MARMARA UNIVERSITY FACULTY OF ENGINEERING

CSE 4094 Special Topics in Computer Engineering

Advanced Data Structures

Project 1 - Report

Mini Search Engine

GÖKHAN ÇULFACI – 150113027

OĞUZHAN BÖLÜKBAŞ - 150114022

Firstly, we search some data structures for this project and compared complexity time, implementation difficulty, applications, memory usage etc. each other. One of the data structures for this project is suffix tree.

All the pattern search algorithms preprocess the pattern to make the pattern searching faster. The best time complexity that we could get by preprocessing pattern is O(n) where n is length of the text. In this post, we will discuss an approach that preprocesses the text. A suffix tree is built of the text. After preprocessing text (building suffix tree of text), we can search any pattern in O(m) time where m is length of the pattern.

Suffix tree can be used for a wide range of problems. Generally using suffix tree provide optimal complexity time for some problems:

Patter Searching, finding the longest repeated substring, finding the longest common substring, finding the longest palindrome in a string.

The memory usage of suffix tree is higher than the actual size of the sequence collection. For a large text, construction may require external memory approaches

Implementation of suffix tree briefly.

How to search a pattern in the built suffix tree?

1) Starting from the first character of the pattern and root of Suffix Tree, do following for every character.

a) For the current character of pattern, if there is an edge from the current node of suffix tree, follow the edge.

b) If there is no edge, print “pattern doesn’t exist in text” and return.

2) If all characters of pattern have been processed, i.e., there is a path from root for characters of the given pattern, then print “Pattern found”.[1]

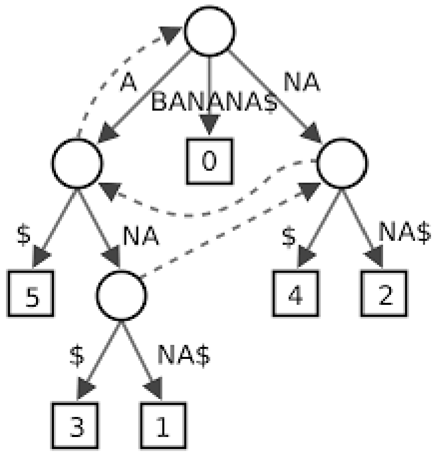


Figure 1: Sample of suffix tree [2]

It looks not bad but we need include words start with “xyz“ and that character number in file and that file name. So we need more junction point each nodes and suffix trees have less junction point than the other data structures(trie and compressed trie).

One of the data structures for this project was suffix tree. Second one is compressed trie.

Actually, compressed trie looks like regular trie but there is a little effective different that memory usage. In terms of runtime complexity, compressed trie tree is same as that regular trie tree. In terms of memory, a compressed trie tree uses very few numbers of nodes which gives you a huge memory advantage especially for long strings with long common prefixes. In terms of speed, a regular trie would be slightly faster because its operations don’t involve any string operations, they are simple loops.

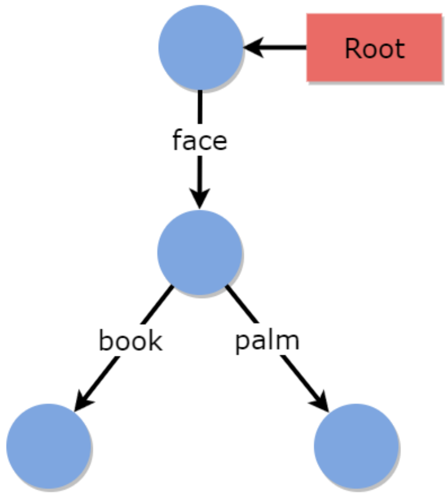


Figure 2: Sample of compressed trie [3]

Compressed trie can be use for this project but compressed trie already looks like regular trie and we don't need earn much more memory space because we don't have long strings. And we searched and found more source (information and implementation different programming language) for trie than compressed trie in internet so we selected trie for this project.

Tries is a tree that stores strings. Maximum number of children of a node is equal to size of alphabet. Trie supports search, insert and delete operations in O(L) time where L is length of key.

Why we are using trie ?

1. With Trie, we can insert and find strings in O(L) time where L is length of a single Word. This is obviously faster that BST. This is also faster than hashing because of the ways it is implemented. We don't need to compute any hash function. No collision handling is required.
2. Another advantage of trie is, we can easily print all words in alphabetical order which is not easily possible with hashing
3. We can efficiently do prefix search with trie.

Implementation of trie briefly

How to search a pattern in the built Trie?

Following are steps to search a pattern in the built Trie.

1) Starting from the first character of the pattern and root of the Trie, do following for every character.

a) For the current character of pattern, if there is an edge from the current node, follow the edge.

b) If there is no edge, print “pattern doesn’t exist in text” and return.

2) If all characters of pattern have been processed, i.e., there is a path from root for characters of the given pattern, then print print all indexes where pattern is present. To store indexes, we use a list with every node that stores indexes of suffixes starting at the node.[4]

Tries have a lot of advantages like above but have disadvantages and main disadvantage of tries is that they need lot of memory for storing the strings. For each node have too many node pointers. But it is not problem for our project.

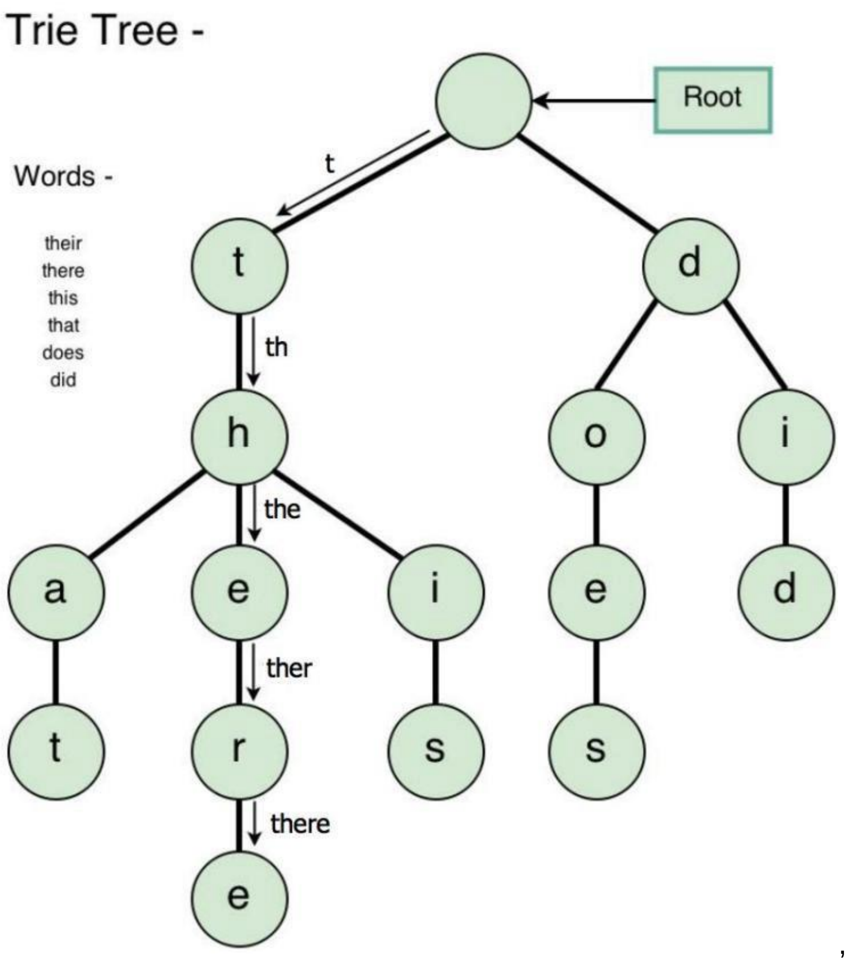


Figure 3: Sample of trie [5]

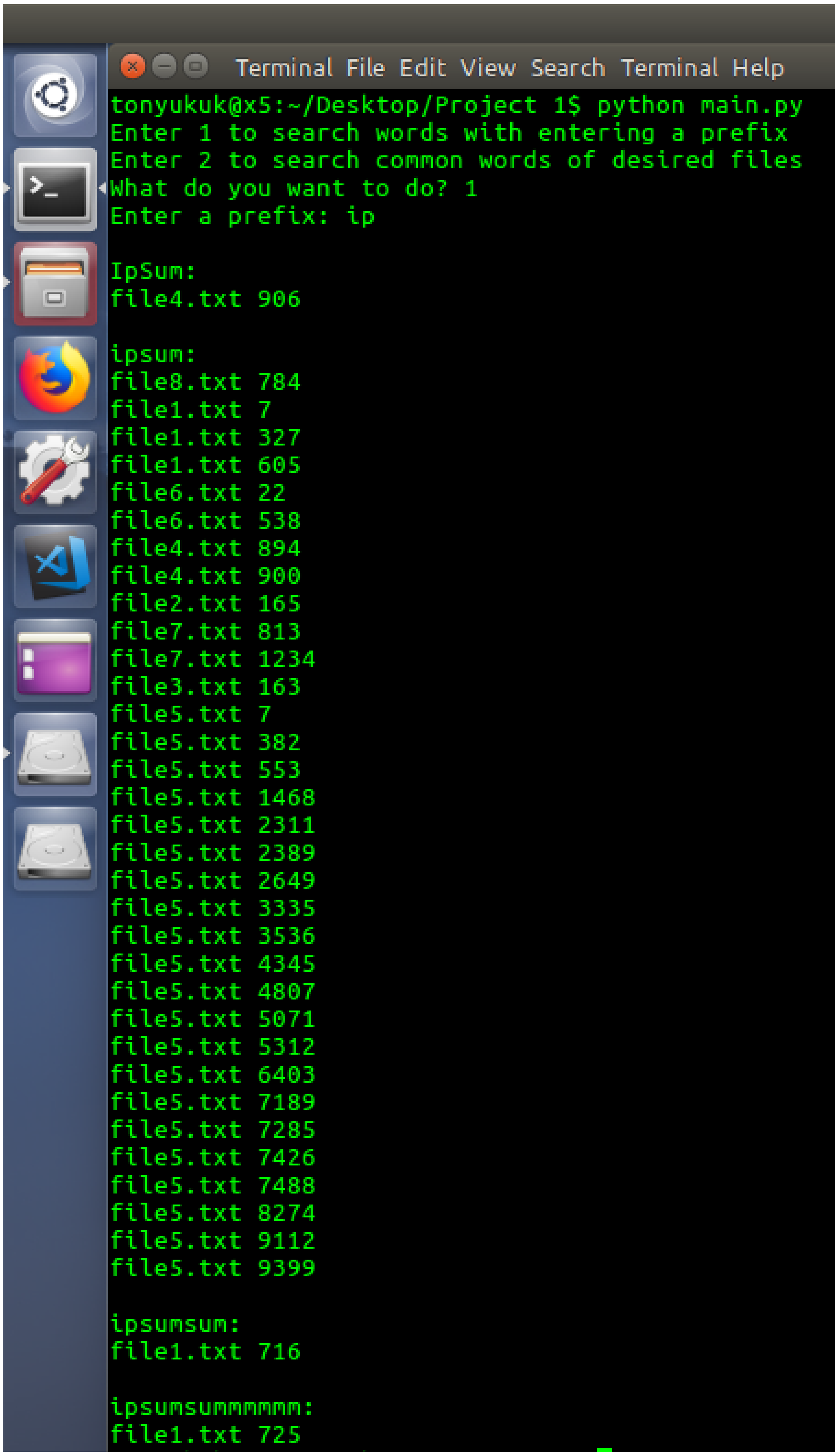
Finally, trie is faster than other data structure models but require huge memory.

After selected data structure model, we implemented and get some output. We can see some input, output and implementation below this paragraph

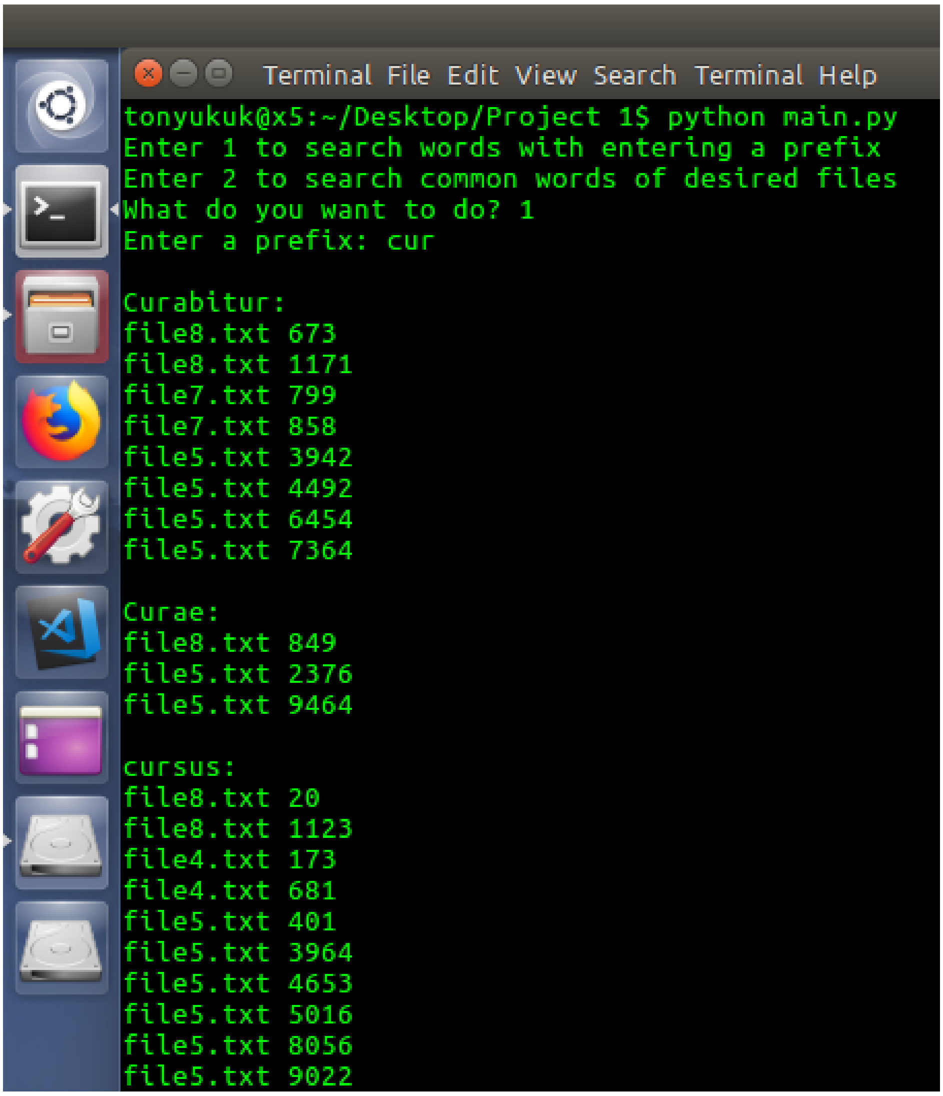
IMPLEMENTATION AND OUTPUTS

1. Finding words which are started with obtained prefix

   a. Searching prefix “ip”

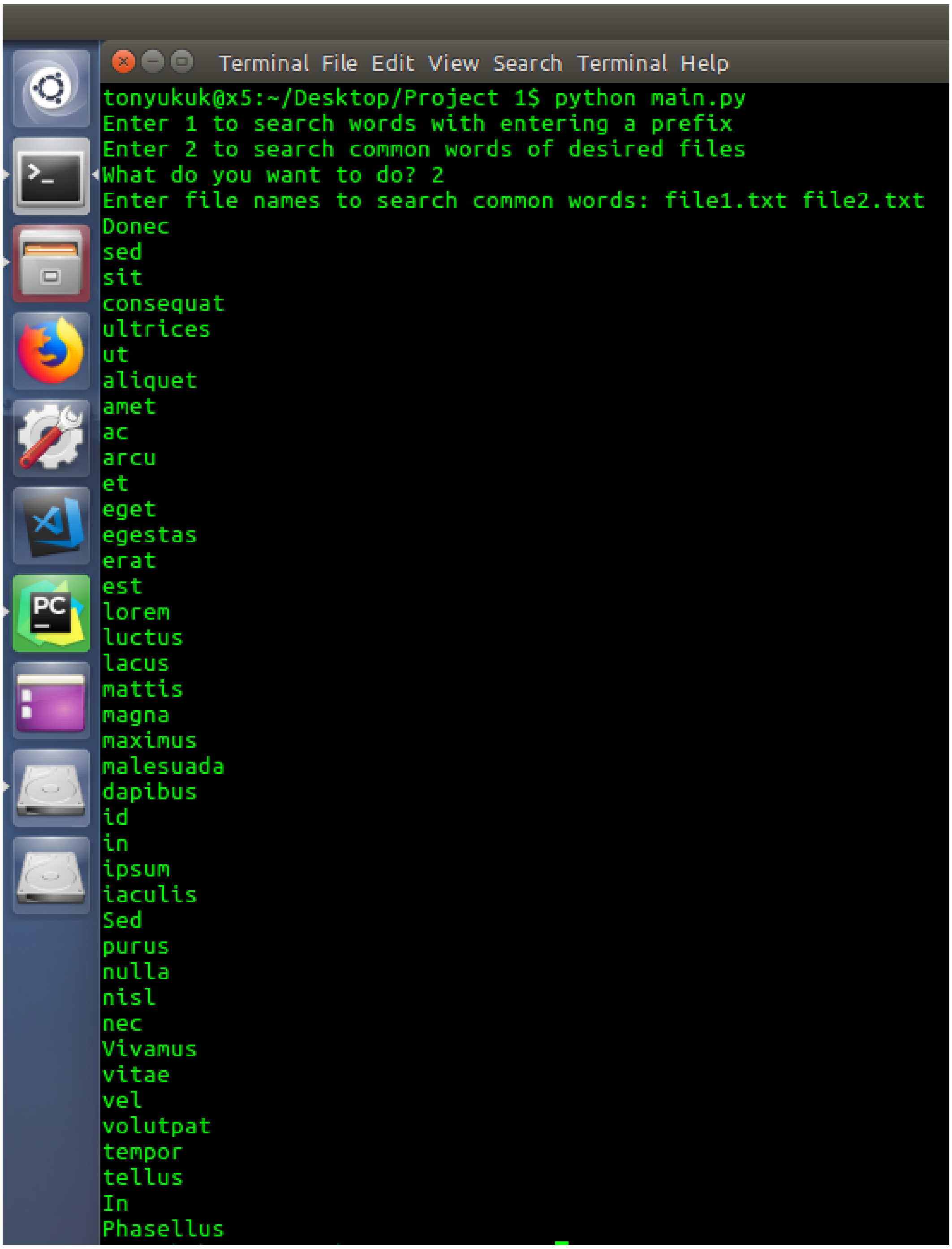


b. Searching prefix “cur”

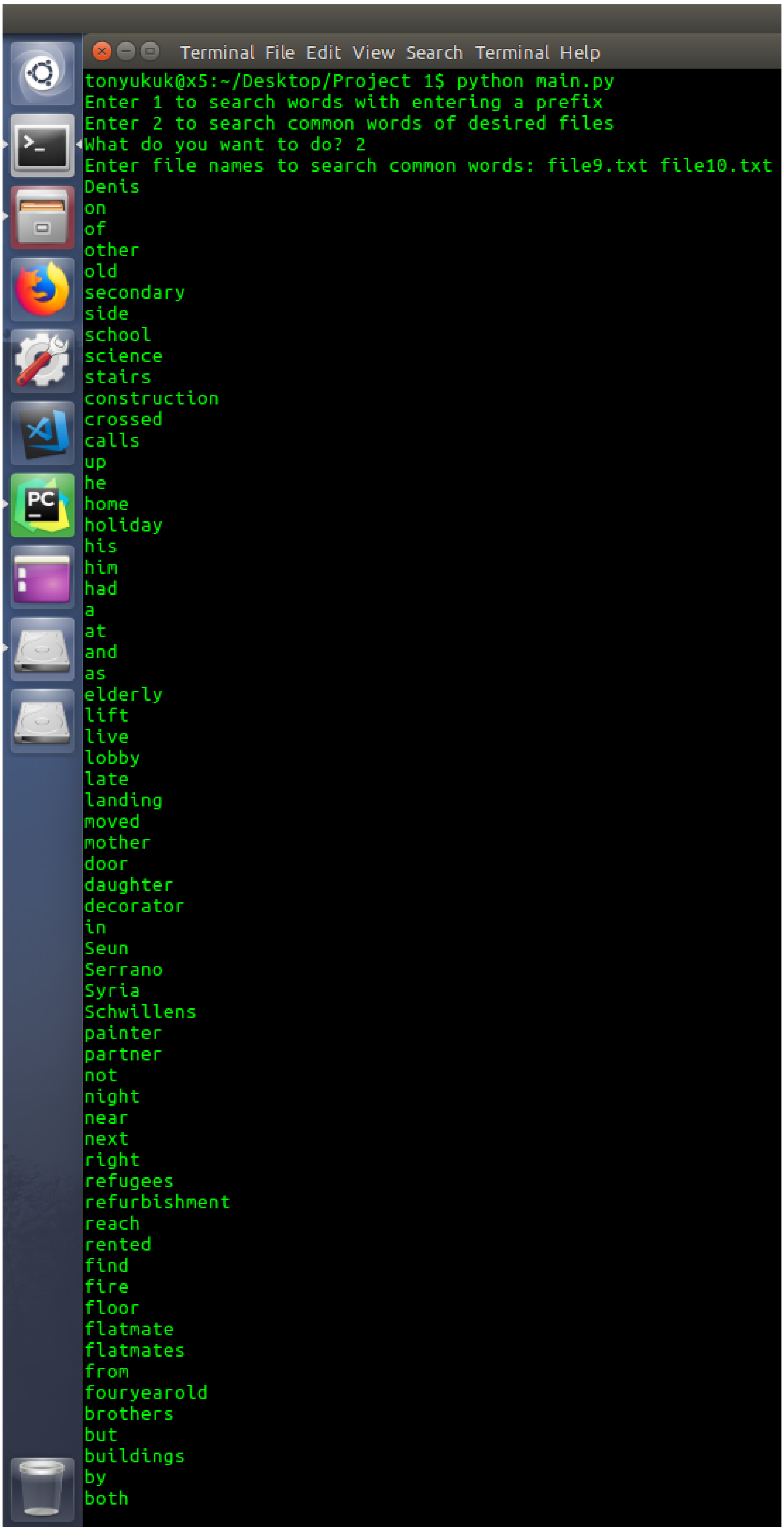


2. Finding common words in obtained files

   a. Common words of file1.txt and file2.txt



b. Common words of file9.txt file10.txt




Terminal File Edit View Search Te
Syria
Schwillens
painter
partner
not
night
near
next
right
refugees
refurbishment
reach
rented
find
fire
floor
flatmate
flatmates
from
fouryearold
brothers
but
buildings
by
both
been
Murphy
Mohammed
teacher
to
took
tower
the
their
they
If
Alhaj
Ali
Alejandro
you
youd
14th
13
111
112
113
114
1997
June
2017
23yearold
Flat
That
The
This
Talabi
was
who
working
worked
were
Omar
Oluwaseun
Oyewole
Rosemary
Robert
But

REFERENCES

[1] Pattern Searching using a Suffix Tree, Building a suffix tree, Search a pattern in the built suffix tree

https://www.geeksforgeeks.org/pattern-searching-using-suffix-tree/

[2] Suffix Tree, Suffix Tree for the text BANANA, 28 January 2019

https://en.wikipedia.org/wiki/Suffix_tree

[3] Compressed Trie Tree, Compressing the tree, Example of compressed trie tree, 15 November 2016

http://theoryofprogramming.com/2016/11/15/compressed-trie-tree/

[4] Pattern Searching using a Trie of all Suffixes, Building a trie of suffixes, Search a pattern in the built trie.

https://www.geeksforgeeks.org/pattern-searching-using-trie-suffixes/

[5] Trie Tree Implementation, Structure of trie tree, Example of trie tree, 16 January 2015

http://theoryofprogramming.com/2015/01/16/trie-tree-implementation/