

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:

Pattern 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]

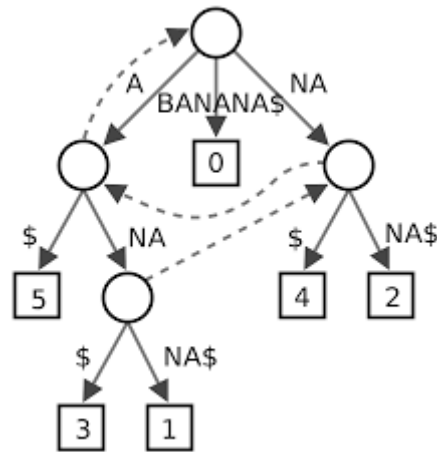


Figure 1: Sample of suffix tree [2]

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.

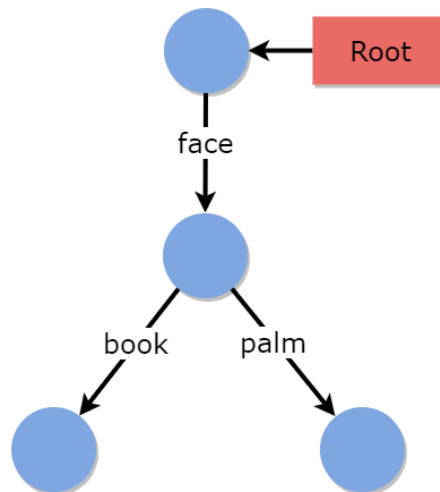


Figure 2: Sample of compressed trie [3]

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

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

Why we are using trie ?

1. With Trie, we can insert and find strings in $O(L)$ time where L is the length of a single word. This is obviously faster than BST. This is also faster than hashing because of the way 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.

Words -

their
there
this
that
does
did

```

graph TD
    Root[Root] --> t((t))
    Root --> d((d))
    t -->|th| h((h))
    t --> i1((i))
    h --> a((a))
    h -->|the| e1((e))
    h --> i2((i))
    e1 -->|ther| r((r))
    e1 --> s1((s))
    r -->|there| e2((e))
    d --> o((o))
    d --> i3((i))
    o --> e3((e))
    o --> s2((s))
    i3 --> d2((d))
  
```

Words -

their
there
this
that
does
did

Figure 3: Sample of trie [5]

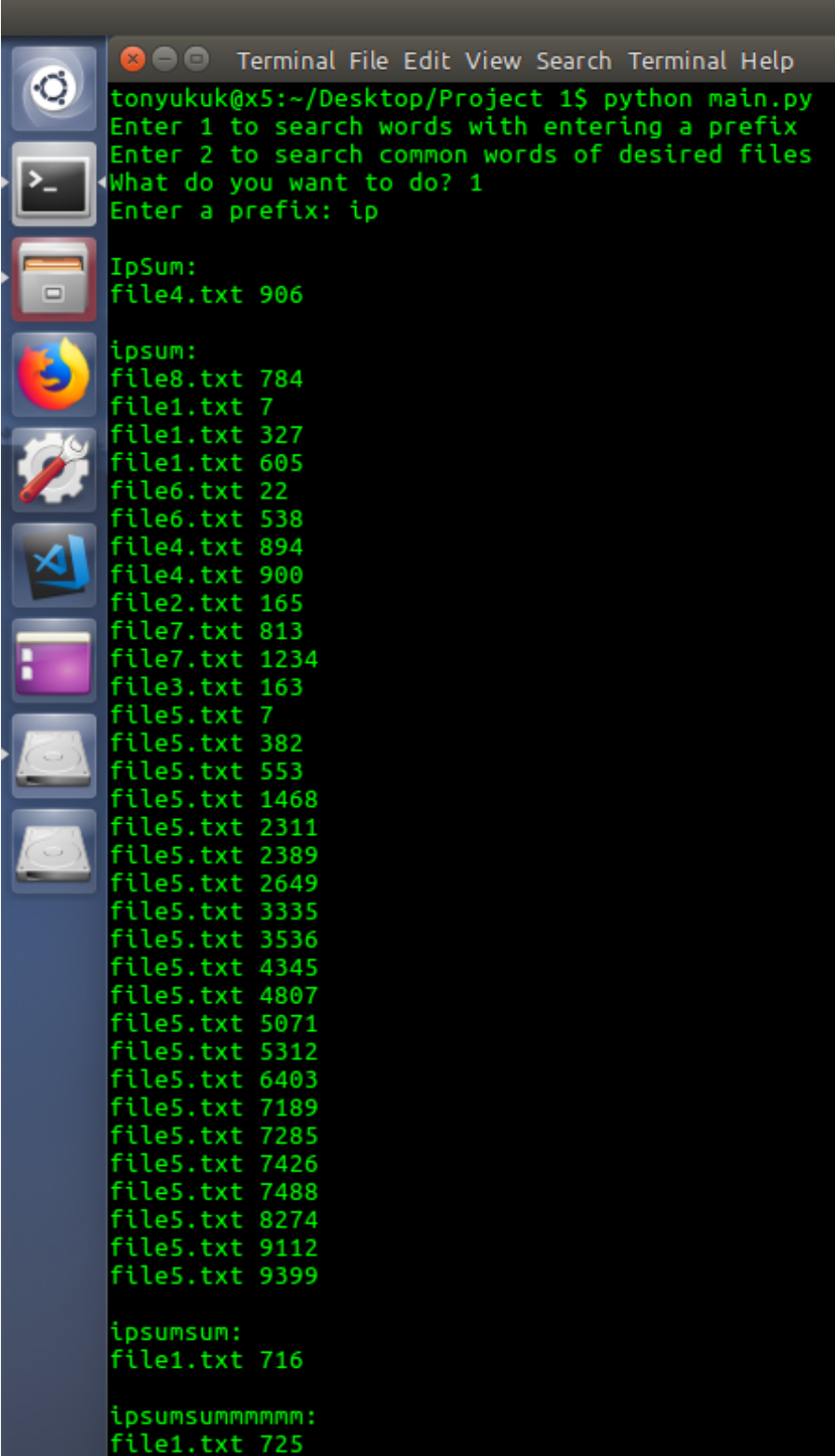
Finally, trie is faster than other data structure models but require huge memory.

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"



```
tonyukuk@x5:~/Desktop/Project 1$ python main.py
Enter 1 to search words with entering a prefix
Enter 2 to search common words of desired files
What do you want to do? 1
Enter a prefix: ip

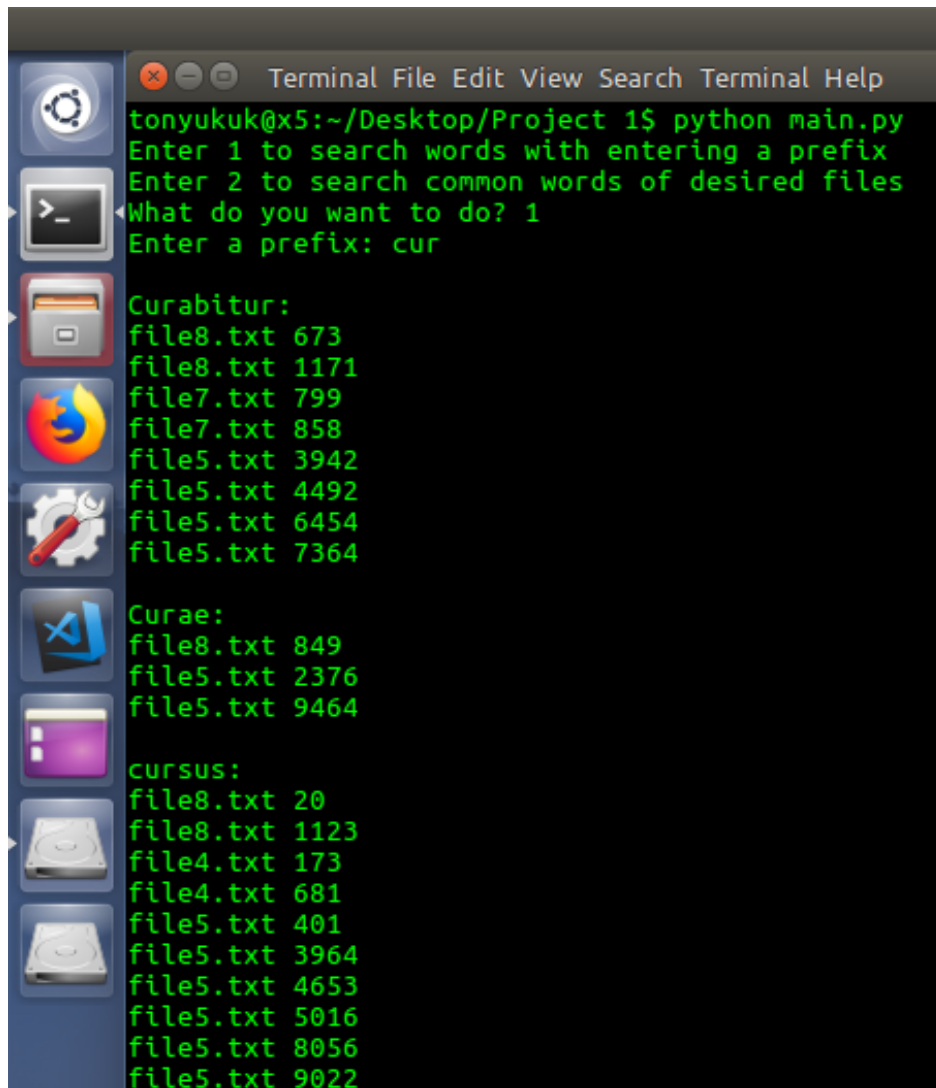
IpSum:
file4.txt 906

ipsum:
file8.txt 784
file1.txt 7
file1.txt 327
file1.txt 605
file6.txt 22
file6.txt 538
file4.txt 894
file4.txt 900
file2.txt 165
file7.txt 813
file7.txt 1234
file3.txt 163
file5.txt 7
file5.txt 382
file5.txt 553
file5.txt 1468
file5.txt 2311
file5.txt 2389
file5.txt 2649
file5.txt 3335
file5.txt 3536
file5.txt 4345
file5.txt 4807
file5.txt 5071
file5.txt 5312
file5.txt 6403
file5.txt 7189
file5.txt 7285
file5.txt 7426
file5.txt 7488
file5.txt 8274
file5.txt 9112
file5.txt 9399

ipsumsum:
file1.txt 716

ipsumsumMMMMM:
file1.txt 725
```

b. Searching prefix “cur”



The image shows a terminal window titled "Terminal File Edit View Search Terminal Help". The user is running a Python script named "main.py" in the directory "~/Desktop/Project 1". The script prompts the user to enter a number (1 or 2) to choose a search method. The user enters "1". The script then prompts the user to enter a prefix. The user enters "cur". The script then displays the results for the prefix "cur", showing the number of occurrences in various files. The results are grouped by prefix: "Curabitur:", "Curae:", and "cursus:". Each group lists the files and the count of occurrences.

```
tonyukuk@x5:~/Desktop/Project 1$ python main.py
Enter 1 to search words with entering a prefix
Enter 2 to search common words of desired files
What do you want to do? 1
Enter a prefix: cur

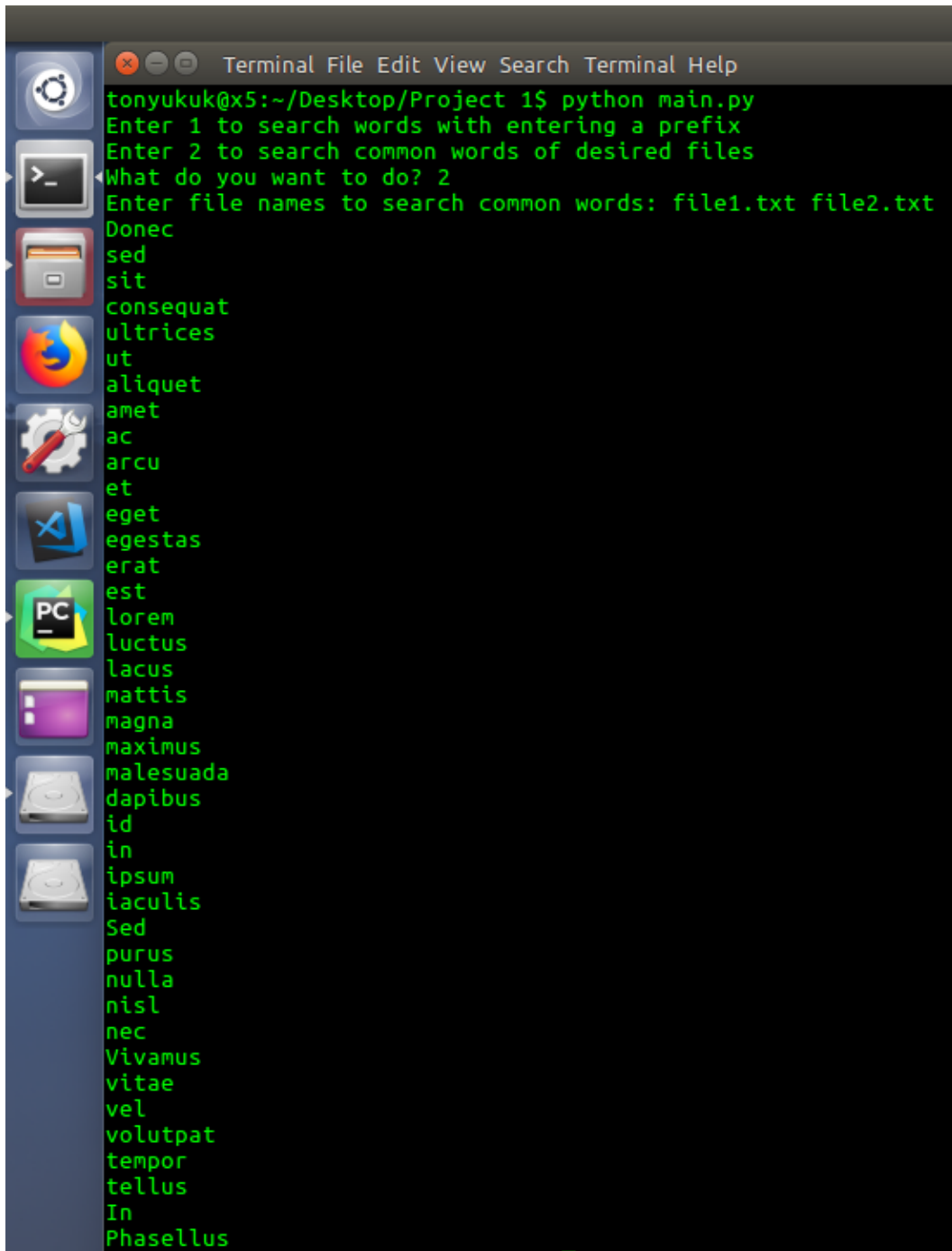
Curabitur:
file8.txt 673
file8.txt 1171
file7.txt 799
file7.txt 858
file5.txt 3942
file5.txt 4492
file5.txt 6454
file5.txt 7364

Curae:
file8.txt 849
file5.txt 2376
file5.txt 9464

cursus:
file8.txt 20
file8.txt 1123
file4.txt 173
file4.txt 681
file5.txt 401
file5.txt 3964
file5.txt 4653
file5.txt 5016
file5.txt 8056
file5.txt 9022
```

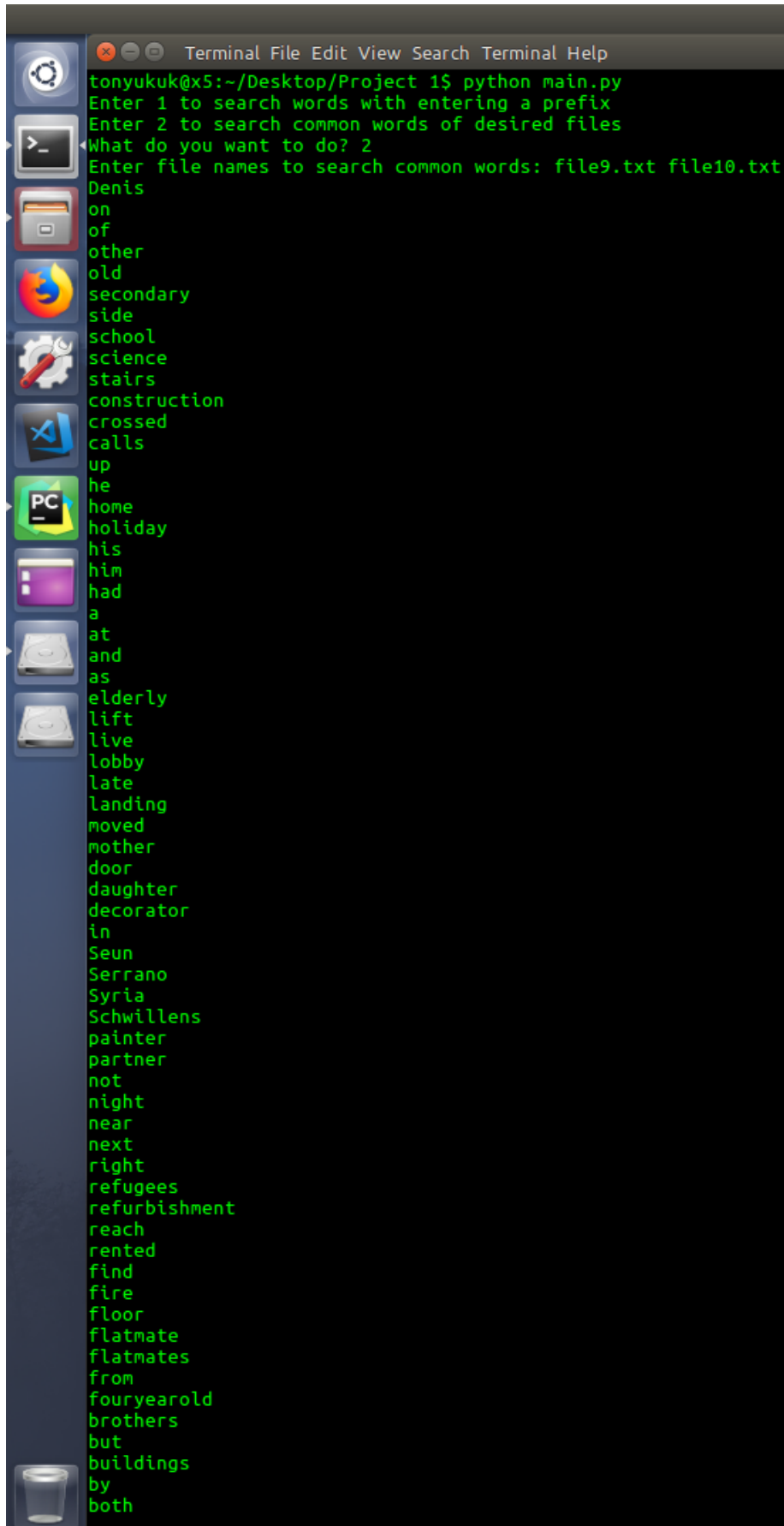

2. Finding common words in obtained files

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

A screenshot of a Linux terminal window. The window has a title bar with 'Terminal' and menu options 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is 'tonyukuk@x5:~/Desktop/Project 1\$'. The user has run 'python main.py'. The script displays instructions: 'Enter 1 to search words with entering a prefix', 'Enter 2 to search common words of desired files', and 'What do you want to do? 2'. The user has entered '2'. The script then prompts 'Enter file names to search common words: file1.txt file2.txt'. The user has entered 'file1.txt file2.txt'. The script then outputs a list of common words: Donec, sed, sit, consequat, ultrices, ut, aliquet, amet, ac, arcu, et, eget, egestas, erat, est, lorem, luctus, lacus, mattis, magna, maximus, malesuada, dapibus, id, in, ipsum, iaculis, Sed, purus, nulla, nisl, nec, Vivamus, vitae, vel, volutpat, tempor, tellus, In, and Phasellus.

```
tonyukuk@x5:~/Desktop/Project 1$ python main.py
Enter 1 to search words with entering a prefix
Enter 2 to search common words of desired files
What do you want to do? 2
Enter file names to search common words: file1.txt file2.txt
Donec
sed
sit
consequat
ultrices
ut
aliquet
amet
ac
arcu
et
eget
egestas
erat
est
lorem
luctus
lacus
mattis
magna
maximus
malesuada
dapibus
id
in
ipsum
iaculis
Sed
purus
nulla
nisl
nec
Vivamus
vitae
vel
volutpat
tempor
tellus
In
Phasellus
```

b. Common words of file9.txt file10.txt



```
Terminal File Edit View Search Terminal Help
tonyukuk@x5:~/Desktop/Project 1$ python main.py
Enter 1 to search words with entering a prefix
Enter 2 to search common words of desired files
What do you want to do? 2
Enter file names to search common words: file9.txt file10.txt
Denis
on
of
other
old
secondary
side
school
science
stairs
construction
crossed
calls
up
he
home
holiday
his
him
had
a
at
and
as
elderly
lift
live
lobby
late
landing
moved
mother
door
daughter
decorator
in
Seun
Serrano
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
```



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/>