Kenneth Marenco (kem78)

You're given a class Benchmark that runs several tests that allow you
to compare the performance of the default, brute-force BaseMarkov and the more
efficient map-based EfficientMarkov code. The code you start with uses data/hawthorne.txt,
which is the text of A Scarlet Letter a text of 487,614 characters
(as you'll see in the output when running the benchmark tests). The class uses BaseMarkov,
but can be easily changed to use EfficientMarkov by changing the appropriate
line in the main method. You're free to alter this class.

1. Determine (from running Benchmark.java) how long it takes for BaseMarkov to
generate 2,000, 4,000, 8,000, 16,000, and 32,000 random characters using the default
file and an order 5 Markov Model. Include these timings in your report. The program also
generates 4,096 characters using texts that increase in size from 487,614 characters to
4,876,140 characters (10 times the number). In your analysis.txt file include an explanation
as to whether the timings support the O(NT) analysis. Use the fact that for some runs N is
fixed and T varies whereas in the other runs T is fixed and N varies.

Brute Force

Starting tests

time	source	#chars
0.170	487614	1000
0.375	487614	2000
0.708	487614	4000
1.262	487614	8000
1.928	487614	16000
3.851	487614	32000
8.658	487614	64000
0.540	487614	4096
0.996	975228	4096
1.394	1462842	4096
1.896	1950456	4096
2.663	2438070	4096
2.902	2925684	4096
5.365	3413298	4096
6.373	3900912	4096
6.884	4388526	4096
7.226	4876140	4096

Finished tests

This assumption is supported by the run times above. We can see that the length of
the ArrayList for getRandomText is populated based on the length that was placed
in the parameter giving a for loop that runs N times. With each time the ArrayList
runs out of space it must allocate more memory to store more Strings. Within the for
loop that runs N times it calls the getFollows method which does a similar process
that takes T. This leads to a overall runtime of O(N*T) when using this ArrayList
Brute-Force method.

2. Determine (from running Benchmark.java) how long it takes for EfficientMarkov to

generate 2,000, 4,000, 8,000, 16,000, and 32,000 random characters using the default file and an order 5 Markov Model. Include these timings in your report. The program also generates 4,096 characters using texts that increase in size from 487,614 characters to 4,876,140 characters (10 times the number). In your analysis.txt file include an explanation as to whether the timings support the O(N+T) analysis.

Efficient

Starting tests

time	source	#chars
0.120	487614	1000
0.115	487614	2000
0.094	487614	4000
0.080	487614	8000
0.080	487614	16000
0.082	487614	32000
0.099	487614	64000
0.078	487614	4096
0.212	975228	4096
0.220	1462842	4096
0.254	1950456	4096
0.362	2438070	4096
0.430	2925684	4096
0.514	3413298	4096
0.569	3900912	4096
0.661	4388526	4096
0.878	4876140	4096

Finished tests

For the Efficient Markov, we utilize the object of hashmaps to reduce the time needed for actions to constant time or O(1) which is utilized N times when we add new Strings to the values of the corresponding key in the setTraining method. This is added T times from the T number of times using the Hashmap of again constant time O(1). These together give the class a runtime of O(N+T).