Markov Analysis

CS201

Zhaoxi Zhang, zz115

2017/02/16

Part A:

Brute vs Efficient

1.

The running time for brute generating random text with different

File name	Input file length/ char	Order	Return length/ char	Running time/s
alice.txt	163187	1	200	0.11391303
alice.txt	163187	1	400	0.163927293
alice.txt	163187	1	800	0.351913155
alice.txt	163187	5	200	0.048222674
alice.txt	163187	5	400	0.079225047
alice.txt	163187	5	800	0.185377318
alice.txt	163187	10	200	0.050227192
alice.txt	163187	10	400	0.09239751
alice.txt	163187	10	800	0.188217415
hawthorne.txt	496768	1	200	0.269766376
hawthorne.txt	496768	1	400	0.566922462
hawthorne.txt	496768	1	800	1.008073325

The running time for different cases are given in table 1. As for controlled cases, for example when the order is the only variable, the number of data point is very limited. For each case, return file length as the only variable, input file length as the only variable, order as the only variable, the result for

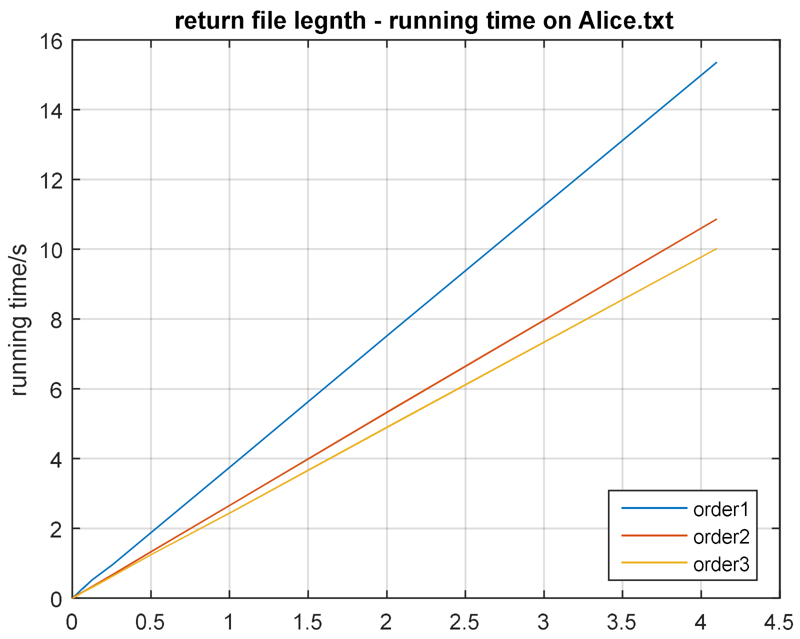

Figure 1

running time are shown in figure 1,2 and 3.
As shown in figure 1, with the same order number, the running times have linear relationships with the return text file length.

As

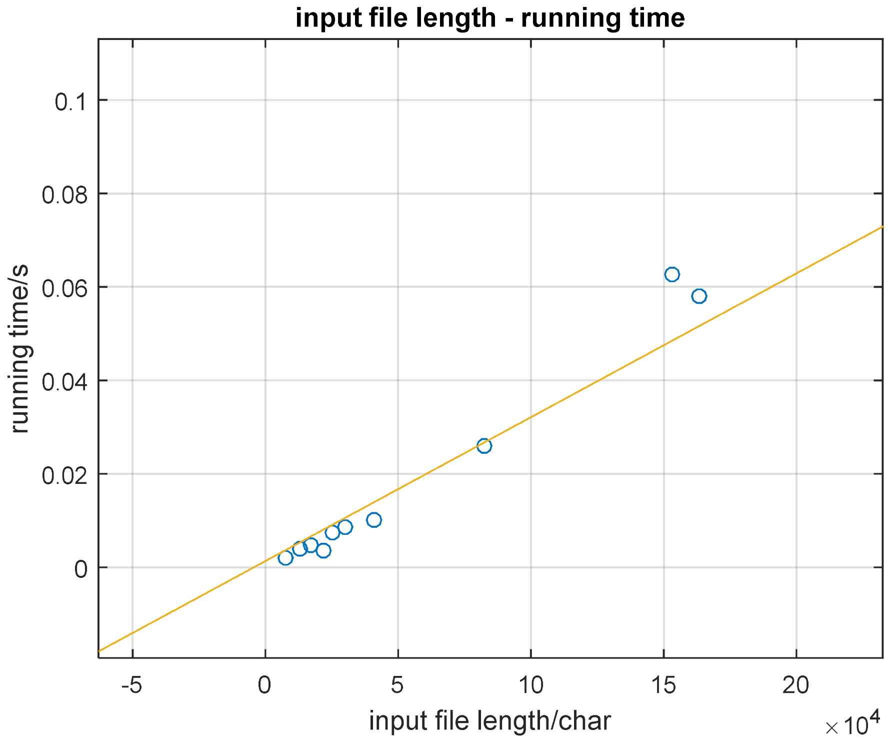

Figure 2.1

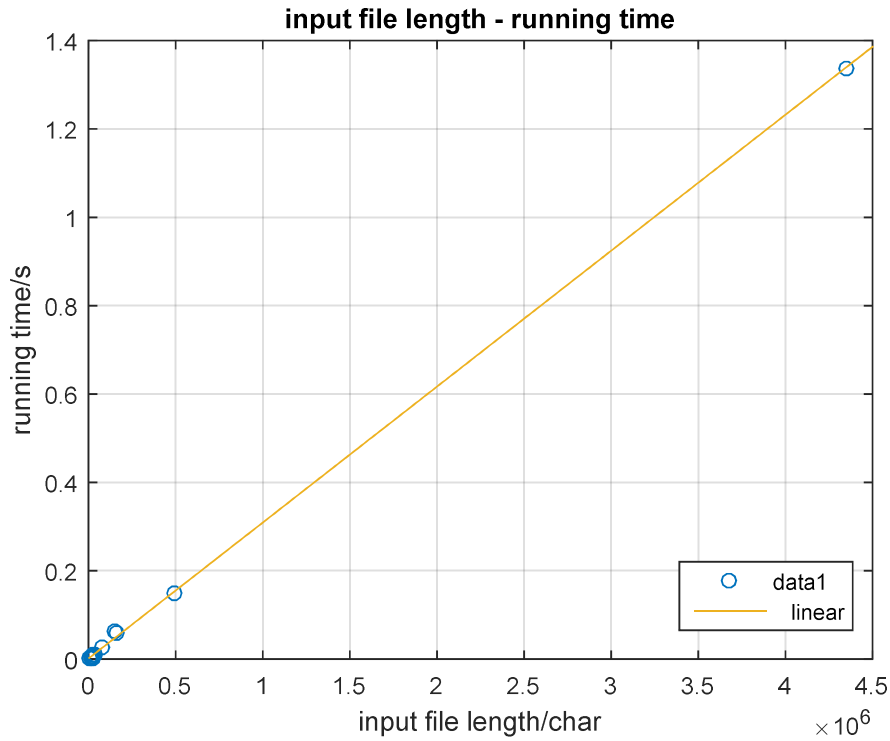

Figure 2.2

shown in figure 2.1 and 2.2, with the same order and return length, the relationships between the input file lengths and the running time is linear.

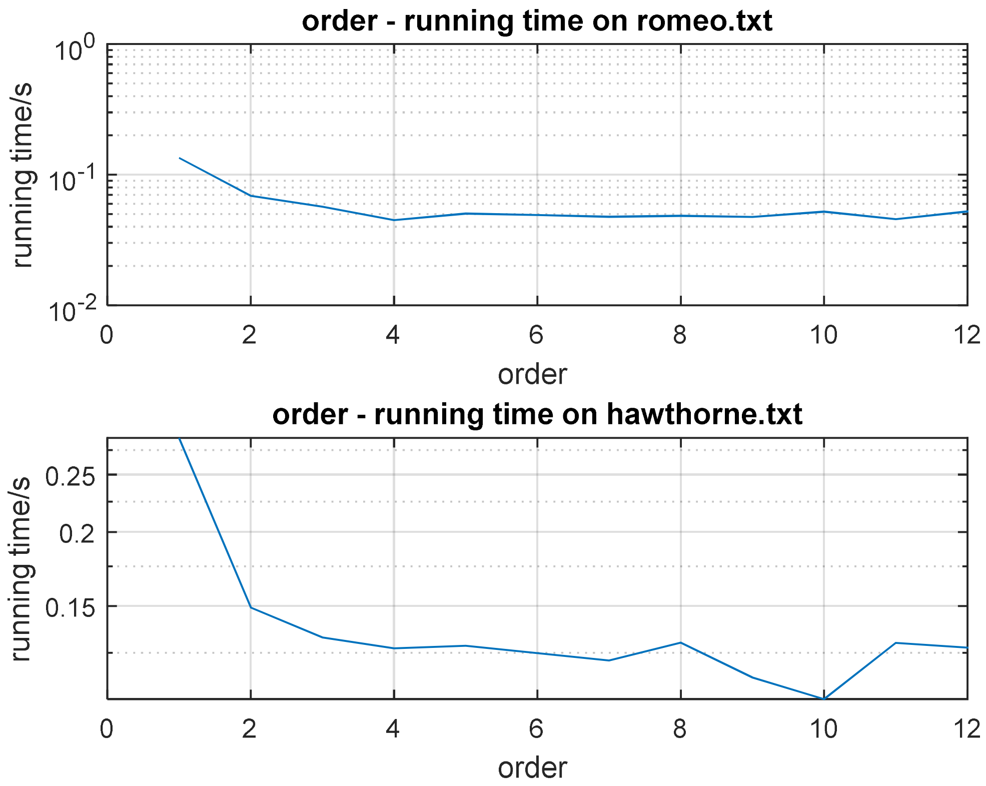

Figure 3

As shown in the semilog plot for order – running time, the relationship is uncertain. It really depends on the diversity of words in the input text. When the order is increased from very low, like 1, to a value of 4, a decrease of running time can be observed As the order number become larger than 4, the size of the following char array will be almost a constant, and therefore, there will not be obvious decrease in running time.

2.

Based on figure 3, the running time for an order 4 is about 1/2 of an order 3 BruteMarkov running time. The running time for order 3 with input length $4 \cdot 10^4\ char$ is about 8s. Therefore for order 4 case, it would be about 4s. According to the linear relationship shown in figure 1, the total time for an input length of $5.5 \cdot 10^6\ char$ will be

$$\frac{5.5 \cdot 10^6}{4 \cdot 10^4} \cdot 4s \approx 550s.$$

3.

The running time for the EfficientMarkov to generate text of length 200, 400, 800 and 1600 are 0.040375004, 0.076509729, 0.14100352, 0.288235697 in second respectively.

4.

Output length of {200, 400, 800, 1600, 3200, 6400} are used for testing. The input file is alice.txt. The order used for testing is 3.

With the given hashcode methods the running times are:

[0.002623753, 0.001988999, 9.01071E-4, 4.6249E-5, 0.001102516, 0.001315136] in seconds.

However this hashcode will generate collisions and causes the EfficientWordMarkov to generate wrong output.

Hash code method 1:

```
StringBuilder for_hash = new StringBuilder();
for (String each: myWords)
{
    for_hash.append(each);
}
String concat_all = for_hash.toString();
myHash = concat_all.hashCode();
```

The running times are:

[0.010957187, 0.014288637, 0.019592013, 0.035635612, 0.05726132, 0.10090911] in seconds.

This method is slow due to the time cost for stringbuilder in each iteration for each wordgram.

Hash code method 3:

```
ArrayList<String> all_words = new ArrayList<>(Arrays.asList(myWords));
        myHash = all_words.toString().hashCode();
```

[0.027439372, 0.034037398, 0.035252898, 0.064607394, 0.083954818, 0.155610619] in seconds

Hash code method 4:

```
myHash = this.toString().hashCode()
```

[0.019400191, 0.028097095, 0.031505521, 0.066295622, 0.092544016, 0.168762285] in seconds

5.

The original file length: 30151

With 50 trials:

The output length in characters are:

[6513, 41024, 14015, 4434, 99725, 13939, 59569, 52173, 52009, 7209, 18495, 6515, 19930, 16835, 24025, 51532, 50154, 48410, 55412, 21117, 6908, 28365, 38590, 58350, 1752, 26345, 2750, 7409, 14973, 24918, 62146, 43569, 34713, 64939, 17652, 12685, 36327, 1602, 6692, 16486, 2638, 12663, 38684, 15621, 41528, 29731, 61575, 44407, 21590, 48790, 29769, 69182, 17603, 86444, 11531, 2750, 18691, 11526, 24568, 845, 16851, 56200, 25864, 6399, 37160, 41678, 12620, 1350, 10054, 36547, 38737, 116118, 14392, 11833, 33463, 52808, 29398, 885, 23946, 64470, 26097, 4442, 49602, 21498, 21436, 34492, 11013, 17307, 18862, 51225, 27395, 13844, 53794, 56231, 76588, 98572, 4133, 3753, 3822, 12922]

The average length is 29981 which is close to the original length of the file

Part B:

The running time of a HashMap should be a constant, when plotted it would be a horizontal line regardless of the map size. In contrast, the extraction time for a TreeMap will be log(n), where n is the size of the map, as binary search is used in a TreeMap. In our cases, the map size is relatively small and the difference is not obvious.