My implementation is done with using C# language and Visual Studio editor. It is a console application so anyone can run the code in terminal by doing these steps if necessary:

For Windows:

1- Create a user variable in environment variables, namely "PATH".
2- Copy the following path in this variable: "C:\Windows\Microsoft.NET\Framework\v4.0.30319" or depending upon which .NET your PC have.
3- Simply use "> csc Program.cs"
4- Then run it with a path argument like (path of the folder that includes .txt files): “> Program D:\Github\CSE4094_Project1\Project\Resources”

I implemented a Trie data structure to work on. The reason I chose this data structure is because it is used in some important features like auto complete. Besides other tree search algorithms, Trie can insert, search, and delete in O(N) time where N is the letter size that user defines. The trade-off here is the memory, but in today’s standards we can trade the memory for speed without a doubt.

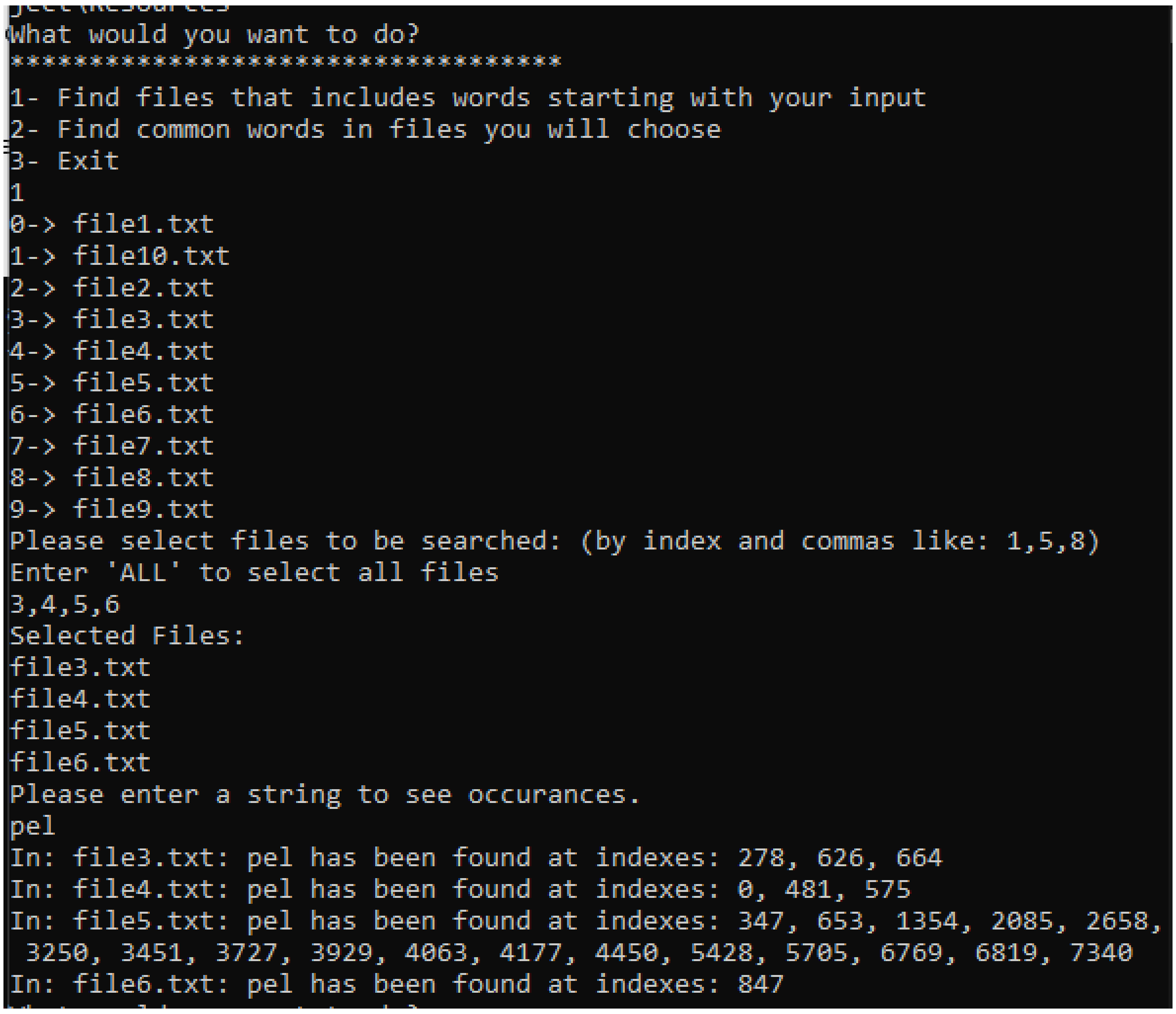
Since the project requires us to find words that starts with the input, I simply decided to use Trie. The main structure is Trie, but I have added other features to the nodes to achieve better results and more information.

Each node has:

- A list of possible child letter nodes.
- A list that holds which files this node is the end of the word.
- A Dictionary I have implemented that can hold the file as a key and a word with an index as the value. Since you cannot have multiple instances of a key in a dictionary, I had to implement my own dictionary so now multiple indexes of a word in an individual file can be stored. The new dictionary class has its own Add method: If there is no instance of an index for a file, it simply creates an index list then inserts it to Dictionary. KeyandValue is of this type: <filename, List<int>>. If another index must be added to the same file in the Dictionary, it inserts the new index value to the index list directly.
- In the preprocessing stage, all the selected files are read, and the words are extracted clearly. Index is stored while passing the words to insertion method, so it could be reached after a search.
- There are 2 different search methods, one finds if a desired word in desired files exist in the Trie, and the other traverses all the tree recursively and finds all the occurrences of common words for the desired files.

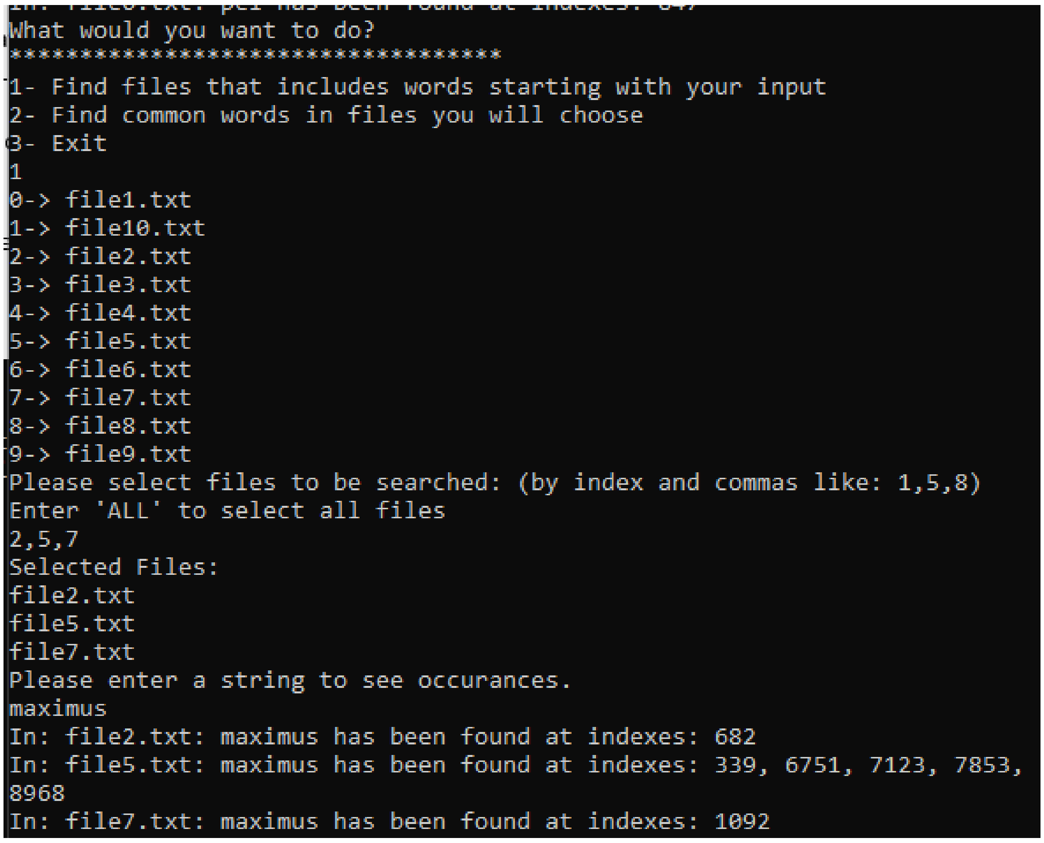
Here are 4 different examples of the code:

1- “pellentesque” is a common word for the files 3,4,5,6. When “pel” is given as an input here is the result:



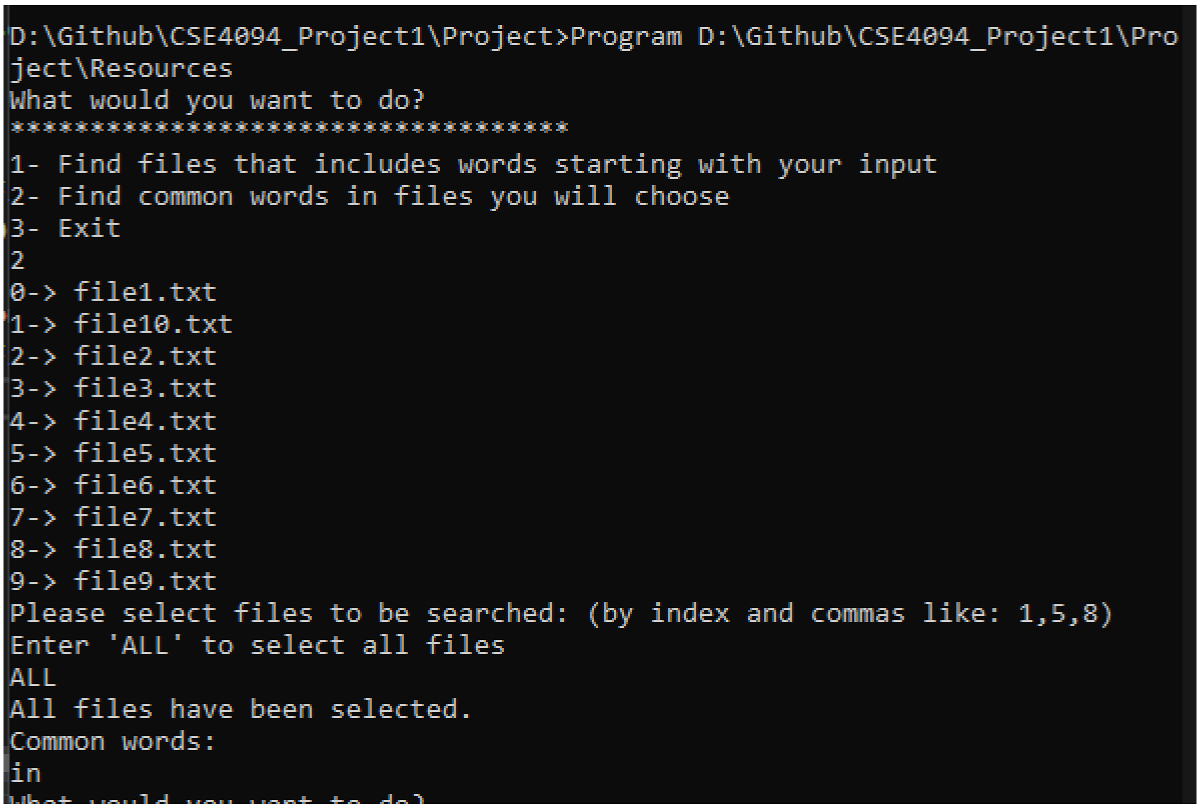
```
What would you want to do?
************************************
1- Find files that includes words starting with your input
2- Find common words in files you will choose
3- Exit
1
0-> file1.txt
1-> file10.txt
2-> file2.txt
3-> file3.txt
4-> file4.txt
5-> file5.txt
6-> file6.txt
7-> file7.txt
8-> file8.txt
9-> file9.txt
Please select files to be searched: (by index and commas like: 1,5,8)
Enter 'ALL' to select all files
3,4,5,6
Selected Files:
file3.txt
file4.txt
file5.txt
file6.txt
Please enter a string to see occurances.
pel
In: file3.txt: pel has been found at indexes: 278, 626, 664
In: file4.txt: pel has been found at indexes: 0, 481, 575
In: file5.txt: pel has been found at indexes: 347, 653, 1354, 2085, 2658,
 3250, 3451, 3727, 3929, 4063, 4177, 4450, 5428, 5705, 6769, 6819, 7340
In: file6.txt: pel has been found at indexes: 847
```

2- The first query type can find occurrences of the given input. That means it can also find full words:



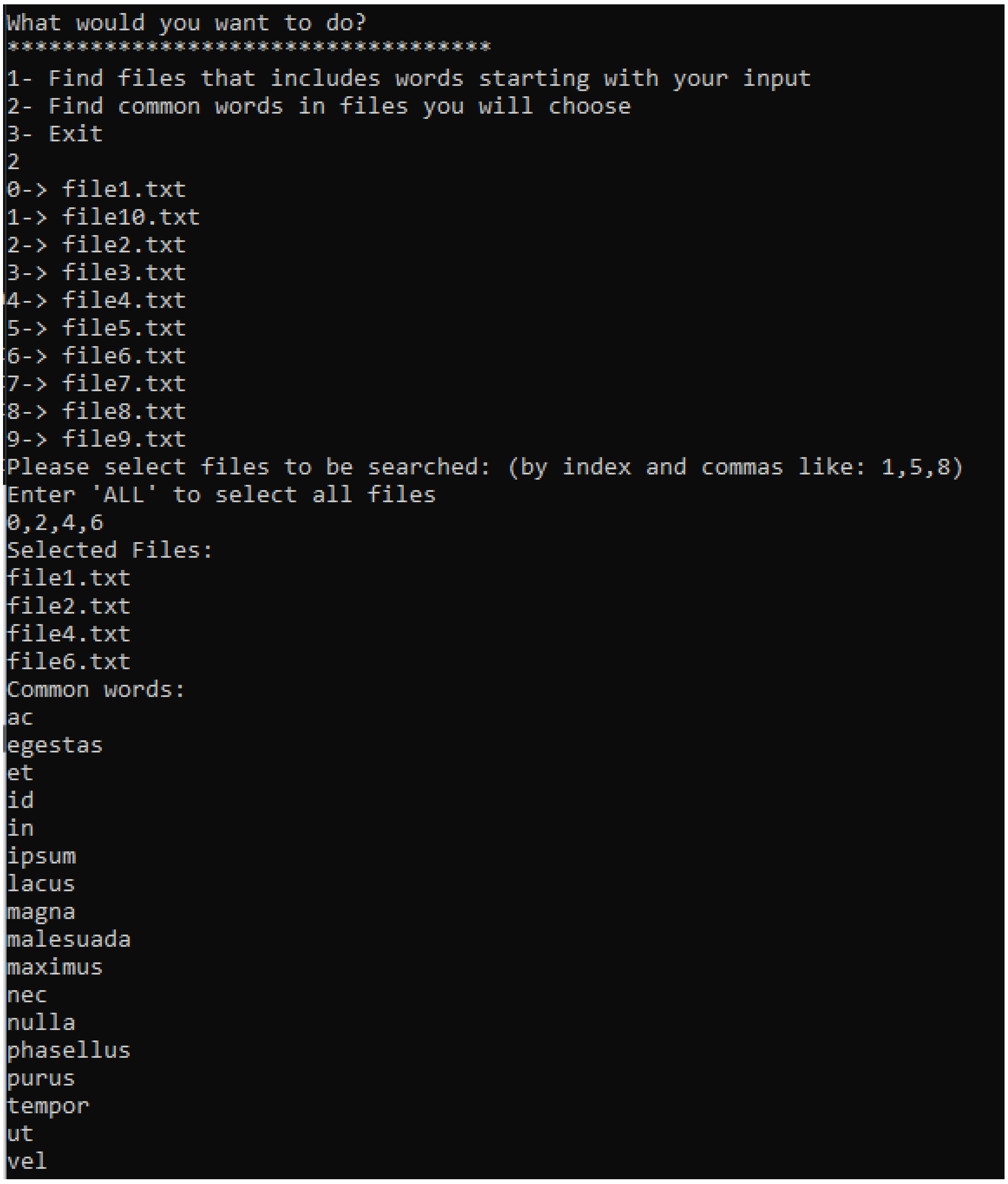
```
What would you want to do?
************************************
1- Find files that includes words starting with your input
2- Find common words in files you will choose
3- Exit
1
0-> file1.txt
1-> file10.txt
2-> file2.txt
3-> file3.txt
4-> file4.txt
5-> file5.txt
6-> file6.txt
7-> file7.txt
8-> file8.txt
9-> file9.txt
Please select files to be searched: (by index and commas like: 1,5,8)
Enter 'ALL' to select all files
2,5,7
Selected Files:
file2.txt
file5.txt
file7.txt
Please enter a string to see occurances.
maximus
In: file2.txt: maximus has been found at indexes: 682
In: file5.txt: maximus has been found at indexes: 339, 6751, 7123, 7853,
8968
In: file7.txt: maximus has been found at indexes: 1092
```

3- For the second query, if we run it for all the files, it will find that "in" is common for all files.



```
D:\Github\CSE4094_Project1\Project>Program D:\Github\CSE4094_Project1\Pro
ject\Resources
What would you want to do?
************************************
1- Find files that includes words starting with your input
2- Find common words in files you will choose
3- Exit
2
0-> file1.txt
1-> file10.txt
2-> file2.txt
3-> file3.txt
4-> file4.txt
5-> file5.txt
6-> file6.txt
7-> file7.txt
8-> file8.txt
9-> file9.txt
Please select files to be searched: (by index and commas like: 1,5,8)
Enter 'ALL' to select all files
ALL
All files have been selected.
Common words:
in
```

4- If we give individual files as an input:



```
What would you want to do?
***********************************
1- Find files that includes words starting with your input
2- Find common words in files you will choose
3- Exit
2
0-> file1.txt
1-> file10.txt
2-> file2.txt
3-> file3.txt
4-> file4.txt
5-> file5.txt
6-> file6.txt
7-> file7.txt
8-> file8.txt
9-> file9.txt
Please select files to be searched: (by index and commas like: 1,5,8)
Enter 'ALL' to select all files
0,2,4,6
Selected Files:
file1.txt
file2.txt
file4.txt
file6.txt
Common words:
ac
egestas
et
id
in
ipsum
lacus
magna
malesuada
maximus
nec
nulla
phasellus
purus
tempor
ut
vel
```

The program is menu-driven but I did not check exceptional inputs, so for some cases, it will give errors if you enter inappropriate inputs.