CMP-5014Y Coursework Assignment 1

100263247 (uxk18qau)

Wed, 4 Dec 2019 12:03

PDF prepared using PASS version 1.17 running on Windows 10 10.0 (amd64).

☑ I agree that by submitting a PDF generated by PASS I am confirming that I have checked the PDF and that it correctly represents my submission.

Contents

Coursework1.pdf

 (Included PDF starts on next page.)

Coursework1

dan.willis.uni

November 2019

1 PartA.1

Algorithm 1 calculateFeatureVector(**A**[],**Q**[]) **return** F

Require: a Dictionary **Q**
Require: a Document **A**
Require: an Empty array **F**
Require: Tree **a**

```
1: for i ← 1 to w do                          ▷ for every word
2:     a.add(A[i])                            ▷ add to tree
3: for i ← 1 to s do                          ▷ for every word in dictionary
4:     F[i] ← a.countNode(Q[i])               ▷ count
   return F
```

Algorithm 2 add(**str**)

Require: *str*

```
1: root ← addrecursive(root,str)   ▷ Goes into recursive to add to the correct
   place
```

2 PartA.2

This is the stripped down psuedo code for the algorithm This means that the algorithem has a run time complexity of w(addRecursive) + s(countNode) in the best, worst and average case. It fully depends upon the other algorithems as to how quick it is. There are n nodes in the tree
The hmax the maximum height of the tree is n where each node only has one child
The havg is the average height of the tree which is log(n)
The hmin is the minimum height of the tree which is 1 where the root only has one child so the other child is free unless the tree is empty in which case it is 0
addRecursive Time complexity:

Algorithm 3 addRecursive(**currentNode**,**str**)**return** *node*

Require: current Node infomation *currentNode*
Require: String to search for *str*
1: **if** *currentNode* = *null* **then** ▷ *if the node being looked at is null* **return** new *Node* ▷ *Return the null node to be populated*
2: **if** *currentNode* > *str* **then** ▷ *if current node greater than the search string*
3: *currentNodeLeftChild* ← addRecursive(*currentNodeLeftChild*,*str*) ▷ *search left*
4: **if** *currentNode* = *str* **then** ▷ *if they are equal*
5: *currentNodeLeftChild* ←addRecursive(*currentNodeLeftChild*,*str*) ▷ *search left*
6: **if** *currentNode* < *str* **then** ▷ *if node less than string*
7: *currentNodeRightChild* ← addRecursive(*currentNodeRightChild*,*str*) ▷ *search right*
return *currentNode*

Algorithm 4 countNode(**str**)**return** *num*

Require: string to search for *str*
Require: *num* ← 0 ▷ *set the number to 0*
1: countNodeRecursive(*root*,*str*) ▷ *count recursive* **return** *num*

Algorithm 5 countNodeRecursive(**currentNode**,**str**)**return** *num*

Require: *num*
Require: Current Node infomation **currentNode**
Require: String to search for **str**
1: **if** *currentNode*! = *null* **then** ▷ *if current node not null*
2: **if** *currentNode* < *str* **then** ▷ *if node less than string*
3: countNodeRecursive(*currentNodeRightChild*,*str*) ▷ *count right*
4: **else**
5: **if** *currentNode* = *str* **then** ▷ *if eqaul*
6: *num* ← *num* + 1 ▷ *add one to count*
7: countNodeRecursive(*currentNodeLeftChild*,*str*) ▷ *search left*

Algorithm 6 calculateFeatureVectorStripped(**A**[],**Q**[]) **return** *F*

1: **for** *i* ← 1 to *w* **do** ▷ *for every word*
2: *root* ← addrecursiveStripped(*root*,*A*[*i*]) ▷ *Goes into recursive to add to the correct place*
3: **for** *i* ← 1 to *s* **do** ▷ *for every word in dictionary*
4: *F*[*i*] ← *a*.countNodeStripped(**Q**[**i**]) ▷ *count*
return *F*

Algorithm 7 addRecursiveStripped(**currentNode**,**str**)**return** *node*

1: **if** *currentNode* = *null* **then** ▷ *if current node null* **return** new *Node* ▷ *return node to be populated*
2: *Child* ← addRecursiveStripped(*Child*,*str*) ▷ *child node = recursive of that child* **return** *currentNode*

Worst Case is O(hmax)
Average case is O(havg)
Best Case is O(hmin)
This is because it has to traverse the tree until it finds a null child and then add there
CountNode Time Complexity:

Algorithm 8 countNodeRecursiveStripped(**currentNode**,**str**)**return** *num*

1: **if** *currentNode*! = *null* **then** ▷ *if node not null*
2: countNodeRecursive(*Child*,*str*) ▷ *count child*

Worst Case is O(hmax)
Average Case is O(havg)
Best Case is O(hmin)
Because it traverses the depth of the tree

Worst Case:

$$\sum_{n=1}^{w} n + \sum_{n=1}^{s} n$$

$= \frac{1}{2}w\,(w+1) + \frac{1}{2}s\,(s+1)$
O($w^2 + s^2$)
Average Case:

$$\sum_{n=1}^{w} log(n) + \sum_{n=1}^{s} log(n)$$

$= log(w!) + log(s!)$
$= wlog(w) + slog(s)$
O($wlog(w) + slog(s)$) Best Case:

$$\sum_{n=1}^{w} 1 + \sum_{n=1}^{s} 1$$

$= w + s$
O(w+s)

3 PartA.3

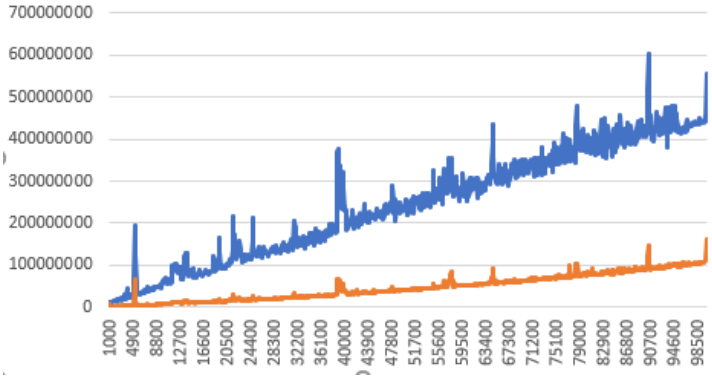
See Java Code

4 PartA.4

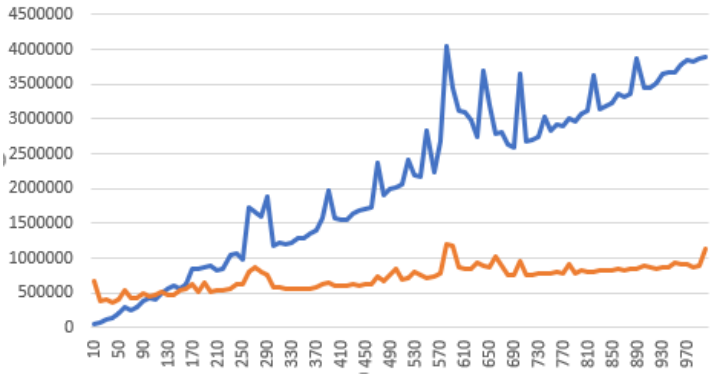
The way that I conducted all the tests was for every test I ran I did it five times and then averaged those five times. For each of the five times, a different set of inputs was created and fed into both the algorithms. Then the output was checked to be the same by both. I did all the times in nanoseconds as this was the most accurate I could make it. All the tests ran overnight at the same program one after another and all the results of all the tests had no difference between the calculations.

Figure 1: Feature Vector Test Results

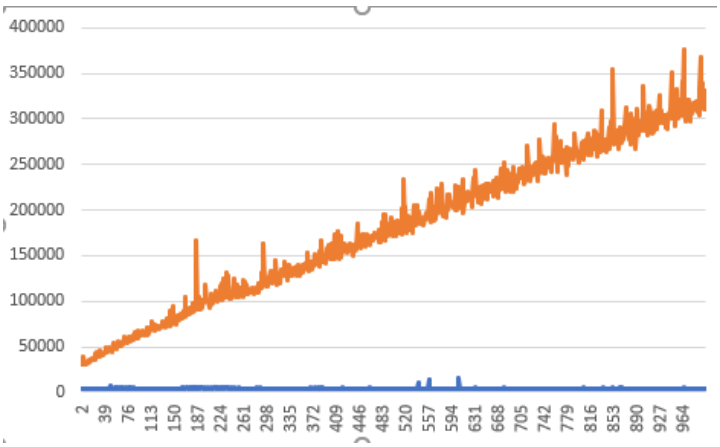
(a) Test One Results



(b) Test Two Results



(c) Test Three Results



1a is a graph of the first test I did which was varying the number of words in the document (w) from 1000 to 100000 incrementing by 1000 each time. This gives me 100 pieces of data. The lower line is the algorithm I used in the end and the other line represents the brute force algorithm. As you can see the gradient of the line representing my algorithm is much less than the gradient of the other line. This means that it is far quicker for a sufficiently large w. The other values were that the dictionary had 1000 words and all the words were 5 letters long.

1b is the results of test two in which I varied the size of the dictionary (s) from 10 to 1000 incrementing by 10 each time again giving me 100 pieces of data. Initially, the brute force algorithm is faster than mine for the lower dictionary size and is faster up until a dictionary size of 130 words. While my algorithm starts slightly higher its gradient is far less than the brute force algorithm so that begins to take longer. This means that for a sufficiently large s it is faster. The other values were 1000 words in the document all of which are 5 characters long.

1c is the results of test three in which I varied the length of the words in the dictionary and the document. From 2 to 1000 incrementing by 1 each time. For this test brute force was quicker every time as the time taken to complete the task doesn't depend upon the length of the words. Whereas my algorithm does because it sorts the words into order and the longer they are the longer it takes to sort them. Therefore my algorithm took longer and had a higher gradient so for a sufficient word length the brute force algorithm would be quicker. The other values were a dictionary size of 10 and a document length of 100.

Test four doesn't have a graph as there is only one piece of data per algorithm. This was just testing the difference in time when the dictionary size and document size were both set high and the word length was kept at 5. The dictionary size was 10000 and the document size was 1000000 this took the brute force algorithm on average 213381 ms. And took my algorithm 3159.700740 ms which is 67.5 times faster.

5 PartB.5

Algorithm 9 calculateDSD(**A**[], **B**[])**return** *DSD*

Require: *DSD*
Require: Feature Vector A **A**[]
Require: Feature Vector B **B**[]
1: $DSD \leftarrow 0$
2: **for** $i \leftarrow 1$ to s **do** ▷ *for each dictionary word*
3: $DSD \leftarrow DSD + —A[i] - B[i]—$ ▷ *add absolute of A-B*
return *DSD*

6 PartB.6

Algorithm 10 findNearestDocument(**Docs**[][], **Q**[])**return** $FND[]$

Require: An Array of Documents *Docs*
Require: A dictionary Q
Require: A Empty Array FND
Require: Number Of Doucment n
Require: A Distance array $Distance[n][n]$
Require: Feature Vector Array $F[n][]$

```
for A ← to n do                                          ▷ for Each Document
    F[A] ← calculateFeatureVector(Docs[A],Q)             ▷ calculate the feature vector
    distance[A][A] ←high                                 ▷ set distance to itself high
for A ← 1 to n do                                        ▷ for each document
    currentBestDistance ←high                            ▷ set closest document to NULL
    currentBestIndex ← 0                                 ▷ set closest documenet to none
    for A2 ← 1 to n do                                   ▷ for each document
        if distance[A][A2] is null then                  ▷ if the distance is null
            distance[A][A2] ←calculateDSD(F[A],F[A2])    ▷ calculate DSD
            distance[A2][A] ← distance[A][A2]            ▷ set mirror
        if distance[A][A2] ¡ currentBestDistance then    ▷ if the document is closer than the closest so far
            currentBestDistance ← distance[A][A2]        ▷ set the current best distance
            currentBestIndex ← A2                        ▷ se the current best index
    FND[A] ← currentBestIndex                            ▷ set the return array
return FND
```

7 PartB.7

This is the stripped pseudo code
Time Complexity = n(calculateFeatureVector) + $\frac{n(n-1)}{2}$(calculateDSD)

Algorithm 11 findNearestDocumentStripped(**Docs**[][], **Q**[])**return** $FND[]$

```
for A ← to n do                                          ▷ for each document
    F[A] ← calculateFeatureVector(Docs[A],Q)             ▷ calculate F
for A ← to n do                                          ▷ for each document
    for A2 ← to n do                                     ▷ for each document
        if distance[A][A2] is null then                  ▷ if uncalculated
            distance[A][A2] ←calculateDSDStripped(F[A],F[A2])   ▷ calculate DSD
return FND
```

Time complexity analysis of calculate DSD All Best, Worst and Average Cases

Algorithm 12 calculateDSDStripped(**A**[], **B**[])**return** DSD

```
1: for i ← 1 to w do
2:     DSD ← DSD + —A[i] − B[i]—
   return DSD
```

are the same which is O(s)
Therefore
WorstCase:
$n(\frac{1}{2}w\,(w+1)+\frac{1}{2}s\,(s+1))+s(\frac{n(n-1)}{2})$
$=\frac{ns^2+n^2s+w^2n+wn}{2}$
$=\mathrm{O}(ns^2+n^2s+w^2n)$

Average Case:
$n(wlog(w)+slog(s))+s(\frac{n(n-1)}{2})$
$\mathrm{O}(n^2s)$

Best Case:
$n(w+s)+s(\frac{n(n-1)}{2})$
$=nw+\frac{n^2s+ns}{2}$
$\mathrm{O}(n^2s)$

8 PartB.8

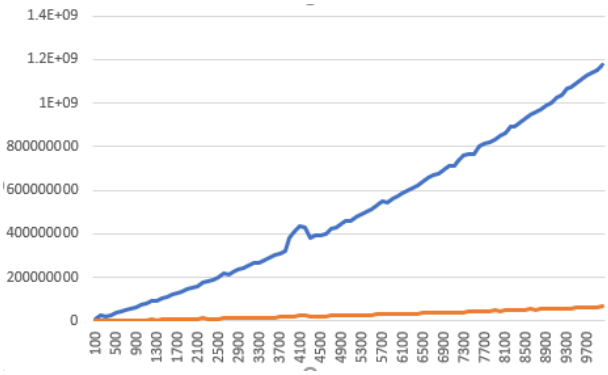
See Java Code

9 PartB.9

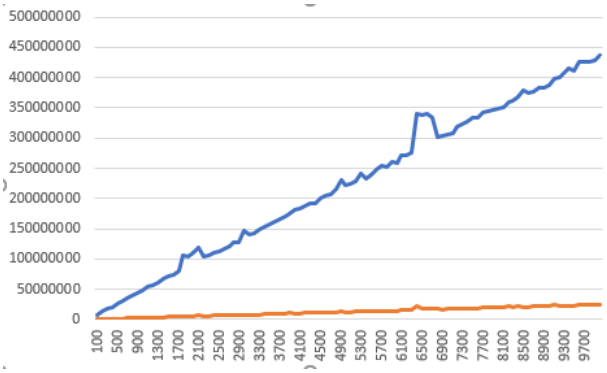
Both of the document similarity functions use the same algorithm to calculate the feature vectors so that it is just the algorithm that makes a difference.

Figure 2: Document Similarity Test Results

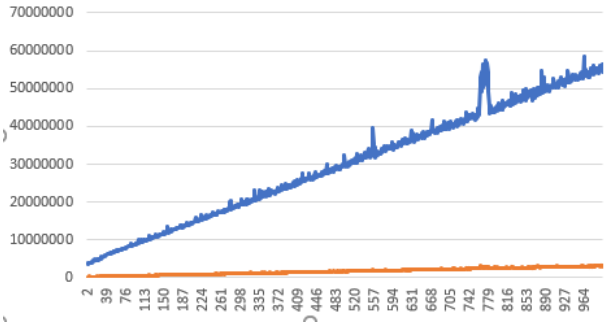
(a) Test Five Results



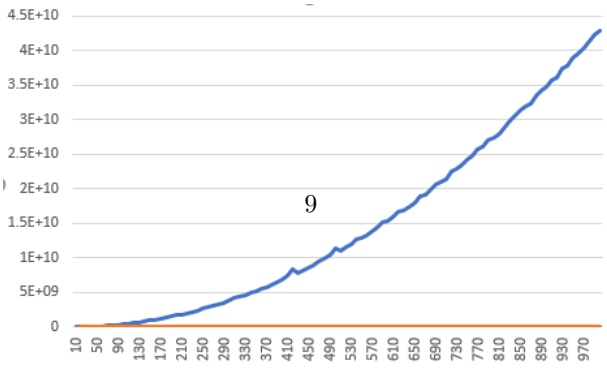
(b) Test Six Results



(c) Test Seven Results



(d) Test Eight Results



2a is the results of test fives, where I changed the document size from 100 to 10000 incrementing by100 each time. Other values where a dictionary size of 100, a word length of 5 and 10 documents. My algorithm is the lower line and has a much lower gradient than the brute force algorithm. This means that it is much faster and will be for a sufficiently large w.

2b is the results of test six, where is changed the dictionary size from 100 to10000 incrementing by 100 each time. The other values were a document size of 100, a word length of 5 and the number of documents being 10. The lower line is my algorithm and has a lower gradient meaning for a sufficiently large s it is still faster.

2c is test seven, where I changed the length of each word from 2 to 1000. The other values where 100 words in a document, 10 words in a dictionary and 10 documents. The gradient of the brute force algorithm is significantly more than my algorithm. So, for a significantly large length of words, my algorithm will be faster.

2d is test eight where I changed the number of documents from 10 to 1000 incrementing by 10 each time. The other values are a document length of 100, a dictionary length of 10 and a word length of 5. The gradient of the brute force has a definite curve of a quadratic which makes sense whereas mine looks relatively linear. And is significantly lower gradient meaning that for a significantly large number of documents my algorithm is faster.

Test nine is just two pieces of data with 100 documents, a dictionary size of 10, word length of 5 and the number of documents being 10000. This took the brute force algorithm on average 4297.7161401 seconds and took my algorithm 4.8981888 seconds on average. This is 877.4 times faster.

BinaryTree.java

```
1  package com.company;
2
3  class BinaryTree {
4      private Node root;
5      private int StringCounter;
6      //adding an element
7      void add(String value) {
8          root = addRecursive(root, value);
9      }
10     private Node addRecursive(Node current, String value) {
11         if (current == null) {
12             return new Node(value);
13         }
14         if (value.compareToIgnoreCase(current.value) <= 0) {
15             current.left = addRecursive(current.left, value);
16         } else if (value.compareToIgnoreCase(current.value) > 0) {
17             current.right = addRecursive(current.right, value);
18         }
19         return current;
20     }
21     //searching for an element
22     int countNode(String value){
23         StringCounter = 0;
24         countNodeRecursive(root,value);
25         return StringCounter;
26     }
27     private void countNodeRecursive(Node current, String value) {
28         if (current != null) {
29             if (value.compareToIgnoreCase(current.value)>0){
30                 countNodeRecursive(current.right, value);
31             } else {
32                 if (value.compareToIgnoreCase(current.value)==0){
33                     StringCounter++;
34                 }
35                 countNodeRecursive(current.left, value);
36             }
37         }
38     }
39     private void traverseInOrder(Node node) {
40         if (node != null) {
41             traverseInOrder(node.left);
42             System.out.print(" " + node.value);
43             traverseInOrder(node.right);
44         }
45     }
46 }
47 class Node {
48     String value;
49     Node left;
50     Node right;
51
52     Node(String value) {
53         this.value = value;
54         right = null;
55         left = null;
56     }
57 }
```

Main.java

```
1  package com.company;
   //import static com.company.CourseworkUtilities.generateDocument;
3  public class Main {
       public static void main(String[] args) {
5
       }
7      private static int[] calculateFeatureVector(String[] A, String[] Q){
           //returns feature vector of given document A and dictionary Q using a
               tree
9          int[] F = new int[Q.length];//initilises the array to return
           BinaryTree a = new BinaryTree();//creates the tree
11         for (String word : A) {//for every word in the document
               a.add(word);//add the word to the tree
13         }
           for (int s = 0;s<Q.length;s++){//for every word in dictionary
15             F[s]=a.countNode(Q[s]);//feature vector array of that word = the word
                   counted in the tree
           }
17         return F;//return the feature vector array
       }
19     private static int calculateDSD(int[] A, int[] B){
           //calculates the Document similarity distance and returns it
21         int DSD = 0;//sets the document similarity distance to 0
           for (int w =0;w<A.length;w++){//for every word in document
23             DSD=DSD + Math.abs(A[w] - B[w]);//add the absolute value of documentA
                   feature vector subtract documentB feature vector to document
                   similarity
           }
25         return DSD;//return document similarity distance
       }
27     public static int[] findNearestDocuments(String[][] Doc, String[] Q){
           int[] FND = new int[Doc.length];//initilise the find nearest document
               array
29         int[][] distance = new int[Doc.length][Doc[0].length];//initilise the
               document
           int[][] F = new int[Doc.length][];//initilise the feature vector array
31         for (int A = 0;A<Doc.length;A++){//for each document
               F[A] = calculateFeatureVector(Doc[A],Q);//calculate the feature
                   vector and populate F array
33             for (int Ax = 0;Ax<Doc.length;Ax++){//for each docuemnt
                   distance[A][Ax]=-1;//make the distance = -1
35             }
               distance[A][A] = Integer.MAX_VALUE;//set each distance to itself to
                   max value
37         }
           for (int A = 0;A<Doc.length;A++){//for every document
39             int currentBestDistance = Integer.MAX_VALUE;//set current best
                   distance high
               int currentBestIndex = -1;//sets current best index to -1
41             for (int Ax = A;Ax<Doc.length;Ax++){//for every document
                   if(distance[A][Ax] == -1){//if untouched
43                     distance[A][Ax] = calculateDSD(F[A],F[Ax]);//calculate
                           distance
                       distance[Ax][A] = distance[A][Ax];//set mirror
45                 }
                   if (distance[A][Ax] < currentBestDistance){//if the checked
                       distance is less than the current best
47                     currentBestIndex = Ax;//set the current best document
                       currentBestDistance = distance[A][Ax]; //set the current best
                           distance
49                 }
```

```
                }
51              FND[A] = currentBestIndex;//set the closest documents array
            }
53          return FND;//return closest documents array
        }
55  }
```

MainTest.java

```
1  package com.company;
2
3  import java.io.*;
4  import java.util.ArrayList;
5  import java.util.List;
6
7  import static com.company.CourseworkUtilities.generateDocument;
8
9  public class Main {
10
11     public static void main(String[] args) {
12         //runs all the tests
13         TestOne();
14         TestTwo();
15         TestThree();
16         TestFour();
17         TestFive();
18         TestSix();
19         TestSeven();
20         TestEight();
21         TestNine();
22         //averages all the tests
23         String[][] FileNames = {{"FVBruteTestOne.csv","FVTreeTestOne.csv"},{"
               FVBruteTestTwo.csv","FVTreeTestTwo.csv"},{"FVBruteTestThree.csv","
               FVTreeTestThree.csv"},{"FVBruteTestFour.csv","FVTreeTestFour.csv"},{"
               DSDBruteTestFive.csv","DSDMineTestFive.csv"},{"DSDBruteTestSix.csv","
               DSDMineTestSix.csv"},{"DSDBruteTestSeven.csv","DSDMineTestSeven.csv"
               },{"DSDBruteTestEight.csv","DSDMineTestEight.csv"},{"DSDBruteTestNine
               .csv","DSDMineTestNine.csv"}};
24         Average(FileNames);
25     }
26     //Tests
27     private static void TestOne() {
28         for (int i = 1000; i<=100000;i=i+100){
29             for (int t = 0;t<5;t++) {
30                 try {
31                     String[] Q = CourseworkUtilities.generateDictionary(1000,5);
32                     String[] A = generateDocument(Q,i);
33                     int[] Brute  = TestCalFV("Brute","A","TestOne",A,Q,t);
34                     int[] Tree  = TestCalFV("Tree","A","TestOne",A,Q,t);
35                     if (Comparison(Brute, Tree)!=-1){
36                         System.out.println("Error: " + i + " Brute & Tree FV is
                               different");
37                     }
38                 } catch (Exception e) {
39                     e.printStackTrace();
40                 }
41
42             }
43
44         }
45         System.out.println("TestOne Complete");
46     }
47     private static void TestTwo() {
48         for (int i = 10; i<=1000;i=i+10){
49             for(int t = 0;t<5;t++) {
50                 try {
51                     String[] Q = CourseworkUtilities.generateDictionary(i, 5);
52                     String[] A = generateDocument(Q,1000);
53                     int[] Brute  = TestCalFV("Brute","Q","TestTwo",A,Q,t);
54                     int[] Tree  = TestCalFV("Tree","Q","TestTwo",A,Q,t);
```

```
55                      if (Comparison(Brute, Tree)!=-1){
                            System.out.println("Error: " + i + " Brute & Tree FV is
                                different");
57                      }
                    } catch (Exception e) {
59                      e.printStackTrace();
                    }
61              }
            }
63          System.out.println("TestTwo Complete");
        }
65      private static void TestThree() {
            for (int i = 2; i<=1000;i++){
67              for (int t = 0;t<5;t++){
                    try {
69                      String[] Q = CourseworkUtilities.generateDictionary(10,i);
                        String[] A = generateDocument(Q,100);
71                      int[] Brute  = TestCalFV("Brute","w","TestThree",A,Q,t);
                        int[] Tree  = TestCalFV("Tree","w","TestThree",A,Q,t);
73                      if (Comparison(Brute, Tree)!=-1){
                            System.out.println("Error: " + i + " Brute & Tree FV is
                                different");
75                      }
                    } catch (Exception e) {
77                      e.printStackTrace();
                    }
79              }
            }
81          System.out.println("TestThree Complete");
        }
83      private static void TestFour() {
            for (int t = 0;t<5;t++){
85              try {
                    String[] Q = CourseworkUtilities.generateDictionary(10000,5);
87                  String[] A = generateDocument(Q,1000000);
                    int[] Brute  = TestCalFV("Brute","N","TestFour",A,Q,t);
89                  int[] Tree  = TestCalFV("Tree","N","TestFour",A,Q,t);
                    if (Comparison(Brute, Tree)!=-1){
91                      System.out.println("Error: Brute & Tree FV is different");
                    }
93              } catch (Exception e) {
                    e.printStackTrace();
95              }
            }
97          System.out.println("TestFour Complete");
        }
99      private static void TestFive() {
            for (int i = 100;i<=10000;i=i+100){
101             for (int t = 0;t<5;t++){
                    try {
103                     String [][] Docs = new String[10][i];
                        String[] Q = CourseworkUtilities.generateDictionary(100,5);
105                     for (int o = 0; o<10;o++){
                            String[] A = generateDocument(Q,i);
107                         Docs[o] = A;
                        }
109                     int[] BruteDSD = TestCalfindNearestDocument("Brute","A","
                            TestFive",Docs, Q, t);
                        int[] MineDSD = TestCalfindNearestDocument("Mine","A","
                            TestFive",Docs, Q, t);
111                     if (Comparison(BruteDSD, MineDSD)!=-1){
                            System.out.println("Error: " + i + " Brute & Mine DSD is
                                different");
```

```
113                     }
                    } catch (Exception e) {
115                     e.printStackTrace();
                    }
117             }
            }
119         System.out.println("TestFive Complete");
        }
121     private static void TestSix() {
            for (int i = 100;i<=10000;i=i+100){
123             for (int t = 0;t<5;t++){
                    try {
125                     String [][] Docs = new String[10][100];
                        String[] Q = CourseworkUtilities.generateDictionary(i,5);
127                     for (int o = 0; o<10;o++){
                            String[] A = generateDocument(Q,100);
129                         Docs[o] = A;
                        }
131                     int[] BruteDSD = TestCalfindNearestDocument("Brute","Q","
                           TestSix",Docs, Q, t);
                        int[] MineDSD = TestCalfindNearestDocument("Mine","Q","
                           TestSix",Docs, Q, t);
133                     if (Comparison(BruteDSD, MineDSD)!=-1){
                            System.out.println("Error: " + i + " Brute & Mine DSD is
                               different");
135                     }
                    } catch (Exception e) {
137                     e.printStackTrace();
                    }
139             }
            }
141         System.out.println("TestSix Complete");
        }
143     private static void TestSeven() {
            for (int i = 2;i<=1000;i++){
145             for (int t = 0;t<5;t++){
                    try {
147                     String [][] Docs = new String[10][100];
                        String[] Q = CourseworkUtilities.generateDictionary(10,i);
149                     for (int o = 0; o<10;o++){
                            String[] A = generateDocument(Q,100);
151                         Docs[o] = A;
                        }
153                     int[] BruteDSD = TestCalfindNearestDocument("Brute","w","
                           TestSeven",Docs, Q, t);
                        int[] MineDSD = TestCalfindNearestDocument("Mine","w","
                           TestSeven",Docs, Q, t);
155                     if (Comparison(BruteDSD, MineDSD)!=-1){
                            System.out.println("Error: " + i + " Brute & Mine DSD is
                               different");
157                     }
                    } catch (Exception e) {
159                     e.printStackTrace();
                    }
161             }
            }
163         System.out.println("TestSeven Complete");
        }
165     private static void TestEight() {
            for (int i = 10;i<=1000;i=i+10){
167             for (int t = 0;t<5;t++){
                    try {
169                     String [][] Docs = new String[i][100];
```

```
                    String[] Q = CourseworkUtilities.generateDictionary(10,5);
                    for (int o = 0; o<i;o++){
                        String[] A = generateDocument(Q,100);
                        Docs[o] = A;
                    }
                    int[] BruteDSD = TestCalfindNearestDocument("Brute","D","
                        TestEight",Docs, Q, t);
                    int[] MineDSD = TestCalfindNearestDocument("Mine","D","
                        TestEight",Docs, Q, t);
                    if (Comparison(BruteDSD, MineDSD)!=-1){
                        System.out.println("Error: " + i + " Brute & Mine DSD is
                            different");
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        System.out.println("TestEight Complete");
    }
    private static void TestNine() {
        for (int t = 0;t<5;t++){
            try {
                String [][] Docs = new String[10000][100];
                String[] Q = CourseworkUtilities.generateDictionary(10,5);
                for (int i = 0; i<10000;i++){
                    String[] A = generateDocument(Q,100);
                    Docs[i] = A;
                }
                int[] BruteDSD = TestCalfindNearestDocument("Brute","N","TestNine
                    ",Docs, Q, t);
                int[] MineDSD = TestCalfindNearestDocument("Mine","N","TestNine",
                    Docs, Q, t);
                if (Comparison(BruteDSD, MineDSD)!=-1){
                    System.out.println("Error: Brute & Mine DSD is different");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        System.out.println("TestNine Complete");
    }
    //Calculating feature vector functions
    private static int[] calculateFeatureVectorTree(String[] A, String[] Q){
        //returns feature vector of given document A and dictionary Q using a
            tree
        int[] F = new int[Q.length];//initilises the array to return
        BinaryTree a = new BinaryTree();//creates the tree
        for (String word : A) {//for every word in the document
            a.add(word);//add the word to the tree
        }
        for (int s = 0;s<Q.length;s++){//for every word in dictionary
            F[s]=a.countNode(Q[s]);//feature vector array of that word = the word
                counted in the tree
        }
        return F;//return the feature vector array
    }
    private static int[] calculateFeatureVectorBrute(String[] A, String[] Q){
        //returns feature vector array F, given a document A and dictionary Q
            using brute force
        int[] F = new int[Q.length];//initilises the feature vector array
        for (int s = 0;s<Q.length;s++) {//for each word in dictionary
            int counter = 0;//set counter to 0
```

```
225             for (int w = 0;w<A.length;w++) {//for every word in document
                    if (Q[s].equals(A[w])){//if dictionary word equals the document
                        word
227                      counter++;//increment the counter
                    }
229             }
                F[s] = counter;//after checking every word in the document, add the
                    counter to the feature vector array
231         }
            return F;//return the feature vector array
233     }
        private static int[] TestCalFV(String Algo,String Var,String FileName,String
            [] A,String[] Q,int t) {
235         //tests the time the algorithem takes to compleate and writes in a csv
                file
            int[] F;//creates the new array for feature vectors
237         long duration;
            if (Algo == "Brute"){//if the algorithem to be used is brute force
239             long startTime = System.nanoTime();//start timer
                F = calculateFeatureVectorBrute(A, Q);//calculate feature vector
                    using brute force
241             long endTime = System.nanoTime();//end timer
                duration = (endTime - startTime); //duration = the time taken in ns
243         } else { //else use tree
                long startTime = System.nanoTime();//start timer
245             F = calculateFeatureVectorTree(A, Q);//calculate feature vectors
                    using tree
                long endTime = System.nanoTime();//end timer
247             duration = (endTime - startTime); //duration = time taken in ns
            }
249         String NewFileName = "FV" + Algo + FileName + ".csv";//creates the file
                name string
            switch (Var) {//deciding which variable to print in the file
251             case "A"://the document length
                    csvAppend(NewFileName , t + "," + A.length + "," + duration + ",
                        ns");//append to the csv file the results
253                 break;
                case "Q"://the dictionary length
255                 csvAppend(NewFileName, t + "," + Q.length + "," + duration + ",ns
                        ");//append to the csv file the results
                    break;
257             case "w"://the length of the words
                    int l = A[0].length();//get the length of the words
259                 csvAppend(NewFileName, t + "," + l + "," + duration + ",ns");//
                        append to the csv file the results
                    break;
261             case "N"://none
                    csvAppend(NewFileName, t + "," + duration + ",ns");//append to
                        the csv file the results
263                 break;
                default:
265                 throw new IllegalStateException("Unexpected value: " + Var);//
                        unexpected value
            }
267         return F;//return the featur vector array for comparison
        }
269     //find the nearest document functions
        private static int calculateDSD(int[] A, int[] B){
271         //calculates the Document similarity distance and returns it
            int DSD = 0;//sets the document similarity distance to 0
273         for (int w =0;w<A.length;w++){//for every word in document
                DSD=DSD + Math.abs(A[w] - B[w]);//add the absolute value of documentA
                    feature vector subtract documentB feature vector to document
```

```
                similarity
275         }
            return DSD;//return document similarity distance
277     }
        public static int[] findNearestDocumentsMine(String[][] Doc, String[] Q){
279         int[] FND = new int[Doc.length];//initilise the find nearest document
                array
            int[][] distance = new int[Doc.length][Doc[0].length];//initilise the
                document
281         int[][] F = new int[Doc.length][];//initilise the feature vector array
            for (int A = 0;A<Doc.length;A++){//for each document
283             F[A] = calculateFeatureVectorTree(Doc[A],Q);//calculate the feature
                    vector and populate F array
                for (int Ax = 0;Ax<Doc.length;Ax++){//for each docuemnt
285                 distance[A][Ax]=-1;//make the distance = -1
                }
287             distance[A][A] = Integer.MAX_VALUE;//set each distance to itself to
                    max value
            }
289         for (int A = 0;A<Doc.length;A++){//for every document
                int currentBestDistance = Integer.MAX_VALUE;//set current best
                    distance high
291             int currentBestIndex = -1;//sets current best index to -1
                for (int Ax = A;Ax<Doc.length;Ax++){//for every document
293                 if(distance[A][Ax] == -1){//if untouched
                        distance[A][Ax] = calculateDSD(F[A],F[Ax]);//calculate
                            distance
295                     distance[Ax][A] = distance[A][Ax];//set mirror
                    }
297                 if (distance[A][Ax] < currentBestDistance){//if the checked
                        distance is less than the current best
                        currentBestIndex = Ax;//set the current best document
299                     currentBestDistance = distance[A][Ax]; //set the current best
                            distance
                    }
301             }
                FND[A] = currentBestIndex;//set the closest documents array
303         }
            return FND;//return closest documents array
305     }
        public static int[] findNearestDocumentsBrute(String[][] Doc, String[] Q){
307         //finds the nearest document using brute force
            int[] FND = new int[Doc.length];//initilise the nearest document array
309         for (int A = 0;A<Doc.length;A++){//for each document
                int closestIndex = -1;//set the closest index to -1
311             int closestDif = Integer.MAX_VALUE;//set the closest difference to
                    max value
                for (int Ax = 0;Ax<Doc.length;Ax++){//for each document
313                 if((Ax !=A) && (calculateDSD(calculateFeatureVectorTree(Doc[A],Q)
                        ,calculateFeatureVectorTree(Doc[Ax],Q)) < closestDif)){//if
                        the documents arent the same and the calculated document
                        distances is less than the current closest document distance
                        closestDif = calculateDSD(calculateFeatureVectorTree(Doc[A],Q
                            ),calculateFeatureVectorTree(Doc[Ax],Q));//set the
                            closest differance
315                     closestIndex = Ax;//set the closest index
                    }
317             }
                FND[A] = closestIndex;//populate the nearest document array
319         }
            return FND;//return the nearest document array
321     }
        private static int[] TestCalfindNearestDocument(String Algo,String Var,String
```

```
        FileName,String[][] Docs,String[] Q,int t) {
323     //tests the time and sends the time taken to a csv file
        int[] FND;//initilises the document distance array
325     long duration;//initilises duration long
        if (Algo == "Brute"){//if the test is on the brute force algorithem
327         long startTime = System.nanoTime();//start timer
            FND = findNearestDocumentsBrute(Docs,Q);//do the calculation using
                brute force
329         long endTime = System.nanoTime();//end timer
            duration = (endTime - startTime);//set duration
331     } else {//else
            long startTime = System.nanoTime();//start timer
333         FND = findNearestDocumentsMine(Docs,Q);//use my algorithem to do the
                calculation
            long endTime = System.nanoTime();//end timer
335         duration = (endTime - startTime); //set duration
        }
337     String NewFileName = "DSD" + Algo + FileName + ".csv";//creates new
            filename as string
        switch (Var) {//figures out which variable to print along with the
            duration
339         case "A"://doucment length
                csvAppend(NewFileName,  t + "," + Docs[0].length + "," + duration
                    + ",ns");//appends to the csvfile
341             break;
            case "Q" ://dictionary length
343             csvAppend(NewFileName,  t + "," + Q.length + "," + duration + ",
                    ns");//appends to the csvfile
                break;
345         case "w" :
                int l = Docs[0][0].length();//word length
347             csvAppend(NewFileName,  t + "," + l + "," + duration + ",ns");//
                    appends to the csvfile
                break;
349         case "D" ://document length
                csvAppend(NewFileName,  t + "," + Docs.length + "," + duration +
                    ",ns");//appends to the csvfile
351             break;
            case "N" ://none
353             csvAppend(NewFileName,  t + "," + duration + ",ns");//appends to
                    the csvfile
                break;
355         default:
                throw new IllegalStateException("Unexpected value: " + Var);//
                    unexpected value
357     }
        return FND;//return the nearest document array for comparison
359 }
    //File handling methods
361 private static void Average(String[][] FileNames) {
        //averages all the files and puts them in one file
363     for (int FileGroup = 0;FileGroup<FileNames.length;FileGroup++){//for each
            file group
            String[][] AllLines = new String[FileNames[FileGroup].length][];//
                initlise all the lines in all the files array
365         for (int FileInGroup = 0;FileInGroup<FileNames[FileGroup].length;
                FileInGroup++){//for each file in the group
                String[] Lines = ReadFile(FileNames[FileGroup][FileInGroup]);//
                    get all the lines
367             AllLines[FileInGroup] = Lines;//set the lines in file into an
                    array
            }
369         String NewFileName = "Average";//initilise file name string
```

```
            for (int LineInFile = 0;LineInFile<AllLines[0].length;LineInFile++){
                //for every line//for every line
371             String Line = "";//initilise line string
                for (int FileInGroup = 0;FileInGroup<AllLines.length;FileInGroup
                    ++){//for every file
373                 if(LineInFile==0){//if its the first line
                        String[] FileSplit = FileNames[FileGroup][FileInGroup].
                            split("\\.");//split the file into name and type
375                     NewFileName=NewFileName+FileSplit[0];//add the name to
                            the new file name
                    }
377                 String[] Values = AllLines[FileInGroup][LineInFile].split(","
                        );//split into cells
                    if(Values.length>1){//if there is actually data
379                     if (Values[0].charAt(0)=='0'){//if it is the first
                            iteration of it
                            if(Values.length==4 && FileInGroup==0){//add the
                                variable number if necessary
381                             Line=Line+Values[1] + ","; //getting the altered
                                    variable
                            }
383                         long total = 0;//set total time to 0
                            for (int PlusLine = 0;PlusLine<5;PlusLine++){//for
                                each line that is the same test
385                             String[] PlusValue = AllLines[FileInGroup][
                                    LineInFile+PlusLine].split(",");//split the
                                    line
                                total=total+Long.parseLong(PlusValue[PlusValue.
                                    length - 2]);//get the duration and add it to
                                     the total
387                         }
                            total=total/5;//total = the average
389                         Line=Line+total+",";//add the total to the line
                                string
                        }
391                 }
                }
393             if (Line != ""){//if the line is not empty
                    csvAppend(NewFileName+".csv",Line+"ns");//append to the csv
                        file
395             }
            }
397     }
    }
399 private static String[] ReadFile(String FileName){
        //reads the whole file and returns the lines in an array
401     File file = new File(FileName);//makes the file
        List<String> lines = new ArrayList<String>();//makes the line arraylist
403     BufferedReader in = null;//sets the buffer reader to null
        try {//try
405         in = new BufferedReader(new FileReader(file));//make the buffer
                reader
            String inputLine;//create a string for the line to be put into
407         while ((inputLine = in.readLine()) != null){//while the line isnt
                empty
                lines.add(inputLine);//add the line to the lines array list
409         }
            in.close();//close the file
411     } catch (FileNotFoundException e) {//catch for filenot found
            e.printStackTrace();//print error
413     } catch (IOException e) {//catch for exception
            e.printStackTrace();//print error
415     }
```

```
            return lines.toArray(new String[0]);//return the arraylist converted to
                an array
417     }
        private static void csvAppend(String FileName, String str){
419         //appends to the csv files
            try{//try
421             BufferedWriter writer = new BufferedWriter(new FileWriter(FileName,
                    true));//make a writer
                writer.append("\n");//write a new line
423             writer.append(str);//write the line
                writer.close();//close the file
425         } catch (IOException e) {//catch exception
                e.printStackTrace();//print error
427         }
        }
429
        private static int Comparison(int[] Brute,int[] Mine){
431         //compares the arrays
            for (int i = 0;i<Brute.length;i++){//for every element in the array
433             if (Brute[i]!=Mine[i]) {//if that element is not equal to the
                    corisponding element in ther other array
                     return i;//return the index of which element is wrong
435             }
            }
437         return -1;//returns -1 if they are the same array
        }
439 }
```