runtime of algorithms also increased.

The runtimes for the algorithms when I used **speeches.txt** as Text string are:

Length of P	2	3	4	5	6	7	8	9	10
bitap	2.33663	2.70297	108.782	279.337	308.752	308.079	313.475	312.327	471.624
rk	116.099	357.396	532.772	524.574	774.337	529.951	545.139	548.446	717.04
bmh	85.2277	134.139	301.119	137.782	120.149	111.267	99.1881	92.8416	82.5941
brute	62.2277	186.149	302.614	309.109	316.426	311.554	324.743	549.376	332.267
kmp	50.7426	305.109	239.337	260.723	263.366	260.901	266.842	279.921	289.465

The graph is shown below:



The output is gained by using 100 pattern strings each of length 2,3,4,5,6,7,8,9,10 and running each string-matching algorithm against them. The time taken by the algorithm is in microseconds and is the Y axis of the graph. We can see that the BMH algorithm has the lowest run time, whereas Rabin Karp took the most time in running the code. For pattern length =10, Rabin Karp performed the worst by taking 717 microseconds and BMH performed the best by taking only 82 microseconds. Furthermore, it can also be observed that Bitap, KMP and Naïve Algorithm had almost comparable times lying between the range of 250-350 microseconds.

Here, when the pattern string had a length of 2, we can see that the difference in runtime of the algorithms was about 100 microseconds. However, when the same length got changed to 10, we can see that difference has now become almost 600 microseconds.

This shows that as the pattern length increases, the BMH algorithm performs the best compared to all other algorithms in the graphs. The Naïve Algorithm has a time complexity that is almost like KMP and Bitap, showing that in some cases the Naïve Algorithm can also provide better solutions for string matching. The BMH and KMP perform better in this case since these algorithms can skip over many characters while pattern matching making them faster and an efficient choice. One of the reasons for Rabin Karp to perform worse than the others is that here, there may be many false positive hashes that may have contributed to larger run times.