Marcus Deans (md374)

Run WordGramBenchmark for wordgrams of size 2-10 and record the number of WordGram values/objects that occur more than once as reported by the runs. For example, with WSIZE = 2, which generates 2-grams, the output of benchmark and benchmarkShift each indicates that the total # wordgrams generated is 177,634 and that the # unique wordgrams is 117,181

This means there are 177,634 - 117,181 = 60,453 WordGram values that occur more than once. Find these same numbers/values for other orders of k and complete the table below for different k-grams/different values of WSIZE

WSIZE	# duplicates
2	60,453
3	10,756
4	1, 987
5	667
6	362
7	226
8	151
9	105
10	73

=====

Explain in your own words the conceptual differences between the benchmark and benchmarkShift methods. (Answered Question 2)

Answer these questions:

(1) Why the results of these methods should be the same in terms of changes made to the HashSet parameter passed to each method.

The two benchmarking methods will achieve the same result testing WordGrams of size WSIZE and using all of the words contained within the file. Each method will end up generating the same Wordgrams so the HashSet for each will end up being the same.

(2) What are the conceptual differences between the two benchmarking methods.

The conceptual difference is that the benchmarkShift uses WordGrams that are run through the ShiftAdd method such that the words that comprise them are shifted through the entire volume of words, whereas benchmark simply generates new WordGrams and moves the starting index for them over each time.

(3) We will now analyze and compare the amount of memory allocated for ALL arrays in the two methods. For the benchmark() method, answer the following questions, giving answers in terms of variables such as WSIZE and the number of words in the text, N:

(a) Are there any String[] arrays created directly within the method? If yes, what are their sizes? You do not need to count ArrayLists.

There is a String[] array created within the method with size the length of the ArrayList list, which in turn derives its length from the number of strings (N) in the file.

(b) How many WordGram objects are created in total, either directly or indirectly? Note that the shiftAdd() method creates and returns a NEW WordGram object every time it's called.

The method will created N-WSIZE+1 WordGram objects every time.

(c) Each WordGram object stores a String[] array, created in the constructor, which takes up memory. What's the size of this array for each WordGram?

For each WordGram, the String[] array has size WSIZE; thus for WSIZE 10 each Wordgram's String[] array has size 10.

(d) Does your shiftAdd() method create additional String[] arrays? If yes, what's the total size of arrays created over all shiftAdd() method calls if applicable? (The answer may be different depending on your implementation.)

Yes, my shiftAdd() method creates additional String[] arrays, due to the implementation method the new arrays have the same size as the existing String[] arrays so will remain at size WSIZE. In any case, benchmark() won't end up calling shiftAdd().

(e) Based on your answers to (a)-(d), what's the total amount of memory taken up by all String[] arrays, in terms of their total size?

The total amount of memory taken up by all String[] arrays will be WSIZE*(N-WSIZE+1) + N. Going through the program it first creates a String size N then creates more WordGram objects with string size WSIZE, combining these gives the answer.

(4) Answer the same questions (a)-(e) from question (3) for the method benchmarkShift()

(f) Are there any String[] arrays created directly within the method? If yes, what are their sizes? You do not need to count ArrayLists.

There is a String[] array created within the method with size WSIZE.

(g) How many WordGram objects are created in total, either directly or indirectly? Note that the shiftAdd() method creates and returns a NEW WordGram object every time it's called.

The method will created N-WSIZE+1 WordGram objects every time. This is the same as above.

(h) Each WordGram object stores a String[] array, created in the constructor, which takes up memory. What's the size of this array for each WordGram?

For each WordGram, the String[] array has size WSIZE; thus for WSIZE 10 each Wordgram's String[] array has size 10.

(i) Does your shiftAdd() method create additional String[] arrays? If yes, what's the total size of arrays created over all shiftAdd() method calls if applicable? (The answer may be different depending on your implementation.)

Yes, my shiftAdd() method creates additional String[] arrays, due to the implementation method the new arrays have the same size as the existing String[] arrays so will remain at size WSIZE.

(j) Based on your answers to (a)-(d), what's the total amount of memory taken up by all String[] arrays, in terms of their total size?

The total amount of memory taken up by all String[] arrays will be WSIZE*(N-WSIZE+1) + WSIZE. Going through the program it first creates a String of size WSIZE then creates more WordGram objects with string size WSIZE, combining these gives the answer.

(5) Compare your answers for total memory from (3)(e) for benchmark() and (4)(e) for benchmarkShift(). Is the total amount of memory the same or different in benchmark() and benchmarkShift()? If different, which one uses more memory?

benchmarkShift() uses more memory.