Ashley Hoffman
Matthew Wipfler

Data Structures Final Project

PART 4:

Profile times for 3 ebooks (with -t ___,10 option) – BEFORE OPTIMAZATION

ebook	Total time (ms)	SSMatrix init time (ms)	Top-J time (ms)
ebook1.txt	1,030	890	421
ebook2.txt	20,108	18,521	1,103
ebook3.txt	202,418	199,824	1,422

For these tests, the collection of vectors is HashMap<String, HashMap<String,Int>>. The semantic vectors are calculated for every word and takes a while to form; however, the analyzing time is very fast because of this. The ability to analyze the words/sentences effeciently will help later in parts 6 and 7 where there is more analyzing than just one Top-J query.

OPTIMIZATION: The above tests show that most time is spent creating the initialization matrix. Further details of the profile results show that most time is spent in the LinkedList.contains(...) method (which iterates over the linked list). Looking back at the code, there were two lines that called the contained method: one line iterated over every word in a sentence (every sentence too!) and the other contains() call was on the wordsUsed (which goes up to n words of a sentence of length n). To fix the delay, the data types were changed.The LinkedList<LinkedList<String>> was changed to a LinkedList<LinkedHashSet<String>> and another LinkedList was also changed to LinkedHashSet. Of course, code existing for parts 1, 2, and 3 had to be updated, but those changes were minimal. The overall code layout stayed the same. Below are the results from running the same tests as above.

ebook	Total time (ms)	SSMatrix init time (ms)	Top-J time (ms)
ebook1.txt	531	202	156
ebook2.txt	4,545	2,655	1,406
ebook3.txt	44,924	42,425	1,389

Further optimization:

ebook	Total time (ms)	SSMatrix init time (ms)	Top-J time (ms)
ebook1.txt	521	156	202
ebook2.txt	4,171	2,358	1,046
ebook3.txt	40,306	37,732	1,240

Ashley Hoffman
Matthew Wipfler

Part 4 Written Answers:

NOTE: N=the number of unique words in a text file and S=the max number of unique words any word can appear with.

a) The data structure used for vectors: The collection of vectors is stored using HashMap<String, HashMap<String,Integer>>. The data structure for a single vector is then HashMap<String,Integer>. The asymptotic memory usage of one vector: O(S). The asymptotic memory usage of all of the vectors: O(N*S). This memory usage is reasonable because, for a single vector, you only need to go through each unique word that the word in question appears with one time; while for the set of vectors, that process is repeated N times.

b) The algorithm for cosine similirity uses two non-nested for loops. The pseudocode is here:

```
Given two vectors b and q:
    Loop through b keys:
            sumU2 += this b value^2.
            If q contains b key, multiply b value and q value and add to sumOfUV.
    Loop through q keys:
            sumV2 += this q value^2
  sqrt = sqrt(sumU2*sumV2)
  if sqrt == 0 return null
  else return sumOfUV / sqrt
```

The first for loop goes through the keySet of a unique base word and the values of query word. This means that the asymptotic running time is O(Sb+Sq) where Sb=S for base and Sq=S for query. This running time is reasonable because it only relies on the max number of unique words any word can appear with and the amount of unique words total.

c) This algorithm calculates the Top-J similar words:

```
Check if comparison key exists and if maxNumber > 1, otherwise return null;
Create resultList for top results.
Loop through N unique word vectors                 ---- O(N)
calculateSimilarity to comparison key              ----O(Sb+Sq) for cosine siimilarity
        Continue if result is null
        Take result and addFirst on list               ----O(1)
        Sort list using comparator                     ----O(JlogJ) where J is the maxNumber
        Check list size, if greater than maxNumber, removeLast      ----O(J) for .size and O(1) for .removeLast
Return list
```

Overall, O(N*(Sb+Sq)) is the complexity because in larger files (such as books) the S values will generally overpower the JlogJ and J values. Sb+Sq was not simplified further because they generally will both be pretty large numbers. This answer can be justified by the following profiling results for this algorithm:

Ebook 2

J 2 = 1328 ms

J 4 = 1046 ms

J 8 = 984 ms

J 16 = 1015 ms

J 32 = 968 ms

J 128 = 984 ms

Notice how the time does not depend on the value of J. This running time is reasonable because O(N*(Sb+Sq)) follows the profile above and is reasonably fast.

d) One improvement that was made was that many LinkedLists were replaced with HashSets because searching for an element in a LinkedList is O(M) (if M=size of the list) while searching for an element in a HashSet is O(1). Profiling was key in determining the changes. Run-time measurements from before and after optimization are shown on page 1. Profiling was extremely informative and was what guided our choices on how to optimize the program.

PART 5:

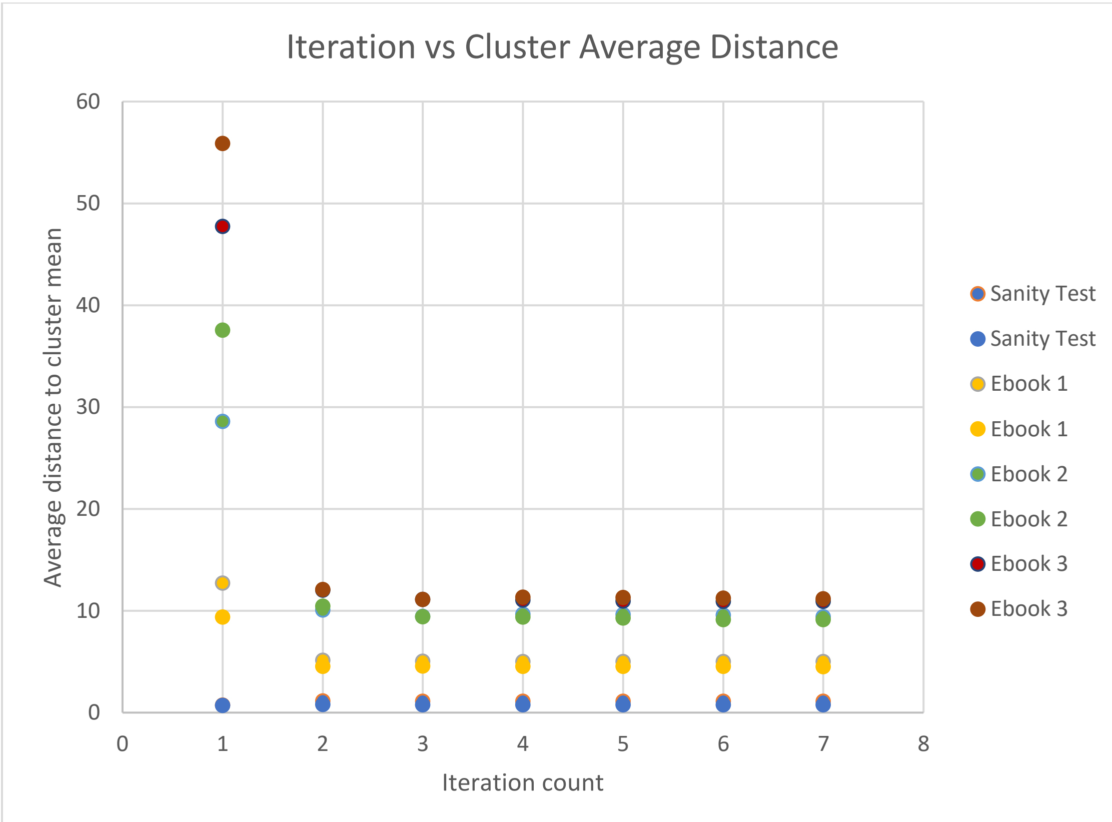
```
Top-J query for Ebook2.txt (ebook 7178). CMD options: -t life,10 -m ____

         FOR COSINE:         FOR EUC:              FOR EUCNORM:
         even : 0.8889       love    : -126.94   even    : -0.471
         might : 0.8849      sinc    : -128.10     might   : -0.479
         without : 0.8823    mind    : -129.08     without : -0.485
         never : 0.8791      thought : -130.11     never   : -0.492
         love : 0.8757       must    : -130.83     love    : -0.4986
         now : 0.8752        long    : -132.04     now     : -0.4994
         sinc : 0.8746       feel    : -132.07     sinc    : -0.5006
         on : 0.8741         made    : -132.20     on      : -0.5017
         see : 0.8739        thing   : -133.75     see     : -0.5022
         seem : 0.8736       though  : -134.87     seem    : -0.5027
```

So far, Cosine and Eucnorm have similar results. At first, I thought a formula was wrong; however, turns out cosine and norm
euc distance keep matrix similarity the same! It helped to visualize two 2D lines and run the forulas on them. The method
for calculating cosine is more efficient than the normal euc distance method, so if efficiency was desired the cosine method
would be used (although, this does not matter that much since the matrix init takes far more time and the hashes are O[1]).

Ashley Hoffman
Matthew Wipfler

PART 6:

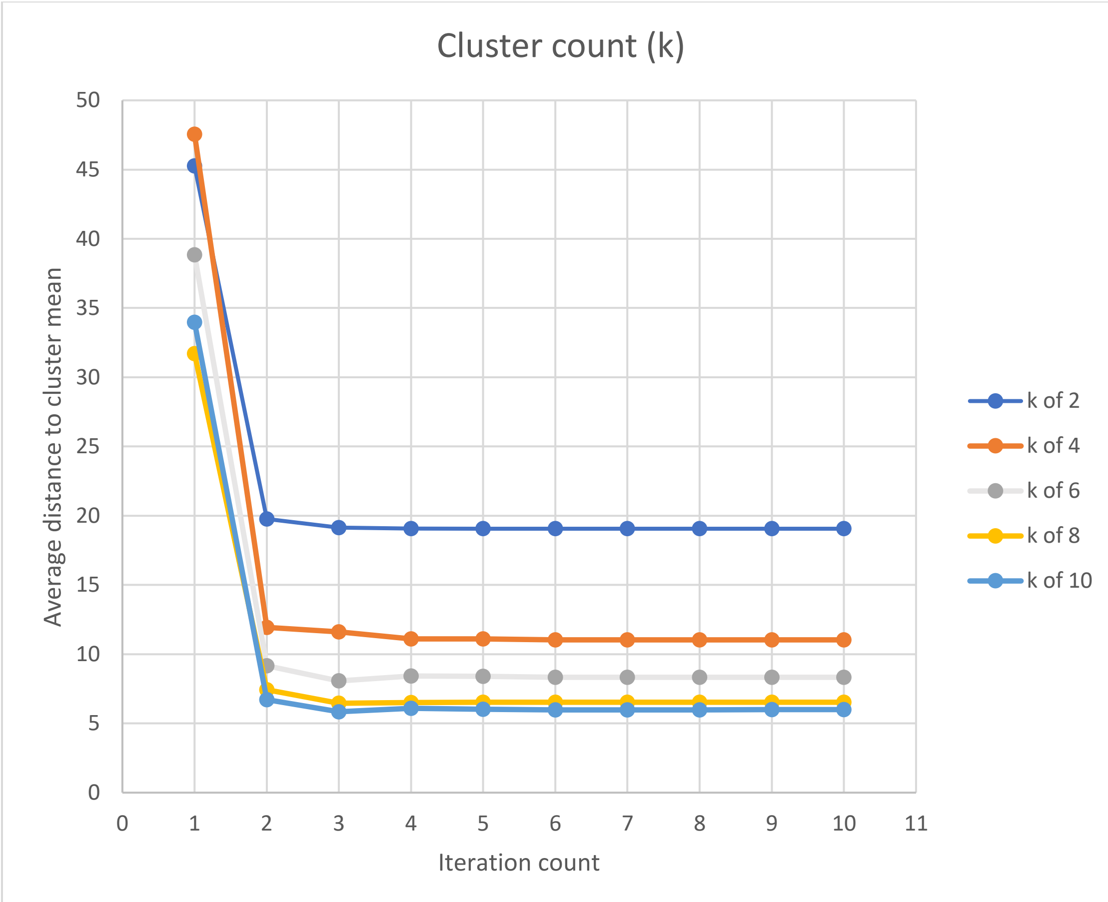


Ran with -k 5,7 (5 clusters, 7 iterations) twice for various text files.

Clusters started converging right away. After the third iteration, the average barely changed between iterations. Note that the cluster algorithm works with any size text file. Raw data below.

Iteration	Sanity Test		Ebook1		Ebook2		Ebook3	
1	0.7256	0.7001	12.71	9.369	28.57	37.525	47.725	55.881
2	1.1356	0.7809	5.122	4.538	10.065	10.439	12.024	12.066
3	1.092	0.7534	5.03	4.552	9.426	9.393	11.112	11.088
4	1.092	0.7534	4.994	4.53	9.657	9.354	11.037	11.303
5	1.092	0.7534	4.987	4.527	9.619	9.286	10.93	11.274
6	1.092	0.7534	4.988	4.52	9.556	9.116	10.916	11.236
7	1.092	0.7534	4.989	4.482	9.397	9.108	10.914	11.175

Ashley Hoffman
Matthew Wipfler



Text file: ebook2.txt

Each cluster started evening out around the 3rd iteration. There were slight differences afterwards, but the word changes causing the size differences were probably the boundary words far away from the cluster mean. An interesting note: k of 2 converged the quickest, while k of 4 converged with the most iterations. Raw values below

Iteration	Kvalues->	2	4	6	8	10
1		45.256	47.542	38.85	31.709	33.967
2		19.77	11.931	9.156	7.428	6.723
3		19.137	11.615	8.067	6.455	5.844
4		19.063	11.104	8.427	6.509	6.094
5		19.062	11.112	8.411	6.532	6.021
6		19.062	11.037	8.335	6.531	5.986
7		19.062	11.035	8.334	6.533	5.985
8		19.062	11.032	8.334	6.533	5.985
9		19.062	11.033	8.334	6.533	5.99
10		19.062	11.033	8.334	6.533	6

Ashley Hoffman
Matthew Wipfler

PART 7:

NOTE: Due to random start words, results on small texts (easy_sample_test.txt) can vary. Usually the numbers, animals, and food are split up into 3 distict groups, but if the start words overlap, they two groups can be in the same cluster.

FILE: ebook2.txt : -k 8,5,8 (8 clusters, 5 iterations, 8 top-J words)

****

Run 1: Random populate mean words: swanninasmuch, balm, policeserg, tone, bubbl, unspeak, gormandis, care

Cluster 1: --cambremerml, 2, musset, excitedli, coffeeandpistachio, frog, fabl, 5,

Cluster 2: --kriss, bourgeoi, pernici, malayan, reintroduc, bleakli, opium, china,

Cluster 3: --chap, redtil, herbinger', kitchengarden, no, overslept, bitterli, anodyn,

Cluster 4: --dislik, brougham, whoever, untroubl, rival, spasm, reproach, unduli,

Cluster 5: --devil, orgfundrais, blatin, hallo, piperaud, gaoler, orglicens, bravo,

Cluster 6: --grandpapa, txt, bailiff, eighteenth, rogationdai, ak, fairbank, 99712,

Cluster 7: --gregori, sonatasnak, serpentàson, serpentàsonnett, unbeliev, firstrat, sazerin, unaffect,

Cluster 8: --peevish, rejoin, pose, codfish, connoisseur, univers, providers, eloqu,

****

Run 2: Random populate mean words: upstair, steepli, closepack, moreand, state, dieth, poke, galopin,

Cluster 1: --2, sonatasnak, countri, cousins, unaffect, goodbyethere, nightcap, glum,

Cluster 2: --devil, bravo, orgfundrais, waterbutt, blatin, hallo, piperaud, k,

Cluster 3: --gregori, state, swamp, unenforc, provis, cabourg, diepp, unarm,

Cluster 4: --cambremerml, rejoin, peevish, codfish, pose, excitedli, coffeeandpistachio,

Cluster 5: --dislik, brougham, whoever, untroubl, rival, spasm, reproach, unduli,

Cluster 6: --grandpapa, bailiff, txt, eighteenth, rogationdai, ak, fairbank, 99712,

Cluster 7: --sadists, wickedli, avaric, lulli, maintenon, shrewd,

Cluster 8: --chap, redtil, herbinger, kitchengarden, no, overslept, bitterli, anodyn,

Ashley Hoffman
Matthew Wipfler

Part 7 writing part: Each run did not have the random starting words in the top-j results. Also, several clusters contained similar words between Run 1 and Run 2. Those are highlighted above. It is interesting how the clusters can be the same or very similar even with random starting points. So, in most cases, the uniqueness/words contained in each cluster would depend on k, the number of clusters, not on starting points. There might be some outliner groups that would could form their own cluster.