**Report**

**Why do we choose Trie structure?**

We chose Trie structure for both queries. For the first query, Trie holds the words starting from the beginning of the word character by character and in the first query we want to find given prefix. For the second query, this query still starts to search from the beginning of the word and finds exact matches among the given files. We don’t prefer to use suffix tree structure because suffix tree holds unnecessary patterns which we don’t need for this search.

**Details of the implementation of Trie structure:**

While we are building Trie, we took the path from the user and we open the files with the given path. We read the files one by one and we constructed the single Trie for all files. The necessary details are held in the nodes.

While we are adding the words to the Trie, we used nodes that includes current char value and children list which is a connection to the next char node. As an example, our current word is “*apple*”. We create a node which char value is “*a*” and this node is added to the children list of root. Then, we create a node which char value is “*p*” and it is added to children list of previous node. This process lasts until the children list is null.

We used pytrie library to implement the Trie in Python.

This construction of the Trie is used for both of queries. The time of this structure is O(m\*Lmax). Space complexity is also O(m\*Lmax).

**Details of the implementation of the first query:**

For the first query, we need to show the occurrences of the given prefix, hold name of the file that includes prefix and corresponding positions. We read the file word by word and we held the positions and file names in the dictionary as a node attribute.

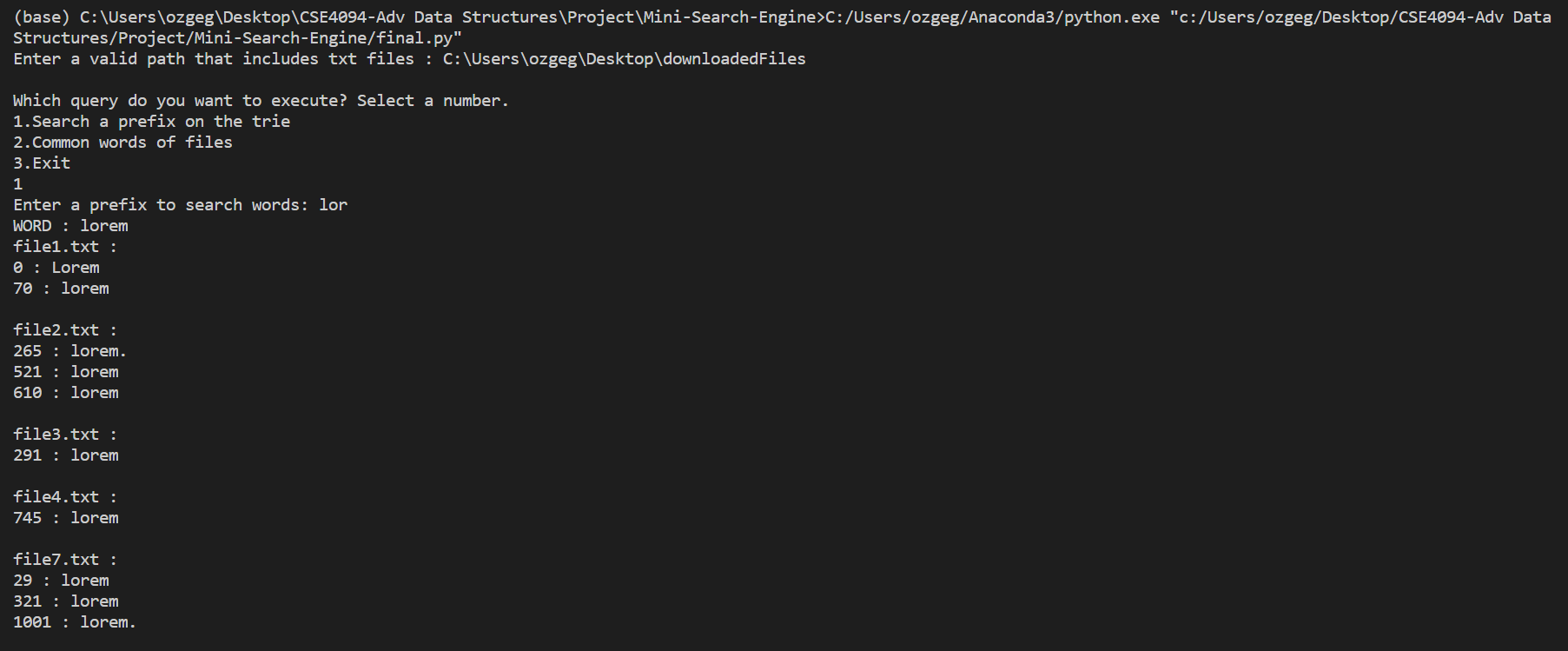
**Details of the implementation of the second query:**

For the second query, we need to show the common words for the specified files. We looked for the end of the words and their file list. If the file list includes given file names by the user, then it is a common word in specified files and we showed these words.

We created two functions to print the output in an order. These function names are printStr() and printWord(). printStr() prints the word values and calls the printWord() to print the word from the original file. printWord() takes the positions of the prefix in the files, files as a text, word and reads the text files with the specified positions in the files. It prints the original string in the files.

**Test Cases and Outputs:**

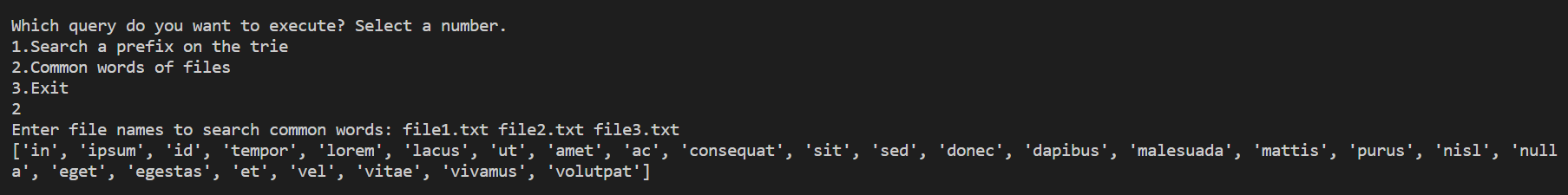
1. We tested our code with lorem ipsum text files. Our first test case is *‘lor’* prefix. These outputs show words, txt file names and positions for *‘lor’* prefix.



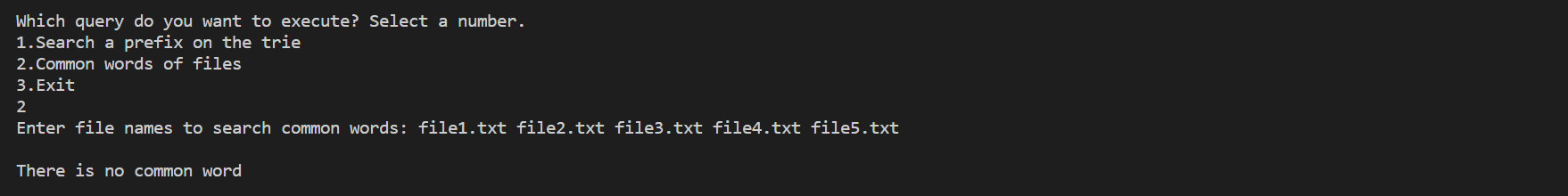
1. We tested our code with lorem ipsum text files. Our second test case is *‘ips’* prefix. These outputs show words, txt file names and positions for *‘ips’* prefix.
2. We tested our code with lorem ipsum text files. Our third test case is *‘co’* prefix. These outputs show words, txt file names and positions for *‘co’* prefix.

1. We tested our code with lorem ipsum text files. Our forth test case is common words with file1.txt file2.txt file3.txt. This output shows common words.



1. We tested our code with lorem ipsum text files. Our fifth test case is common words with file1.txt file2.txt file3.txt file4.txt file5.txt. This output shows nothing because file5.txt is an empty file.



1. We tested our code with lorem ipsum text files. Our fifth test case is *‘1’* prefix. These outputs show words, txt file names and positions for *‘1’* prefix.

