Analytical write-up for Markov

Yuanjie Jin

Part 1.

To generate each random character, the brute force method needs to loop through the whole String. Thus, it's obvious that the run time is linearly proportional to the number of random characters MarkovModel generates, denoted as *N*, and the size of the text it trains on, denoted as T. The trend seen from Table 1 confirms this.

In generating the same number of random characters using MarkovModel, order-1 takes more time than order-5 does, whereas order-10 takes about the same time as order-5 (see Table 1). My hypothesis is that since order-1 grams consists of single letters, they would have much more occurrence than order-5 and order-10 grams which are strings. Therefore, order-1 grams have more following characters to store in the ArrayLists and the model takes a little more time to run. On the other hand, the occurrences of order-5 and order-10 grams are expected to be at a similar magnitude, which does not cause a substantial change in running time.

Table 1. The time spent by MarkovModel in generating random texts of different lengths trained on Romeo.txt

Number of letters \ order	1	5	10
100	0.125	0.062	0.062
200	0.219	0.125	0.125
400	0.422	0.25	0.25
800	0.844	0.5	0.5
1600	1.687	1	1

As the time is proportional to the text size and the hathorne.txt file has about three times longer size as the romeo.txt file, I expect that the time taken on hathorne.txt would be roughly three times as long as the time taken on romeo.txt. The results shown in Table 2 basically match my expectation. Using the same reasoning, I think producing 1600 random characters with an order-5 Markov model trained on the King James Bible will take around 28.7 sec, because its size is about 28.7 fold longer than romeo.txt.

Table 2. The time spent by MarkovModel in generating random texts of different lengths trained on hathorne.txt

order=5 / Number of letters	expected	empirical
400	0.817	0.718
800	1.634	1.422
1600	3.268	2.844

The smarter mode takes about 0.391 sec to generate the initial 200-letter text. However, after that, generating texts with the same size or larger sizes is almost immediate (the time displays 0 most of the time, meaning that it is too short to be detected at the millisecond level, see Table 3). This makes sense because the first time the model is called for a particular order, the whole file needs to be scanned and a HashMap needs to be constructed, which takes a fair amount of time. After that, the HashMap can be reused to generate random texts using the Markov model as long as the order remains the same, which is apparently faster than the brute force method.

Table 3. The time spent by MapMarkovModel in generating random texts of different lengths trained on Romeo.txt

order / Number of letters	5
200 (first time)	0.391
200	0
400	0
800	0
1600	0

Part 2.

To test the performance of TreeMap and HashMap in WordMarkovModel, I choose order-5 grams to generate 20,000 random words with the two kinds of Maps trained on text files of different number of keys. The names of text files I used for training are listed in Table 4 (kjv3-7 are truncated versions of kjv10). The time listed is the time for generating random texts only, not including the time of Map construction. I run the program three times for each file and calculate the average time, which is used in plotting Diagram 1.

When the number of keys is relatively small, there is no significant difference in the performance between TreeMap and HashMap, with TreeMap spending a slightly shorter time (see Diagram 1 and the blue shaded part in Table 4). However, as the number of keys gets greater, TreeMap tends to take more time than HashMap. This difference becomes more obvious when the number of keys increases (see Diagram 1 and the yellow shaded part in Table 4). Although tests with numbers of keys greater than 383,175 is not able to be preformed in my laptop, the general trend can be expected from current results, that is, HashMap has a slight advantage in performance over TreeMap with a large number of keys, at least in WordMarkovModel.

Diagram 1.

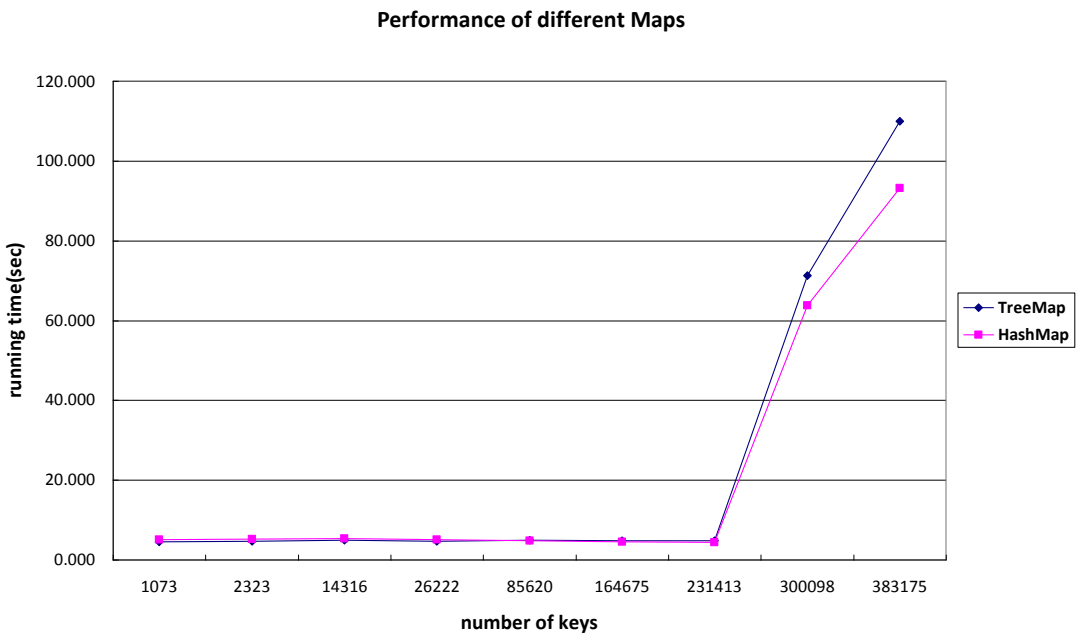

Table 4. the difference in performance between TreeMap and HashMap using order-5 WordMarkovModel generating 20,000 random words

text file		obama-nh	poe	melville	Alice	kjv3	kjv4	kjv5	kjv6	kjv7
number of keys		1073	2323	14316	26222	85620	164675	231413	300098	383175
TreeMap Runtime (sec)	1	4.531	4.688	4.953	4.656	4.875	4.641	4.875	68.968	111.953
	2	4.547	4.656	4.828	4.594	4.891	4.578	4.718	72.344	108.063
	3	4.531	4.657	4.828	4.625	4.984	5.063	4.797	72.562	109.828
	mean	4.536	4.667	4.870	4.625	4.917	4.761	4.797	71.291	109.948
HashMap Runtime (sec)	1	5.016	5.125	5.25	5.031	4.625	4.547	4.421	64.062	93.281
	2	5.063	5.141	5.266	5.078	4.781	4.438	4.359	63.562	93.062
	3	5.047	5.156	5.281	4.984	4.719	4.5	4.422	63.531	93.188
	mean	5.042	5.141	5.266	5.031	4.708	4.495	4.401	63.718	93.177