|  |
| --- |
| TMA3: Assignment 3 |
| BIT694 Designing Applications with c#  Due Date October 15 2017 |
| Penelope Ann Williamson  3431274 |

|  |
| --- |
|  |

Libraries for reading files:

* Code7248.word\_reader (code7248, 2012)
* iTextSharp (asubach, 2017)
* CSharpJExcel (grimholtz, 2012)

Porter Stemmer: (Patton, 2015)

Code for converter class: (Open Polytechnic, 2017)

### **Screen Shots:**

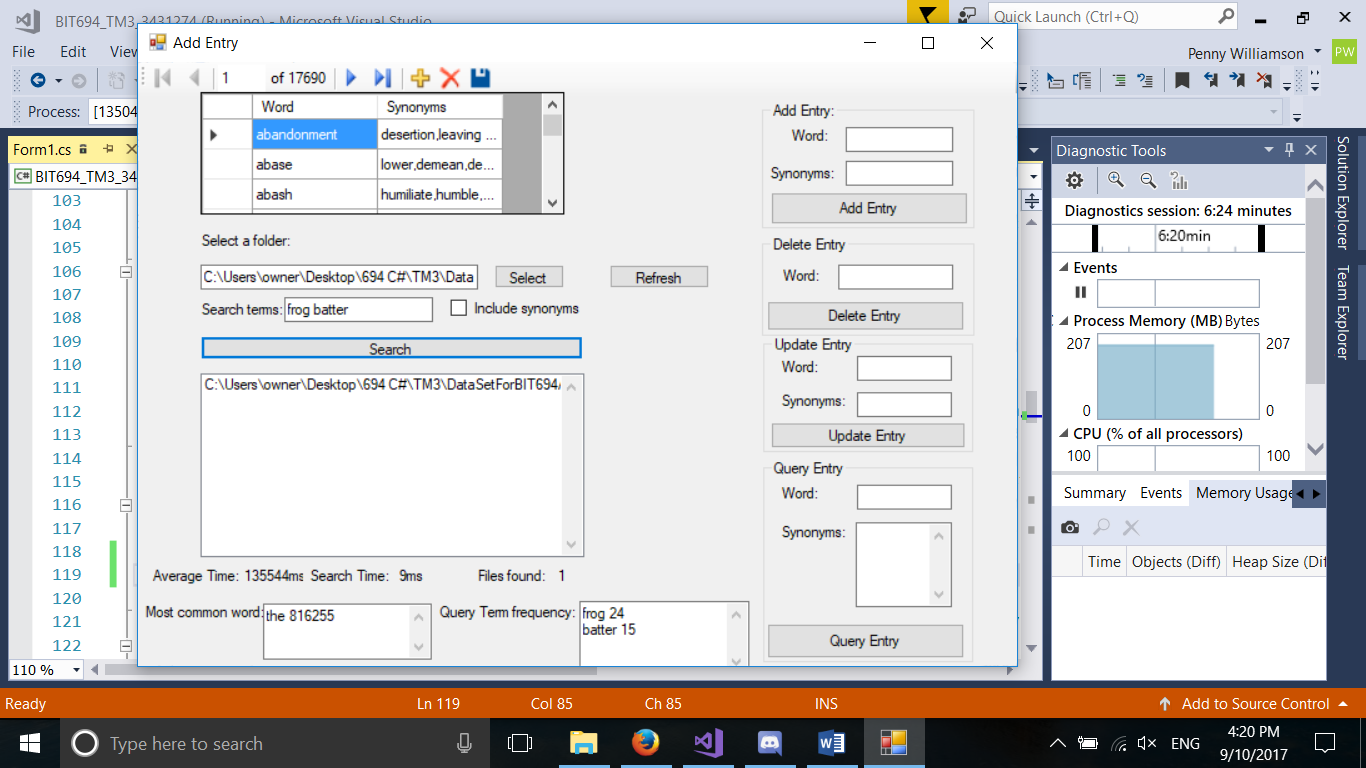


Figure 1. Screen shot of searching already built index.

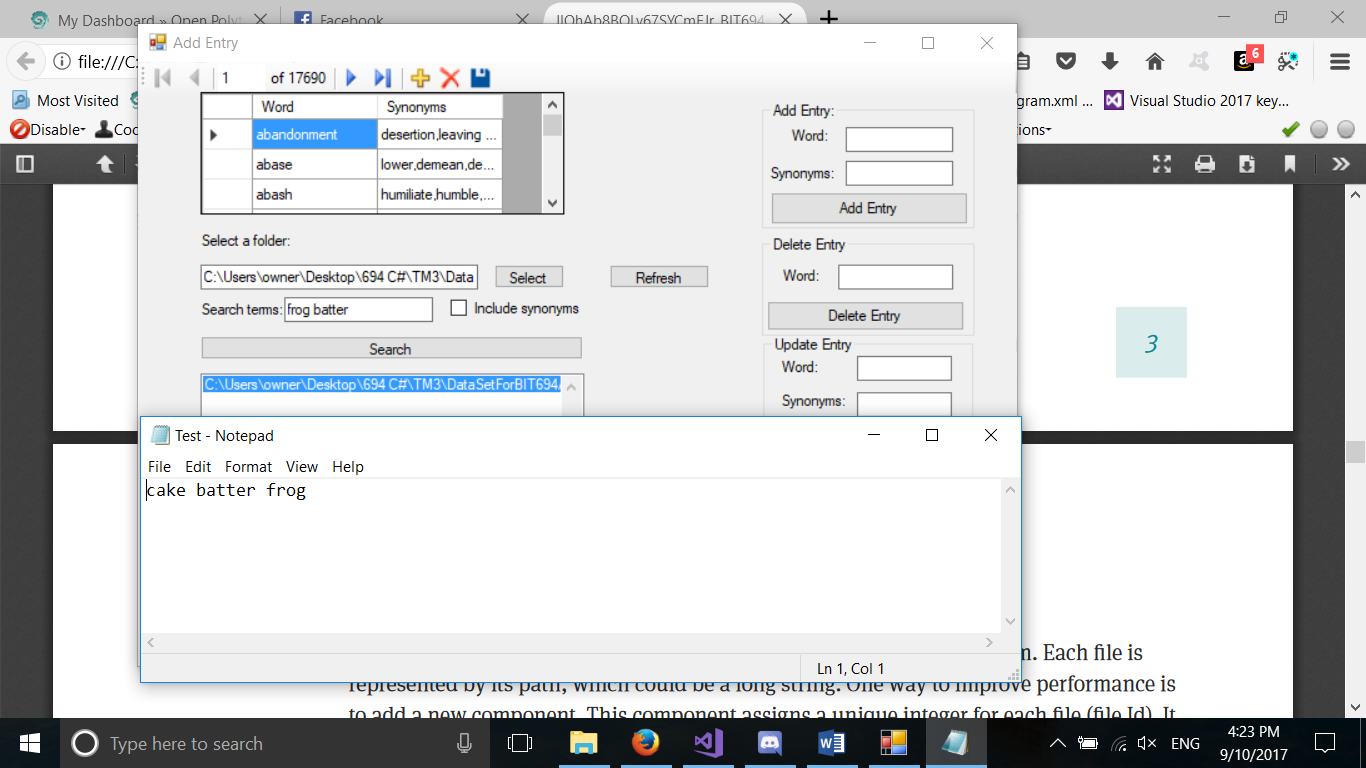


Figure 2. Opening the file from list.

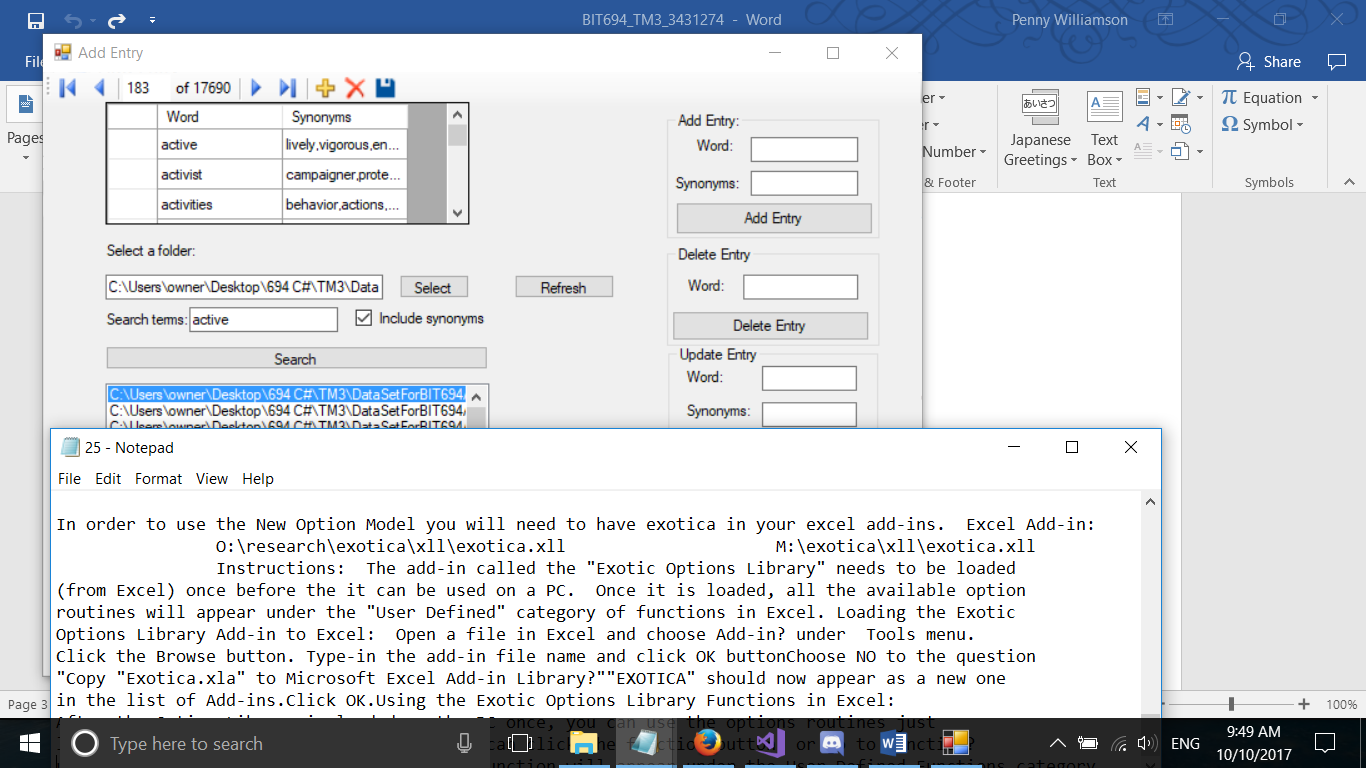


Figure 3. Searching with synonyms, and opening file. Search by order file one. Contains 14 synonyms, or stemmed words that stem the same.

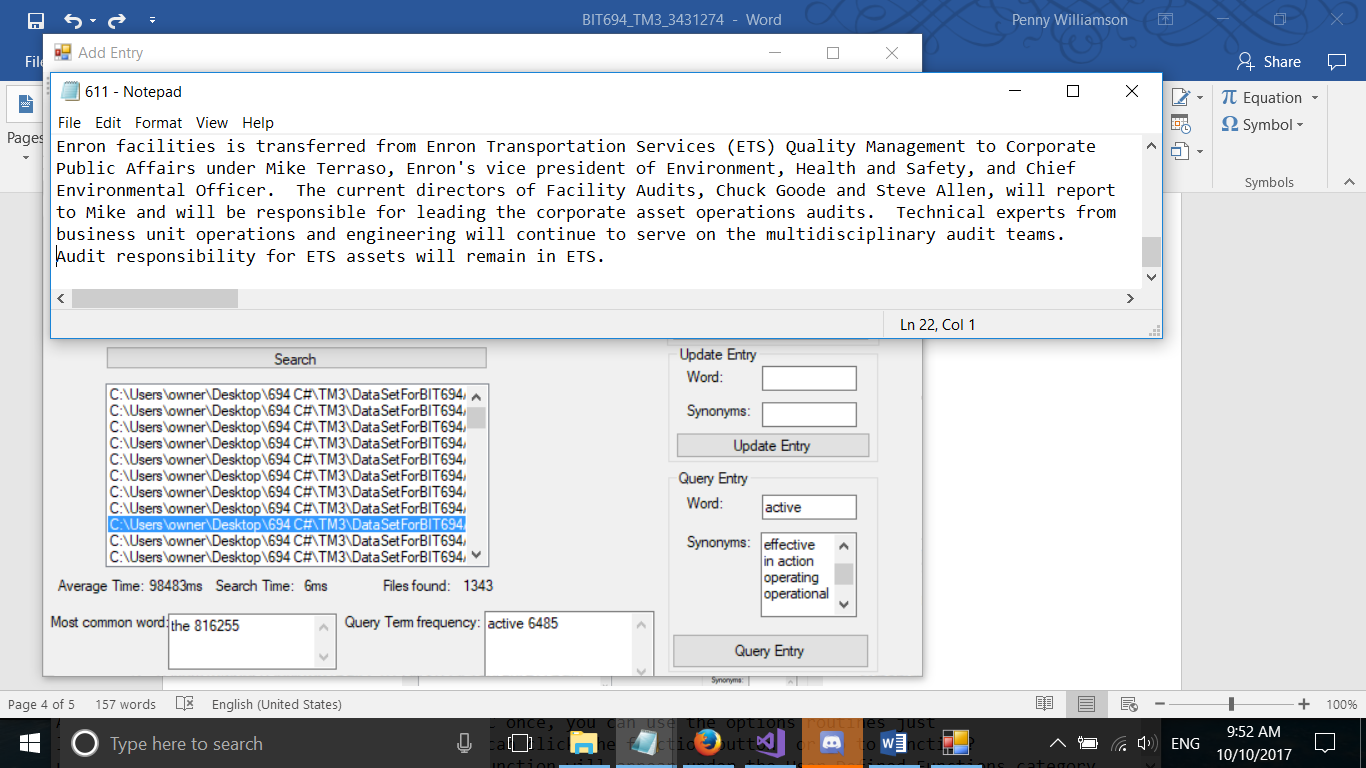


Figure 4. Results from order file two. Contains two synonyms or words that would stem to the same as stemmed synonym.

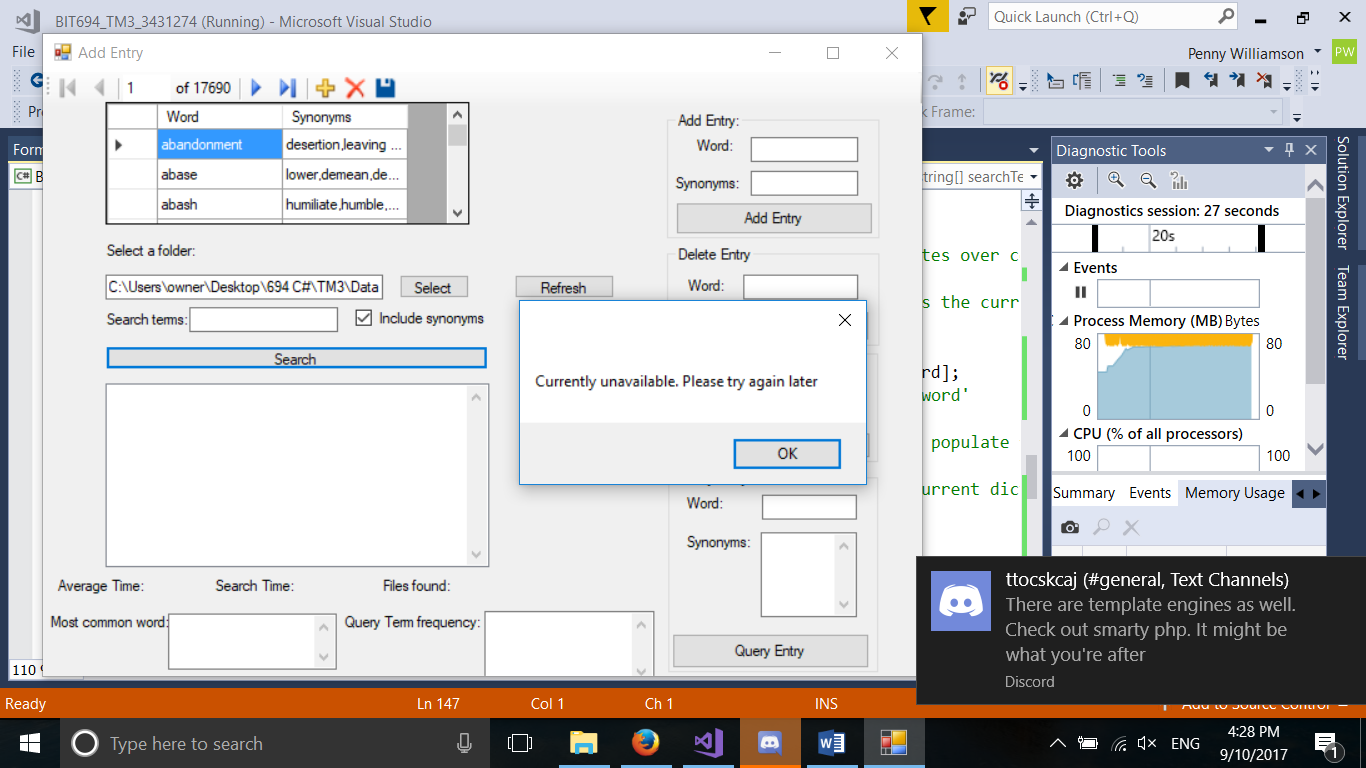
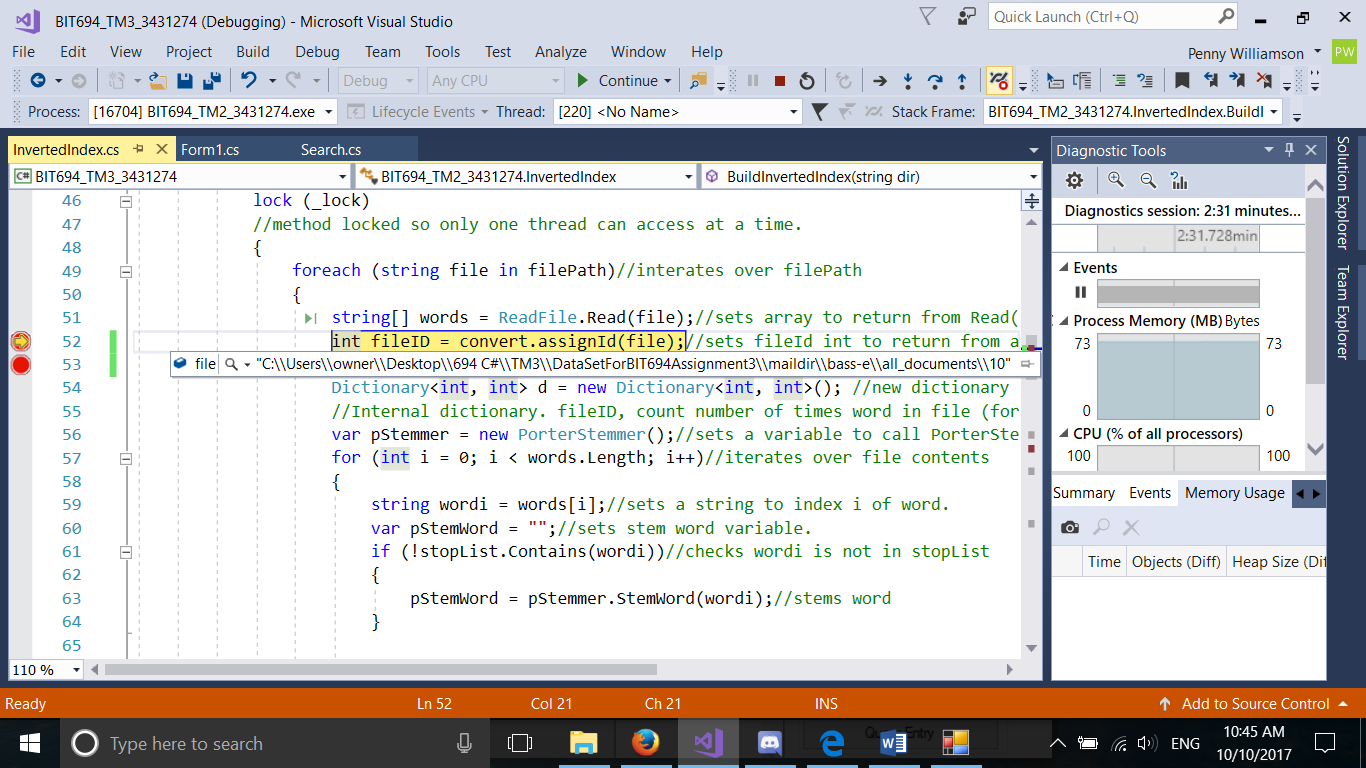


Figure 5. The message while rebuilding index in separate thread.



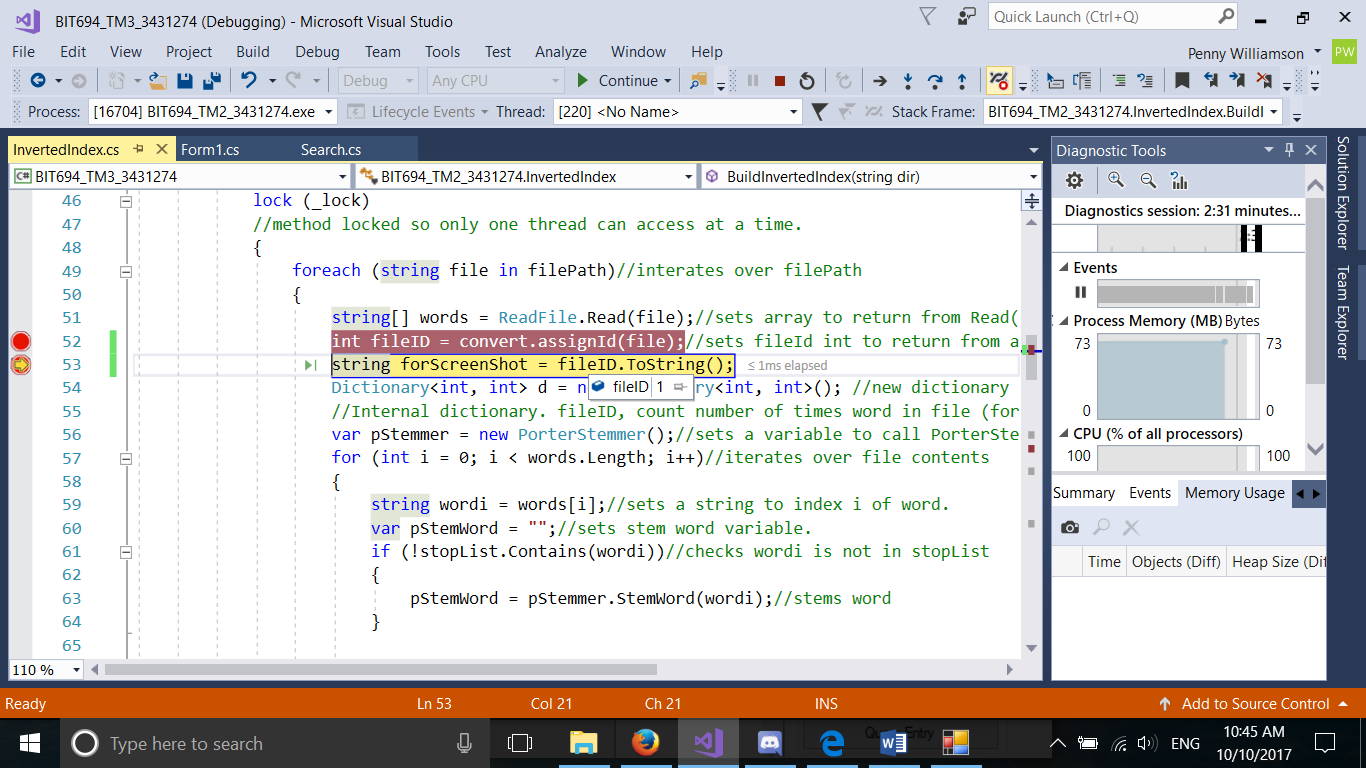


Figure 6. Screen shots showing fileID being assigned.

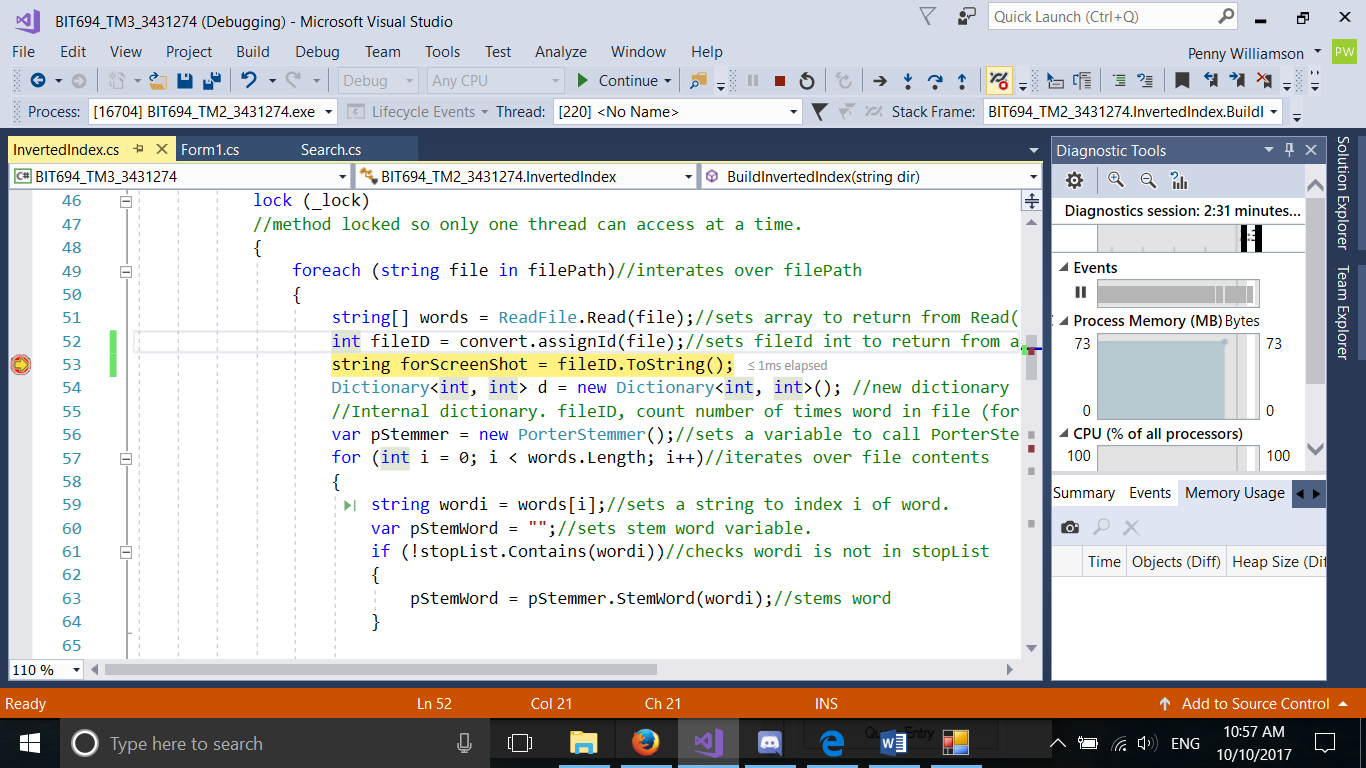


Figure 7. Screen shot of rebuilding index, showing non- main thread being used.

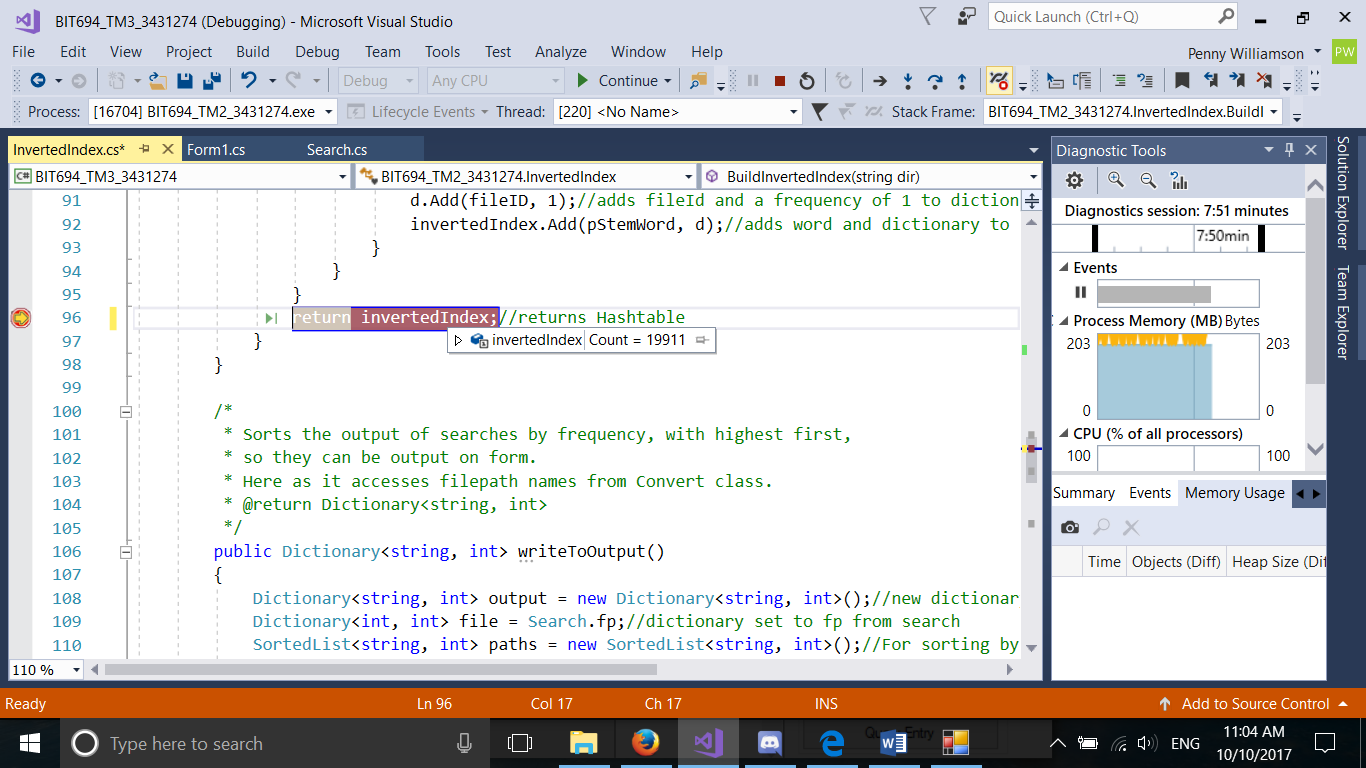


Figure 8. Screen shot showing final size of inverted index.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Maximum mark** | **Self-assessed mark** |
| Handling different file formats | 5% | 5% |
| Adding a stemmer | 10% | 10% |
| Adding the inverted index | 20% | 20% |
| Multi-threading for rebuilding the inverted index | 15% | 15% |
| Optimisation, storing the in the inverted index. The application includes at least 5 classes | 10% | 10% |
| Modular Programming   * Short methods * The use of several classes | 10% | 10% |
| Error handling   * Files do not exist * Files with wrong formats * Other unexpected errors whenever appropriate | 10% | 10% |
| Comments and clarity   * All variables declared as class instance fields need comment statement for their usage; all lines of code need to be indented properly according to their logical scope. * Clear use of private, protected and public properties | 10% | 10% |
| Ordering files by number of query terms | 10% | 10% |
|  | 100% | 100% |

**Form1 Class Code:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

using System.Collections;

using System.Threading;

/\*

\* Class for handling reading from, writing

\* to and opening from form.

\* Also handles the time taken for SearchDir() to run.

\* All methods private for encapsulation of GUI.

\*/

namespace BIT694\_TM2\_3431274

{

public partial class Form1 : Form

{

private string[] searchTerms;//holds the search terms entered by user

private string folderPath;//holds the folder path for the folder selected by user

private List<HashSet<string>> newSearch;//holds the search terms and their synonyms.

private int numSearch; //the number of times a new search has occured.

private long elapsedTime;//holds the time it took SearchDir() to run.

private long average;//holds the average time it takes for SearchDir() to run

private long totalTime;//Holds the total time all SearchDir() runs have taken.

FileUtilities fileUtil = new FileUtilities();//constructor to call methods in FileUtilities class.

private Boolean rebuild;//holds boolean for second thread running

private InvertedIndex index;//Declares new instance for Inverted Index class

public Form1()

{

InitializeComponent();

index = new InvertedIndex();//intializes a new InvertedIndex class object

}

/\*

\* Brings up a folder browser dialog, allowing

\* user to navigate to and select a folder to

\* search.

\* @param object sender, EventArgs e

\*/

private void SelectBut\_Click(object sender, EventArgs e)

{

DialogResult result = folderBrowserDialog1.ShowDialog();

//sets result to button selected on dialog box

//displays folder browser window.

if (result == DialogResult.OK)//fires if a folder is selected and OK pressed.

{

folderPath = folderBrowserDialog1.SelectedPath;//sets folderpath to the selected file path.

SelectFolder.Clear();//sets the textbox to clear

SelectFolder.Text = folderPath;//sets the textbox to folderpath.

index.ResetInvertedIndex();// resets inverted index in memory to a new Hashtable, so it can be built.

index.StopWords();//builds the stopwords ArrayList

}

}

/\*

\* Takes input of file path and search terms.

\* Splits search terms into an array, and if include

\* synonyms is checked, fires method for setting

\* terms and synonyms (newSearch varible).

\* Calls SearchDirectory, runs timer while that is running,

\* and outputs time taken to form.

\* Fires the method to populate form with returned

\* data.

\* @param object sender, EventArgs e

\*/

private void Searchbut\_Click(object sender, EventArgs e)

{

if (rebuild)//fires if inverted index is being refreshed

{

MessageBox.Show("Currently unavailable. Please try again later");

}

else

{

if (SelectFolder.Text == "")//error handling for no file path selected

{

MessageBox.Show("Please select a file and try again");

}

if (SearchTerms.Text == "")//error handling for no search terms entered.

{

MessageBox.Show("Please a enter a search term, and try again");

}

FilePathOutput.Items.Clear();//clears list box.

if (SearchTerms.Text != "" && SelectFolder.Text != "")//fires if there is a filepath and search term entered.

{

numSearch = numSearch + 1;//adds one to variable holding number of searches. Used for average time.

searchTerms = SearchTerms.Text.Split(',', ' ');//splits the search terms at space or comma

for (int i = 0; i < searchTerms.Length; i++)

{

searchTerms[i] = searchTerms[i].ToLower();//sets all entered search terms to lower case, to match the database.

}

if (IncludeSynonyms.Checked)

{

DbUtilities dbUtil = new DbUtilities(newWordsFullDataSet);

//constructor for DbUtilities, takes in dataset as parameter.

//Dataset out of scope in class variables.

newSearch = dbUtil.SearchWithDb(searchTerms);

//sets newSearch variable to return from SearchWithDb(), (terms and their synonyms)

}

var watch = System.Diagnostics.Stopwatch.StartNew();//starts a new instance of stop watch

if (InvertedIndex.invertedIndex.Count == 0)

//Builds inverted index if it yet to be built

{

index.BuildInvertedIndex(folderPath);//builds inverted Index

}

//Searches files and folders, reads contents, then fires the frequency methods.

if (IncludeSynonyms.Checked)

{

Search.IndexedFileMatchTermDb(newSearch);

}

else

{

Search.IndexedFileMatchTerm(searchTerms);

}

watch.Stop();//stops stop watch.

long time = watch.ElapsedMilliseconds;//gets the time in milliseconds

SearchTime.Text = time.ToString() + "ms"; //outputs time to form.

AverageTime.Text = Average(time).ToString() + "ms";//outputs average time to form.

FileToForm();//populates the listbox, number of files found and frequency textboxes

}

}

}

/\*

\* sets time for SearchDir() to elapsed time.

\* Fires Average().

\* @param long time

\* @return long elapsedTime

\*/

public long ElapsedTime(long time)

{

elapsedTime = time;

Average(time);

return elapsedTime;

}

/\*

\* Calculates the average time

\* for a search.

\* @param long aTime

\* @return long average

\*/

public long Average(long aTime)

{

totalTime = totalTime + aTime;

average = totalTime / numSearch;

return average;

}

/\*

\* Populates form with filepaths, most common word,

\* query term frequency and files found.

\* Fires resets for collectionFreq and output,

\* so they are new objects for next search.

\* @param Boolean useDatabase.

\*/

private void FileToForm()

{

Dictionary<string, int> output = index.WriteToOutput();

if (output.Count == 0)//handles no files being found.

{

FilePathOutput.Items.Add("No files found");

}

foreach (string s in output.Keys)//iterates through output, using it's getter.

{

FilePathOutput.Items.Add(s);//adds each string to list box.

}

Found.Text = (output.Count).ToString();//sets Found.Text to length of output, using it's getter.

commonWord.Text = TermFreq.CollectionFreq();//sets commonWord.Text to return variable from CollectionFreq()

QueryTermFreq.Text = TermFreq.QueryTermFreq(searchTerms);//sets QueryTermFreq to variable returned from QueryTermFreq()

ICollection key = InvertedIndex.invertedIndex.Keys;//remove before submission, for debugging.

}

private void wordsBindingNavigatorSaveItem\_Click(object sender, EventArgs e)

{

this.Validate();

this.wordsBindingSource.EndEdit();

this.tableAdapterManager.UpdateAll(this.newWordsFullDataSet);

}

private void Form1\_Load(object sender, EventArgs e)

{

// TODO: This line of code loads data into the 'newWordsDataSet.Words' table. You can move, or remove it, as needed.

this.wordsTableAdapter.Fill(this.newWordsFullDataSet.Words);

}

/\*

\* Takes the input from AddWord and AddSynonyms textboxes

\* and passes them to AddWord in DbUtilities to be added

\* to the database.

\*/

private void AddEntryBut\_Click(object sender, EventArgs e)

{

DbUtilities dbUtil = new DbUtilities(newWordsFullDataSet);

//constructor for DbUtilities, takes in dataset as parameter.

//Dataset out of scope in class variables.

if (AddWord.Text == "")//error checking for blank textbox

{

MessageBox.Show("Please enter a word, and try again");

}

if (AddWord.Text != "")//fires is AddWord textbox is not blank.

{

String word = AddWord.Text;//setting variable to be passed

String syn = AddSynonym.Text;//setting variable to be passed

dbUtil.AddWord(word, syn);//calls method in DbUtilities

//uses local word and syn, and passes the database, so it can be accessed.

}

//resets textboxes to blank.

AddWord.Text = "";

AddSynonym.Text = "";

}

/\*

\* Takes the input from DeleteWord textbox

\* and passes it to the method DeleteWord

\* in DbUtilities to remove it from the

\* database.

\*/

private void DeleteWordBut\_Click(object sender, EventArgs e)

{

DbUtilities dbUtil = new DbUtilities(newWordsFullDataSet);

//constructor for DbUtilities, takes in dataset as parameter.

//Dataset out of scope in class variables.

if (DeleteWord.Text == "")//checks to see if the textbox is blank.

{

MessageBox.Show("Please enter a word and try again");

}

if (DeleteWord.Text != "")//fires if textbox is not blank.

{

String delWord = DeleteWord.Text;//sets variable to be passed.

dbUtil.DeleteWord(delWord);//calls DeleteWord method in DbUtilitites.

//Passes local variable delWord and the database, so it can be accessed.

DeleteWord.Text = "";//sets textbox back to blank.

}

}

/\*

\* Takes the input from UpdateWord and UpdateSynonym textboxes

\* and passes them to the method UpdateWord in DbUtilities

\* class, so that the new synonym can be appended to the end

\* of the synonym list.

\*/

private void UpdateEntryBut\_Click(object sender, EventArgs e)

{

DbUtilities dbUtil = new DbUtilities(newWordsFullDataSet);

//constructor for DbUtilities, takes in dataset as parameter.

//Dataset out of scope in class variables.

if (UpdateWord.Text == "")//checks to see if UpdateWord textbox is blank

{

MessageBox.Show("Please enter a word and try again");

}

if (UpdateWord.Text != "")//fires if textbox is not blank.

{

String word = UpdateWord.Text;//sets local variable to be passed.

String syn = UpdateSynonym.Text;//sets local variable to be passed

dbUtil.UpdateWord(word, syn);//calls UpdateWord method in DbUtilities.

//Passes local variables word and syn as parameters. Also passes the database so it can be accessed.

UpdateSynonym.Text = "";//sets textbox to blank

UpdateWord.Text = "";//sets textbox to blank.

}

}

/\*

\* Takes the input from QueryWord textbox, and passes it to QueryWord

\* in DbUtilities, which queries it, then returns a string of

\* the synonyms, to be displayed in QuerySynonym textbox.

\*/

private void QueryEntryBut\_Click(object sender, EventArgs e)

{

DbUtilities dbUtil = new DbUtilities(newWordsFullDataSet);

//constructor for DbUtilities, takes in dataset as parameter.

//Dataset out of scope in class variables.

if (QueryWord.Text == "")//error handling for no value entered into textbox.

{

MessageBox.Show("Please enter a word and try again");

}

if (QueryWord.Text != "")//fires if textbox is not blank.

{

string word = QueryWord.Text;//sets local variable for passing.

string aStr = dbUtil.Query(word);//Calls Query in DbUtilities.

//Sets local variable aStr to the return str from Query.

QuerySynonym.Text = aStr;//sets the text in the text box to aStr.

}

}

/\*

\* Allows file to be opened by double clicking on them.

\*/

private void FilePathOutput\_MouseDoubleClick(object sender, MouseEventArgs e)

{

try

{

int index = FilePathOutput.IndexFromPoint(e.Location);

//sets the index for folder mouse has double clicked.

if (index != System.Windows.Forms.ListBox.NoMatches)//fires if index is not -1

{

System.Diagnostics.Process.Start(FilePathOutput.SelectedItem.ToString());

//opens the file that was double clicked on.

}

}

catch(Exception err)

{

MessageBox.Show("Error: File cannot be opened." + err);

}

}

/\*

\* Rebuilds inverted index in a seperate thread

\* when refresh button pressed

\*/

private void RefreshIndex\_Click(object sender, EventArgs e)

{

rebuild = true;// triggers message box when search pressed.

if (folderPath == null)//checks a folder path has been selected

{

MessageBox.Show("Please select a folder and try again.");

}

else

{//Declares a new thread method

Thread T1 = new Thread(new ThreadStart(() =>

{

index.ResetInvertedIndex();//clears index

index.BuildInvertedIndex(folderPath);//builds index

rebuild = false;//sets boolean for thread running to false

}

));

T1.Start();//starts thread

}

}

}

}

**InvertedIndex Class code:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace BIT694\_TM2\_3431274

{

class InvertedIndex

{

static internal Hashtable invertedIndex = new Hashtable();//internal so it can be accessed from other classes

//static so they are not garbage collected

static private List<string> stopList;//holds stop list

static readonly object \_lock = new object();//for multi threading

private ArrayList filePath; //arraylist for holding file paths

private Converter convert; //constructor for convert class

public InvertedIndex()

{

convert = new Converter();

//new convert class object.

}

/\*

\* Resets data structures for rebuild

\*/

public void ResetInvertedIndex()

{

invertedIndex.Clear();// clears contents of invertedIndex

TermFreq.ResetFreq();//resets collectionWordsFreq

}

/\*

\* Builds an inverted index, with fileIds from Converter class.

\* @param string dir (starting filepath)

\* @return Hashtable invertedIndex

\*/

public Hashtable BuildInvertedIndex(string dir)

{

FileUtilities fileUtil = new FileUtilities();//constructor so fileUtiltes can be called

filePath = new ArrayList();//new ArrayList

filePath = fileUtil.Search(dir);//sets filePath to return from Search() method

lock (\_lock)

//method locked so only one thread can access at a time.

{

foreach (string file in filePath)//interates over filePath

{

string[] words = ReadFile.Read(file);//sets array to return from Read() method

int fileID = convert.AssignId(file);//sets fileId int to return from assignId()

Dictionary<int, int> d = new Dictionary<int, int>(); //new dictionary

//Internal dictionary. fileID, count number of times word in file (for sort)

var pStemmer = new PorterStemmer();//sets a variable to call PorterStemmer

for (int i = 0; i < words.Length; i++)//iterates over file contents

{

string wordi = words[i];//sets a string to index i of word.

var pStemWord = "";//sets stem word variable.

if (!stopList.Contains(wordi))//checks wordi is not in stopList

{

pStemWord = pStemmer.StemWord(wordi);//stems word

}

if (invertedIndex.ContainsKey(pStemWord))//fires if collection already contains word.

{

try //handles parse errors

{

d = (Dictionary<int, int>)invertedIndex[pStemWord];

//gets dictionary for pStemWord

if (d.Keys.Contains(fileID))//checks to see if fileId is already in dictionary

{

int v = d[fileID] + 1;//sets frequency to frequency plus 1

d[fileID] = v;//changes frequency

}

else

{

d.Add(fileID, 1);

//adds fileID and frequency 1 to dictionary, if fileId not already in dictionary.

}

}

catch (Exception err) //catches parse errors

{

MessageBox.Show("" + err);

}

}

else//fires if word not present in index.

{

d = new Dictionary<int, int>();//sets new dictioanry to add

d.Add(fileID, 1);//adds fileId and a frequency of 1 to dictionary

invertedIndex.Add(pStemWord, d);//adds word and dictionary to Hashtable.

}

}

}

return invertedIndex;//returns Hashtable

}

}

/\*

\* Sorts the output of searches by frequency, with highest first,

\* so they can be output on form.

\* Here as it accesses filepath names from Convert class.

\* @return Dictionary<string, int>

\*/

public Dictionary<string, int> WriteToOutput()

{

Dictionary<string, int> output = new Dictionary<string, int>();//new dictionary

Dictionary<int, int> file = Search.fp;//dictionary set to fp from search

SortedList<string, int> paths = new SortedList<string, int>();//For sorting by value

foreach (int id in file.Keys)//iterates over keys

{

int termCount = file[id]; //gets the frequency count

paths.Add(convert.GetPath(id), termCount);//adds filpath and frequency to sorted list

}

output = paths.OrderByDescending(kvp => kvp.Value).ToDictionary(kvp => kvp.Key, kvp => kvp.Value);

// sets out put to result of sorting list by values, then setting it to a dictionary.

return output;//returns output to form method.

}

/\*

\* A stop list of common words and common words from files.

\* Used to improve inverted index.

\*/

public void StopWords()

{

stopList = new List<string>();//sets new stop list

stopList.Add("the");

stopList.Add("be");

stopList.Add("to");

stopList.Add("of");

stopList.Add("and");

stopList.Add("a");

stopList.Add("in");

stopList.Add("that");

stopList.Add("have");

stopList.Add("I");

stopList.Add("because");

stopList.Add("was");

stopList.Add("is");

stopList.Add("are");

stopList.Add("always");

stopList.Add("85663781075852814876javamailevansthyme 1");

stopList.Add("101014 2");

stopList.Add("brbra 7");

stopList.Add("uhmain 13");

stopList.Add("thackermajestic 55");

stopList.Add("rodgers 98");

stopList.Add("city 5119");

stopList.Add("in 256590");

stopList.Add(" 4452640");

}

}

}

**Converter Class code:** (Open Polytechnic, 2017)

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BIT694\_TM2\_3431274

{

/\*

\*Obtained from:

\*Open Polytechnic. (2017).

\* Performance and optimization.

\* Retrieved September 29, 2017,

\* from BIT694 Developing Applications using c#,

\* Open Polytechnic:

\* https://openpolytechnic.iqualify.com/course/-KkhzLXn7OHayauSqVt6/#/page/p68

\*

\* Modified to make paths and ids static and namespace changed.

\* Comments: Penny Williamson

\*/

class Converter

{

private int counter; //counter to hold current fileId

//Static to avoid garbage collection.

static private Hashtable paths;//

static private ArrayList ids;

/\*

\* Constructor

\*/

public Converter()

{

counter = 0;//sets counter for fileId to 0

paths = new Hashtable();//sets new HashTable

ids = new ArrayList();//sets new ArrayList

}

/\*

\* Returns the fileId for a string path

\* @param String path

\* @return int fileId

\*/

public int GetID(String path)

{

return ((int)paths[path]);

}

/\*

\* Returns the file path associated with fileId

\* @param int id

\* @return String filePath

\*/

public string GetPath(int id)

{

return ((string)ids[id]);

}

/\*

\* Assigns fileId to filePath.

\* @param string path

\* @return int fileId

\*/

public int AssignId(string path)

{

int id = counter;//sets id to current counter value

counter++;//increases counter by 1

paths[path] = id;//associates path with id

ids.Add(path);//associates id with file path

return (id);//returns the fileId

}

}

}

**Search Class code:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace BIT694\_TM2\_3431274

{

static class Search

{

/\*

\* Static as it has no class variables.

\* A static class to handle searching the contents of

\* files and matching them to term or term and synonym.

\*/

static internal Dictionary<int, int> fp;

//Holds fileIds and frequency for output

//internal so InvertedIndex can access it.

/\*

\* Called when Include Synonyms is unchecked.

\* Matches the contents of inverted index to the

\* search terms.

\* @param string[] searchTerms

\*/

static public void IndexedFileMatchTerm(string[] searchTerms)

{

fp = new Dictionary<int, int>(); //Dictionary for holding filepaths in loop.

Dictionary<int, int> d = new Dictionary<int, int>();

//Sets up a collection of the keys of InvertedIndex hashtable for searching.

var pStemmer = new PorterStemmer();//sets a variable to call PorterStemmer

for (int i = 0; i < searchTerms.Length; i++)//iterates over searchTerms

{

var pStemSearchTerm = pStemmer.StemWord(searchTerms[i].ToLower());

//stems the search term.

foreach (string word in InvertedIndex.invertedIndex.Keys)//iterates over collection of keys

{

if (word == pStemSearchTerm)//fires if current key equals search term.

{

d = new Dictionary<int, int>();

d = (Dictionary<int, int>)InvertedIndex.invertedIndex[word];

//sets a dictionary variable for the values of the key 'word'

if (searchTerms[0] == searchTerms[i])//Fires on first run to populate fp.

{

foreach (int key in d.Keys)//iterates over keys of current dictionary

{

int h = d[key];//gets frequency of key for term

fp.Add(key, h);//adds key and frequency to fp

}

}

if (i > 0)

//fires on second and subsequent runs to restrict fp to ids that contain all terms

{

List<int> fpKey = fp.Keys.ToList();

//sets a list to fp keys, as fp is changed in loop.

foreach (int id in fpKey)//iterates over list

{

if (!d.Keys.Contains(id))

//fires if current dictionary key does not match any ids in fpKeys

{

fp.Remove(id);//removes id from fp

}

if (d.Keys.Contains(id))//fires if fpKeys contains current dictionary key.

{

int j = d[id];//gets frequency of key for term

int k = fp[id];//gets frequency of fp id for term

int v = j + k;//adds frequencies together

fp[id] = v;//changes frequency of id to new frequency

}

}

}

}

}

}

}

/\*

\* Called when Include Synonyms is checked.

\* Matches the contents of inverted index to the

\* search terms and synonyms.

\* @param List<HashSet<string>> newSearch

\*/

static public void IndexedFileMatchTermDb(List<HashSet<string>> newSearch)

{

fp = new Dictionary<int, int>(); //Dictionary for holding filepaths in loop.

Dictionary<int, int> d = new Dictionary<int, int>();

Dictionary<int, int> hold;

//Sets up a collection of the keys of InvertedIndex hashtable for searching.

for (int i = 0; i < newSearch.Count; i++)//iterates over newSearch

{

hold = new Dictionary<int, int>();

//newSearch stemmed in DBUtilities SearchWithDB()

foreach (string word in InvertedIndex.invertedIndex.Keys)//iterates over collection of keys

{

if (newSearch[i].Contains(word))//fires if newSearch contains the current key.

{

d = new Dictionary<int, int>();

d = (Dictionary<int, int>)InvertedIndex.invertedIndex[word];

//sets a dictionary variable for the values of the key 'word'

if (newSearch[0] == newSearch[i])//Fires on first run to populate fp.

{

foreach (int key in d.Keys)//iterates over keys of current dictionary

{

if (!fp.Keys.Contains(key))

{

int h = d[key];//gets frequency of key for term

fp.Add(key, h);//adds key and frequency to fp

}

else

{

int j = d[key];//gets frequency of dictionary key for term

int k = fp[key];//gets frequency of fp id for term

int v = j + k;//Adds frequenies together.

hold[key] = v;//changes frequency of id to new frequency

}

}

}

}

if (i > 0)

//fires on second and subsequent runs to set dictionary of all file ids and

//frequencies of a term and its synonyms

{

List<int> dKey = d.Keys.ToList();

//sets a list to fp keys, as fp is changed in loop.

foreach (int key in d.Keys)

{

if (!hold.Keys.Contains(key))

//fires if current dictionary key does not match any ids in fpKeys

{

int h = d[key];

hold.Add(key, h);

}

if (hold.Keys.Contains(key))//fires if fpKeys contains current dictionary key.

{

int j = d[key];//gets frequency of dictionary key for term

int k = hold[key];//gets frequency of fp id for term

int v = j + k;//Adds frequenies together.

hold[key] = v;//changes frequency of id to new frequency

}

}

}

}

if (i > 0)//fires only after hold has been populated, on second and subsequent runs

{

List<int> fpKey = fp.Keys.ToList();

List<int> holdKey = hold.Keys.ToList();

//sets a list to fp keys, as fp is changed in loop.

foreach (int id in fpKey)

{

if (!holdKey.Contains(id))

//fires if current dictionary key does not match any ids in fpKeys

{

fp.Remove(id);

}

if (holdKey.Contains(id))//fires if fpKeys contains current dictionary key.

{

int j = hold[id];//gets frequency of dictionary key for term

int k = fp[id];//gets frequency of fp id for term

int v = j + k;//Adds frequenies together.

fp[id] = v;//changes frequency of id to new frequency

}

}

}

}

}

**ReadFile Class Code:** (asubach, 2017)**,** (code7248, 2012)**,** (grimholtz, 2012)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

using Code7248.word\_reader;

using iTextSharp.text;

using iTextSharp.text.pdf;

using iTextSharp.text.pdf.parser;

using CSharpJExcel;

using CSharpJExcel.Jxl;

using System.Collections;

namespace BIT694\_TM2\_3431274

{

/\*

\* Class for reading file formats

\*

\* Third party libraries:

\*

\* asubach, a. ,.-a. (2017, September 13).

\* iTextShap a .Net pdf library.

\* Retrieved September 20, 2017,

\* from SourceForge.net: http://sourceforge.net/projects/itextsharp/ ,

\*

\* code7248, t. (2012, September 26).

\* Code7248.word\_reader.

\* Retrieved September 20, 2017,

\* from SourceForge.net: https://sourceforge.net/projects/word-reader/ ,

\*

\* grimholtz, l. (2012, October 08).

\* A Java library for reading/writing Excel.

\* Retrieved September 20, 2017,

\* from SourceForge.net: https://sourceforge.net/projects/jexcelapi/

\*/

static class ReadFile

{

/\*

\* Determines file extension of file, and sends it to

\* appropiate method. And returns output to invertedIndex()

\*

\* @param string file (filepath)

\* @return string[]

\*/

static public string[] Read(string file)

{

string[] words = null;

try//catches any files that cannot be opened.

{

string ext = System.IO.Path.GetExtension(file);

//gets file extension

if (ext == ".txt" || ext == "")

{

words = ReadTxt(file);

}

if (ext == ".doc")

{

words = ReadDoc(file);

}

if (ext == ".pdf")

{

words = ReadPdf(file);

}

if (ext == ".xls")

{

words = ReadXls(file);

}

}

catch (Exception err)

{

MessageBox.Show("Error: " + err);

}

return words;

}

/\*

\* Reads text files using StreamReader

\* @param string file (filepath)

\* @return string[]

\*/

static private string[] ReadTxt(string file)

{

string[] words = null;//sets a new array

try //caches StreamReader errors

{

String line;//variable to hold output from StreamReader

TextReader tr = new StreamReader(file);//reads the file

while ((line = tr.ReadLine()) != null)//fires if line is not null

{

if (!line.Contains("X-FileName") || !line.Contains("Message-ID:") || !line.Contains("To:"))

//removes some headers

{

words = SetArray(line, file);//sets words to return from method

}

}

tr.Close();//closes the stream reader

}

catch (Exception err)

{

MessageBox.Show("Error: " + err);

}

return words;//returns array to read() method

}

/\*

\* Reads .doc files

\* Useage patterns from (code7248, 2012)

\* @param string file (filepath)

\* @return string[]

\*/

static private string[] ReadDoc(string file)

{

string[] words = null;//sets new array

try

{

TextExtractor te = new TextExtractor(file);//readers text

string line = te.ExtractText();//sets string to read line

words = SetArray(line, file);//sets array to line read

}

catch (Exception err)

{

MessageBox.Show("Error: " + err);

}

return words;//returns array to read() method

}

/\*

\* reads .pdf files.

\* Useage patterns from (asubach, 2017)

\* @param string file (filepath)

\* @return string[]

\*/

static private string[] ReadPdf(string file)

{

string[] words = null;//sets new array

try

{

StringBuilder line = new StringBuilder();//stringbuilder object to hold text

using (PdfReader read = new PdfReader(file))//pdf reader object

{

for (int j = 1; j <= read.NumberOfPages; j++)//iterates over pages

{

line.Append(PdfTextExtractor.GetTextFromPage(read, j));

}//adds page to line

read.Close();//clese pdf reader

}

words = SetArray(line.ToString(), file);//sets array to return from method

}

catch (Exception err)

{

MessageBox.Show("Error: " + err);

}

return words;//returns array to read() method

}

/\*

\* Reads .xls files

\* Useage patterns from (grimholtz, 2012)

\* @param string file (filepath)

\* @return string[]

\*/

static private string[] ReadXls(string file)

{

string[] words = null;//sets new Array

string line = "";//sets line to hold contents

try

{

Workbook wrkbk = Workbook.getWorkbook(new System.IO.FileInfo(file));//gets workbook

for (int i = 0; i < wrkbk.getNumberOfSheets(); i++)//iterate sheets

{

var sheet = wrkbk.getSheet(i); //sets a variable holding current sheet

for (int j = 0; j < sheet.getColumns(); j++)//iterate columns

{

for (int k = 0; k < sheet.getRows(); k++)//iterate rows

{

var con = sheet.getCell(j, k).getContents();//sets a variable holding a current cell

line += con;//adds content of cell onto the string line.

}

}

}

words = SetArray(line, file);//sets array

wrkbk.close();//closes the workbook

}

catch (Exception err)

{

MessageBox.Show("Error: " + err);

}

return words;//returns word to read() method

}

/\*

\* Sets the arrays for methods.

\* @param string line (the contents of file)

\* @param string file (filepath)

\* @return string[]

\*/

static private string[] SetArray(string line, string file)

{

string[] words;//sets new array

String trimLine = line.Trim();//trims whitespace, /r and /n from start and end of line.

String aLine = Regex.Replace(trimLine, @"[^\w\s \_]", "");//removes punctuation, and replaces it with "".

words = aLine.Split(' ');//sets an array to the line split at spaces.

for (int i = 0; i < words.Length; i++)

{

if (words[i] != "" || words[i].Length < 50)//fires if word[i] is not "".

{

words[i] = words[i].ToLower();//sets the words to lowercase

String wordi = words[i];//sets the String variable wordi to words[i]

TermFreq.AddToCollectionFreq(wordi);//adds the word to collectionWordFreq Hashtable.

}

}

return words;//returns words array to method that called SetArray()

}

}

}

**FileUtilities Class Code:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace BIT694\_TM2\_3431274

{

class FileUtilities

{

static readonly object \_lock = new object();//used for multi threading

private ArrayList filePath = new ArrayList();

//ArrayList to hold file paths to be read. Set here, as method iterates

/\*

\* Constructor

\*/

public FileUtilities() { }

/\*

\* Recursive method that searches over directory structure

\* to find the files. Then reads the contents of each file, and

\* calls addToCollectionFreq() for each word.

\* Calls appropriate FileMatch method for whether

\* the search is being done with database or without.

\* if true is returned, writeToOutput() is called.

\* @params string dir, Boolean useDataBase, string[] searchTerms,

\* @params List<HashSet<string>> newSearch

\*/

public void SearchDir(string dir)

{

try//handles TextReader and StreamReader errors and non existing-file paths.

{

foreach (string file in Directory.GetFiles(dir))//iterates over files in the current directory

{

lock (\_lock)

//method locked so only one thread can access at a time.

//Locked here as to avoid problems with same file being accessed.

{

filePath.Add(file);//adds file path to ArrayList

}

}

foreach (string dirs in Directory.GetDirectories(dir))//iterates through dirctories in directory structure.

{

SearchDir(dirs);//calls its self, so files in next directory can be searched.

}

}

catch (Exception err)//catches TextReader and StreamReader errors

{

MessageBox.Show("" + err);

}

}

/\*

\* Calls searchDir for InvertedIndex.

\*

\* @param string dir (starting file path)

\* @return ArrayList filepath

\*/

public ArrayList Search(string dir)

{

SearchDir(dir);

return filePath;

}

}

}

**PorterStemmer Class Code:** (Patton, 2015)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BIT694\_TM2\_3431274

{ ///Obtained from:

///Patton, B. (2015).

///csharp3.

///Retrieved September 21, 2017,

///from tartarus.org: https://tartarus.org/martin/PorterStemmer/csharp3.txt

///

/// Modified by namespace being changed from Poseidon.Analysis.

///

/// <summary>

/// The Stemmer class transforms a word into its root form.

/// Implementing the Porter Stemming Algorithm

/// </summary>

/// <remarks>

/// Modified from: http://tartarus.org/martin/PorterStemmer/csharp2.txt

/// </remarks>

/// <example>

/// var stemmer = new PorterStemmer();

/// var stem = stemmer.StemWord(word);

/// </example>

public class PorterStemmer

{

// The passed in word turned into a char array.

// Quicker to use to rebuilding strings each time a change is made.

private char[] wordArray;

// Current index to the end of the word in the character array. This will

// change as the end of the string gets modified.

private int endIndex;

// Index of the (potential) end of the stem word in the char array.

private int stemIndex;

/// <summary>

/// Stem the passed in word.

/// </summary>

/// <param name="word">Word to evaluate</param>

/// <returns></returns>

public string StemWord(string word)

{

// Do nothing for empty strings or short words.

if (string.IsNullOrWhiteSpace(word) || word.Length <= 2) return word;

wordArray = word.ToCharArray();

stemIndex = 0;

endIndex = word.Length - 1;

Step1();

Step2();

Step3();

Step4();

Step5();

Step6();

var length = endIndex + 1;

return new String(wordArray, 0, length);

}

// Step1() gets rid of plurals and -ed or -ing.

/\* Examples:

caresses -> caress

ponies -> poni

ties -> ti

caress -> caress

cats -> cat

feed -> feed

agreed -> agree

disabled -> disable

matting -> mat

mating -> mate

meeting -> meet

milling -> mill

messing -> mess

meetings -> meet \*/

private void Step1()

{

// If the word ends with s take that off

if (wordArray[endIndex] == 's')

{

if (EndsWith("sses"))

{

endIndex -= 2;

}

else if (EndsWith("ies"))

{

SetEnd("i");

}

else if (wordArray[endIndex - 1] != 's')

{

endIndex--;

}

}

if (EndsWith("eed"))

{

if (MeasureConsontantSequence() > 0)

endIndex--;

}

else if ((EndsWith("ed") || EndsWith("ing")) && VowelInStem())

{

endIndex = stemIndex;

if (EndsWith("at"))

SetEnd("ate");

else if (EndsWith("bl"))

SetEnd("ble");

else if (EndsWith("iz"))

SetEnd("ize");

else if (IsDoubleConsontant(endIndex))

{

endIndex--;

int ch = wordArray[endIndex];

if (ch == 'l' || ch == 's' || ch == 'z')

endIndex++;

}

else if (MeasureConsontantSequence() == 1 && IsCVC(endIndex)) SetEnd("e");

}

}

// Step2() turns terminal y to i when there is another vowel in the stem.

private void Step2()

{

if (EndsWith("y") && VowelInStem())

wordArray[endIndex] = 'i';

}

// Step3() maps double suffices to single ones. so -ization ( = -ize plus

// -ation) maps to -ize etc. note that the string before the suffix must give m() > 0.

private void Step3()

{

if (endIndex == 0) return;

/\* For Bug 1 \*/

switch (wordArray[endIndex - 1])

{

case 'a':

if (EndsWith("ational")) { ReplaceEnd("ate"); break; }

if (EndsWith("tional")) { ReplaceEnd("tion"); }

break;

case 'c':

if (EndsWith("enci")) { ReplaceEnd("ence"); break; }

if (EndsWith("anci")) { ReplaceEnd("ance"); }

break;

case 'e':

if (EndsWith("izer")) { ReplaceEnd("ize"); }

break;

case 'l':

if (EndsWith("bli")) { ReplaceEnd("ble"); break; }

if (EndsWith("alli")) { ReplaceEnd("al"); break; }

if (EndsWith("entli")) { ReplaceEnd("ent"); break; }

if (EndsWith("eli")) { ReplaceEnd("e"); break; }

if (EndsWith("ousli")) { ReplaceEnd("ous"); }

break;

case 'o':

if (EndsWith("ization")) { ReplaceEnd("ize"); break; }

if (EndsWith("ation")) { ReplaceEnd("ate"); break; }

if (EndsWith("ator")) { ReplaceEnd("ate"); }

break;

case 's':

if (EndsWith("alism")) { ReplaceEnd("al"); break; }

if (EndsWith("iveness")) { ReplaceEnd("ive"); break; }

if (EndsWith("fulness")) { ReplaceEnd("ful"); break; }

if (EndsWith("ousness")) { ReplaceEnd("ous"); }

break;

case 't':

if (EndsWith("aliti")) { ReplaceEnd("al"); break; }

if (EndsWith("iviti")) { ReplaceEnd("ive"); break; }

if (EndsWith("biliti")) { ReplaceEnd("ble"); }

break;

case 'g':

if (EndsWith("logi"))

{

ReplaceEnd("log");

}

break;

}

}

/\* step4() deals with -ic-, -full, -ness etc. similar strategy to step3. \*/

private void Step4()

{

switch (wordArray[endIndex])

{

case 'e':

if (EndsWith("icate")) { ReplaceEnd("ic"); break; }

if (EndsWith("ative")) { ReplaceEnd(""); break; }

if (EndsWith("alize")) { ReplaceEnd("al"); }

break;

case 'i':

if (EndsWith("iciti")) { ReplaceEnd("ic"); }

break;

case 'l':

if (EndsWith("ical")) { ReplaceEnd("ic"); break; }

if (EndsWith("ful")) { ReplaceEnd(""); }

break;

case 's':

if (EndsWith("ness")) { ReplaceEnd(""); }

break;

}

}

/\* step5() takes off -ant, -ence etc., in context <c>vcvc<v>. \*/

private void Step5()

{

if (endIndex == 0) return;

switch (wordArray[endIndex - 1])

{

case 'a':

if (EndsWith("al")) break; return;

case 'c':

if (EndsWith("ance")) break;

if (EndsWith("ence")) break; return;

case 'e':

if (EndsWith("er")) break; return;

case 'i':

if (EndsWith("ic")) break; return;

case 'l':

if (EndsWith("able")) break;

if (EndsWith("ible")) break; return;

case 'n':

if (EndsWith("ant")) break;

if (EndsWith("ement")) break;

if (EndsWith("ment")) break;

/\* element etc. not stripped before the m \*/

if (EndsWith("ent")) break; return;

case 'o':

if (EndsWith("ion") && stemIndex >= 0 && (wordArray[stemIndex] == 's' || wordArray[stemIndex] == 't')) break;

/\* j >= 0 fixes Bug 2 \*/

if (EndsWith("ou")) break; return;

/\* takes care of -ous \*/

case 's':

if (EndsWith("ism")) break; return;

case 't':

if (EndsWith("ate")) break;

if (EndsWith("iti")) break; return;

case 'u':

if (EndsWith("ous")) break; return;

case 'v':

if (EndsWith("ive")) break; return;

case 'z':

if (EndsWith("ize")) break; return;

default:

return;

}

if (MeasureConsontantSequence() > 1)

endIndex = stemIndex;

}

/\* step6() removes a final -e if m() > 1. \*/

private void Step6()

{

stemIndex = endIndex;

if (wordArray[endIndex] == 'e')

{

var a = MeasureConsontantSequence();

if (a > 1 || a == 1 && !IsCVC(endIndex - 1))

endIndex--;

}

if (wordArray[endIndex] == 'l' && IsDoubleConsontant(endIndex) && MeasureConsontantSequence() > 1)

endIndex--;

}

// Returns true if the character at the specified index is a consonant.

// With special handling for 'y'.

private bool IsConsonant(int index)

{

var c = wordArray[index];

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') return false;

return c != 'y' || (index == 0 || !IsConsonant(index - 1));

}

/\* m() measures the number of consonant sequences between 0 and j. if c is

a consonant sequence and v a vowel sequence, and <..> indicates arbitrary

presence,

<c><v> gives 0

<c>vc<v> gives 1

<c>vcvc<v> gives 2

<c>vcvcvc<v> gives 3

.... \*/

private int MeasureConsontantSequence()

{

var n = 0;

var index = 0;

while (true)

{

if (index > stemIndex) return n;

if (!IsConsonant(index)) break; index++;

}

index++;

while (true)

{

while (true)

{

if (index > stemIndex) return n;

if (IsConsonant(index)) break;

index++;

}

index++;

n++;

while (true)

{

if (index > stemIndex) return n;

if (!IsConsonant(index)) break;

index++;

}

index++;

}

}

// Return true if there is a vowel in the current stem (0 ... stemIndex)

private bool VowelInStem()

{

int i;

for (i = 0; i <= stemIndex; i++)

{

if (!IsConsonant(i)) return true;

}

return false;

}

// Returns true if the char at the specified index and the one preceeding it are the same consonants.

private bool IsDoubleConsontant(int index)

{

if (index < 1) return false;

return wordArray[index] == wordArray[index - 1] && IsConsonant(index);

}

/\* cvc(i) is true <=> i-2,i-1,i has the form consonant - vowel - consonant

and also if the second c is not w,x or y. this is used when trying to

restore an e at the end of a short word. e.g.

cav(e), lov(e), hop(e), crim(e), but

snow, box, tray. \*/

private bool IsCVC(int index)

{

if (index < 2 || !IsConsonant(index) || IsConsonant(index - 1) || !IsConsonant(index - 2)) return false;

var c = wordArray[index];

return c != 'w' && c != 'x' && c != 'y';

}

// Does the current word array end with the specified string.

private bool EndsWith(string s)

{

var length = s.Length;

var index = endIndex - length + 1;

if (index < 0) return false;

for (var i = 0; i < length; i++)

{

if (wordArray[index + i] != s[i]) return false;

}

stemIndex = endIndex - length;

return true;

}

// Set the end of the word to s.

// Starting at the current stem pointer and readjusting the end pointer.

private void SetEnd(string s)

{

var length = s.Length;

var index = stemIndex + 1;

for (var i = 0; i < length; i++)

{

wordArray[index + i] = s[i];

}

// Set the end pointer to the new end of the word.

endIndex = stemIndex + length;

}

// Conditionally replace the end of the word

private void ReplaceEnd(string s)

{

if (MeasureConsontantSequence() > 0) SetEnd(s);

}

}

}

**TermFreq Class Code:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace BIT694\_TM2\_3431274

{

static class TermFreq

{

/\*

\* Static as the ony class variable is static.

\* Static class for adding words from files to

\* Hashtable collectionWordsFreq.

\* And iterating over collectionWordsFreq

\* for query term frequency and most common word.

\*/

static private Hashtable collectionWordsFreq = new Hashtable();

//static varible to hold collectionTermsFreq in memory.

//Initilized here as AddToCollectionFreq is called for every word in a file.

/\*

\* Sets collectionWordsFreq to a new

\* instance of Hashtable. Allows a new

\* search to have an empty dataset.

\*/

static public void ResetFreq()

{

collectionWordsFreq.Clear();

}

/\*

\* Adds a word to collectionWordsFeq if it is not

\* already present, and sets frequency to one.

\* Otherwise increases frequency of word by one.

\* @param String wordi

\* @return Hashtable collectionWordsFreq

\*/

static public Hashtable AddToCollectionFreq(String wordi)

{

if (collectionWordsFreq.ContainsKey(wordi))//fires if collection already contains word.

{

try //handles parse errors

{

collectionWordsFreq[wordi] = double.Parse(collectionWordsFreq[wordi].ToString()) + 1;

// increases frequency by one.

}

catch (Exception err) //catches parse errors

{

MessageBox.Show("" + err);

}

}

else//fires if word not present.

{

collectionWordsFreq.Add(wordi, 1);//adds word and frequency of 1 to Hashtable.

}

return collectionWordsFreq;//returns Hashtable

}

/\*

\* Works out the most common word found in files.

\* @return string mostCommon

\*/

static public string CollectionFreq()

{

string[] common = new string[1];//holds most common term for collection frequency

int freq = 0;//holds current highest frequency value

int i = 0;//holds value for array position

foreach (String p in collectionWordsFreq.Keys)//iterates over collectionWordsFreq uing the keys (words)

{

try //handles parse errors

{

int val = int.Parse(collectionWordsFreq[p].ToString());//sets value to the value of key frequency

if (val > freq)//fires if frequency of word is higher than current highest frequency.

{

if (p != "")//fires only if word is not a line return or other null value.

{

common[i] = p + " " + collectionWordsFreq[p];//adds word and frequency

i = i + 1;//increase i by 1.

freq = val;//sets freq to new highest val.

}

if (i == 1)//fires if i = 1

{

i = 0;//sets i back to 0.

}

}

}

catch (Exception err)//catches parse errors

{

MessageBox.Show("" + err);

}

}

string mostCommon = common[0];//sets string mostCommon to value of common[0]

return mostCommon;//returns string.

}

/\*

\* Uses collectionWordsFreq to work out the frequency

\* of query terms in the files.

\* @param string[] searchTerms

\* @return string terms.

\*/

static public string QueryTermFreq(string[] searchTerms)

{

string terms = "";//sets string for return.

foreach (String s in searchTerms)//iterates over searchTerms

{

foreach (String t in collectionWordsFreq.Keys)//iterates over collection freq

{

if (s == t)//fires if key in collectionWordsFreq equals current searchTerm(s)

{

terms += t + " " + collectionWordsFreq[t].ToString() + "\r\n";

//Adds word and frequency, followed by a new line to string terms

}

}

}

return terms;//returns string.

}

}

}

**DBUtilities Class Code:**

using BIT694\_TM2\_3431274.NewWordsFullDataSetTableAdapters;

//Allows me to use wordTableAdapters to update the table.

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using System.Windows.Forms;

/\*

\* The class for methods that call the database,

\* or use the database.

\*/

namespace BIT694\_TM2\_3431274

{

class DbUtilities

{

private NewWordsFullDataSet newWordsFullDataSet; //dataset variable for class use.

/\*

\* Construtor. Takes in the dataset

\* newWorldsFullDataSet. Allows it to be

\* used by methods, without having to pass

\* it in the method signature.

\* @param NewWordsFullDataSet ds

\*/

public DbUtilities(NewWordsFullDataSet ds) { this.newWordsFullDataSet = ds; }

/\*

\* Public, so it can be called from Form1.cs.

\* Searches the database for the synonyms of

\* the search terms. Creates an List of HashSet's of the search

\* terms and their synonyms.

\* @params searchTerms array and newWordsFullDataSet

\* @returns List<HashSet<string>> newSearch.

\*/

public List<HashSet<string>> SearchWithDb(string[] searchTerms)

{

var pStemmer = new PorterStemmer();//sets a variable to call PorterStemmer.

//Stemmed here as it is easier than stemming the HashSet<string> in Search class.

List<HashSet<string>> newSearch = new List<HashSet<string>>();

//initialises and declares the List of HashSets to hold searchTerms and their synonyms.

//Returned at end of method.

try//handles database call errors.

{

foreach (string s in searchTerms) //iterates over passed in searchTerm array

{

HashSet<string> term = new HashSet<string>();//HashSet to add to List

var pStemS = pStemmer.StemWord(s); //stems the search term.

term.Add(pStemS);//Adds the stemmed searchTerm pStemS to HashSet.

NewWordsFullDataSet.WordsRow wordsRow = newWordsFullDataSet.Words.FindByWord(s);

//gets the row with s as index

if (wordsRow != null)//checks that a row is retrieved.

{

string[] synList = wordsRow.Synonyms.ToString().Split(',');//splits the word row at , and stores them in an array.

for (int j = 0; j < synList.Length; j++)//iterates over the synonyms array, length - 1 times.

{

var pStemSyn = pStemmer.StemWord(synList[j]); //stems the synonym.

term.Add(pStemSyn);//Adds the stemmed synonym to the term HashSet.

}

}

newSearch.Add(term);//Adds the HashSet to the List

}

}

catch (Exception err)

{

MessageBox.Show("" + err);

}

return newSearch; //returns the List<HashSet<string>> containing searchTerms and their synonyms.

}

/\*

\* Public so it can be called from Form1.cs.

\* Adds the passed in word, and its passed in

\* synonyms to the database.

\* @params word, syn and newWordsFullDataSet.

\*/

public void AddWord(String word, String syn)

{

try//handles database call errors

{

DataRow newRow = newWordsFullDataSet.Tables["Words"].NewRow();//local variable containing a blank row for words table.

newRow["Word"] = word;//sets the Word column in newRow to passed in word variable.

newRow["Synonyms"] = syn;//sets the Synonyms column in newRow to passed in syn variable.

newWordsFullDataSet.Tables["Words"].Rows.Add(newRow);//adds newRow to the dataSet in memory.

WordsTableAdapter wordsTableAdapter = new WordsTableAdapter();//creates an instance of TableAdapter

wordsTableAdapter.Update(newWordsFullDataSet);//Adds the new row to the database table Words.

MessageBox.Show(word + " and synonyms added to database");

}

catch (Exception err)

{

MessageBox.Show("" + err);

}

}

/\*

\* Public so it can be called from Form1.cs

\* Deletes the row the passed in varaible

\* delWord indexes, from the database.

\* @params delWord, newWordsFullDataSet.

\*/

public void DeleteWord(String delWord)

{

try//handles database errors

{

NewWordsFullDataSet.WordsRow wordsRow = newWordsFullDataSet.Words.FindByWord(delWord);

if (wordsRow == null)

{

MessageBox.Show(delWord + " is not in database");

}

if (wordsRow != null)

{

//sets dataSet newRow to the row in the dataSet indexed by delWord

wordsRow.Delete();//deletes from the dataSet.

MessageBox.Show(delWord + " and synonyms deleted");

}

}

catch (Exception err)

{

MessageBox.Show("Error occured " + err);

}

}

/\*

\* Public so it can be called from Form.cs

\* Updates the synonym list in the DataSet,

\* for the word indexed by passed in variable word,

\* with the passed in variable syn.

\* @params word, syn and newWordsFullDataSet.

\*/

public void UpdateWord(String word, String syn)

{

try//handles database errors

{

NewWordsFullDataSet.WordsRow wordsRow = newWordsFullDataSet.Words.FindByWord(word);

if (wordsRow == null)

{

MessageBox.Show(word + " is not in database");

}

if (wordsRow != null)

{

//sets wordsRow to the DataSet row indexed by word.

wordsRow.Synonyms += "," + syn; //adds to the list of existing synonyms in wordsRow.

MessageBox.Show("Synonym added to " + word + " synonym list");

}

}

catch (Exception err)

{

MessageBox.Show("" + err);

}

}

/\*

\* Public so it can be called from Form1.cs.

\* Queries the DataSet for the synonyms indexed by the passed in variable

\* word. Splits the synonym csv and reads it into string array.

\* @params word, newWordsFullDataSet

\* @return string str

\*/

public string Query(string word)

{

string str = "";//the string variable that is returned.

try//handles database errors.

{

NewWordsFullDataSet.WordsRow wordsRow = newWordsFullDataSet.Words.FindByWord(word);

//sets wordsRow to the dataSet row indexed by passed in variable word.

if (wordsRow == null)

{

MessageBox.Show(word + " is not in database");

}

if (wordsRow != null)//fires if word is an index in DataSet.

{

string[] synList = wordsRow.Synonyms.ToString().Split(',');

//sets an Array to the split result for the synonym csv.

foreach (string s in synList)//iterates over synList array

{

str += s + "\r\n";

//sets str to a series of string varaibles, that will display on new lines.

}

}

}

catch (Exception err)

{

MessageBox.Show("" + err);

}

return str;

//returns the string of synonyms to be displayed in QuerySynonym textbox.

}

}

}

# **References**

asubach, a. ,.-a. (2017, September 13). *iTextShap a .Net pdf library*. Retrieved September 20, 2017, from SourceForge.net: http://sourceforge.net/projects/itextsharp/

code7248, t. (2012, September 26). *Code7248.word\_reader*. Retrieved September 20, 2017, from SourceForge.net: https://sourceforge.net/projects/word-reader/

grimholtz, l. (2012, October 08). *A Java library for reading/writing Excel*. Retrieved September 20, 2017, from SourceForge.net: https://sourceforge.net/projects/jexcelapi/

Open Polytechnic. (2017). *Performance and optimization*. Retrieved September 29, 2017, from BIT694 Developing Applications using c#, Open Polytechnic: https://openpolytechnic.iqualify.com/course/-KkhzLXn7OHayauSqVt6/#/page/p68

Patton, B. (2015). *csharp3*. Retrieved September 21, 2017, from tartarus.org: https://tartarus.org/martin/PorterStemmer/csharp3.txt