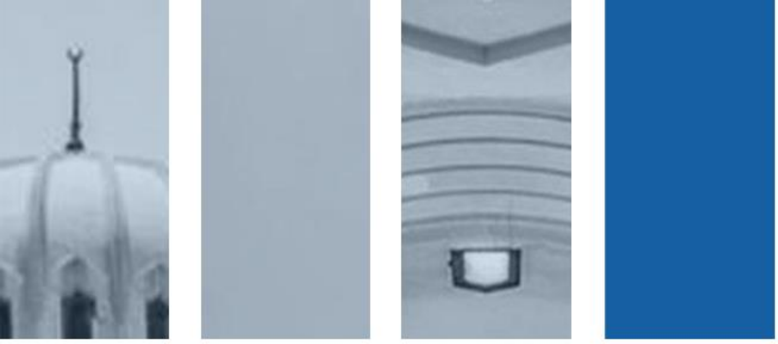


Inverted Index

Program:

Course Code: CSE323

Course Name: programming with Data structure

Examination Committee

Prof. Hosam Fahmy

Dr. Islam El-Maddah

Ain Shams University
Faculty of Engineering
Spring Semester – 2020

mv

Student Personal Information for Group Work

Student Names:	Student Codes:
محمود محمد بنيامين محمد	1501367
محمد ياسر أحمد حنفي العسيلي	1601287
محمد حمادة محمد محمود	17X0040
محمد عبادة بهاء محمد	16W0061
مصطفي أشرف محمد عبدالعظيم	16T0107

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: Mostafa Ashraf / Mohamed Obada / mohamed Yasser Date: 31 /5 / 2020

Mohamed hamada / Mahmoud Mohamed

Submission Contents

Contents

01

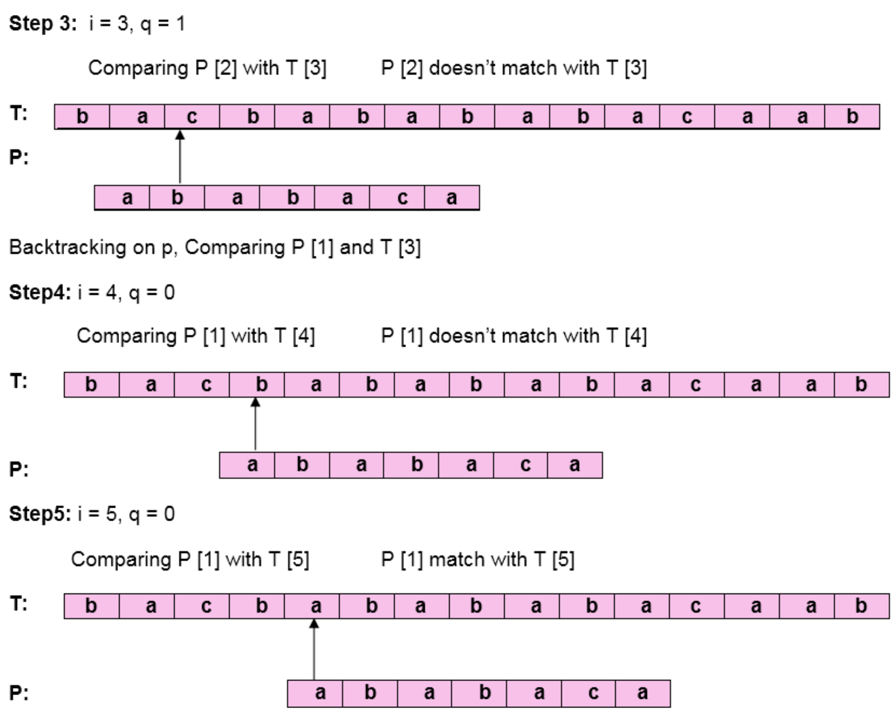
First Topic

Background

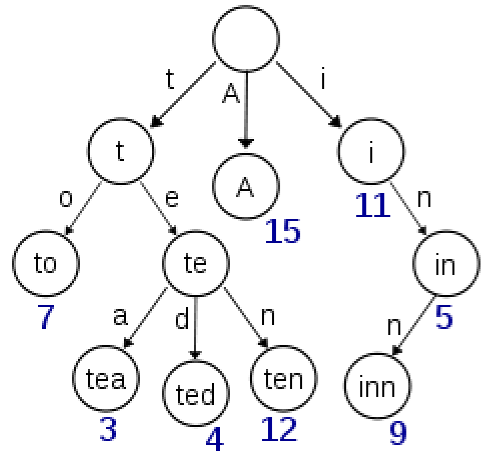
Introduction

Searching about words in a large dataset is hard to balance between searching speed and optimize using memory. By using string matching algorithms, it will take a lot time but using low memory in large files dataset example:

KMP algorithm (fast string-matching algorithms) it has complexity O(m*n)



which m is word length and n is text length, hence each time to search word it will take O(m*n). on other hand, using trie data structure will save time and optimize using memory. Trie is a data structure type whose nodes store the letters of an alphabet. It builds only once to read files' dataset. The Complexity of searching is O(m).



Restructure data in all files

comparison between Forward and Inverted index

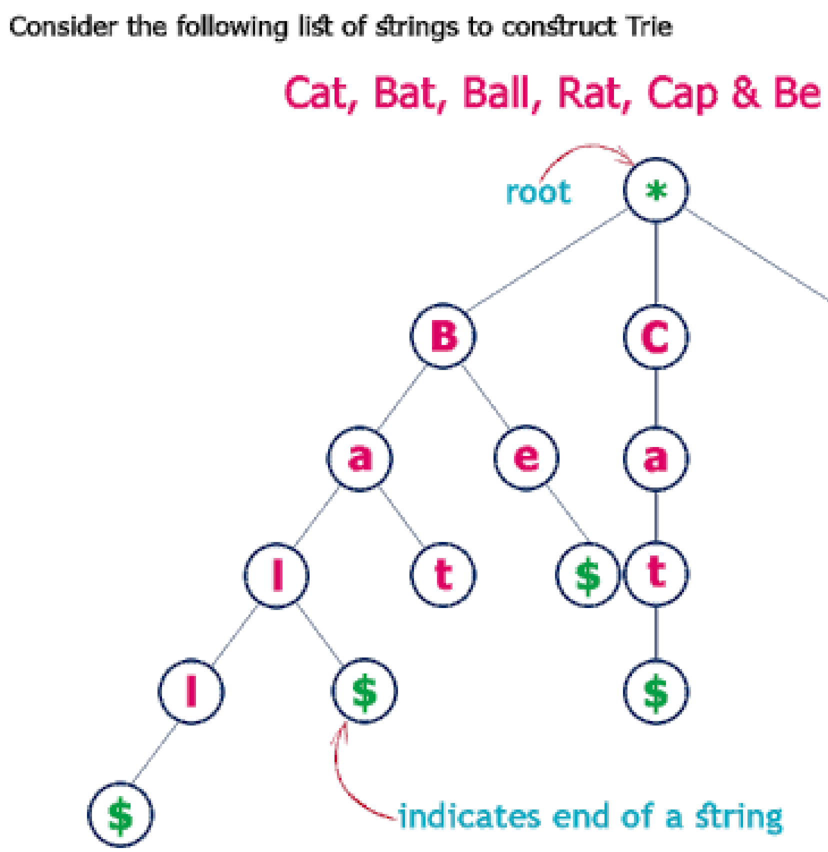
point of comparison	**forward index**	**inverted index**
how it works	it uses the document name as the key for the words inside (mapped values)	it uses the words as the keys and document names as the mapped value
building process	get document name read all words inside extract words not repeated map them where the document name is the key and the unique words are the mapped value	• get document name • read all the words inside • extract unique words • map each unique word as the key and the document name as the mapped value
how much time does the indexing take	the indexing is fast as it only maps the words inside one key	the indexing is slow as you have to check the words then add it as a key
repeated words	there is as the words may be in many documents	there aren't as repeated words not added to index
searching	searching takes time as you have to see if it exists in every document	searching is fast as there isn't repeated words
examples	document 1 hello my friend document 2 hello world in c document 3 drink coffee each morning *note that hello is repeated	hello document 1, document 2 friend document 1 coffee document 3
real world examples	Table of contents in book. DNS lookup	A glossary at the end of the index Reverse Lookup.
notes		indexing can be faster if used with a Trie or similar data structure

Trie

Definition:

This abstract is based on the tree data structure used in an efficient form. With high amount of documentation used in the world; it is easier to retrieve a document if organization is done properly. So, it is also important to classify the data into different categories efficiently.

A trie is a tree-like data structure whose nodes store the letters of an alphabet. By structuring the nodes in a way, words and strings can be retrieved from the structure by traversing down a branch path of the tree.



The shape and the structure of a trie is always a set of linked nodes, connecting back to an empty root node. An important thing to note is that the number of child nodes in a trie depends completely upon the total number of values possible. For example, if we are representing the English alphabet, then the total number of child nodes is directly connected to the total number of letters possible. In the English alphabet, there are 26 letters, so the total number of child nodes will be 26 if we are using a static implementation

Searching

1. searching is the same as in setting, the only difference is that now, new memory need not be allocated.

2. if a NULL is reached by travelling from the root even if the extracted word isn't completely searched, it means that the word does not exist in the loaded file.

3. if the word is processed completely and the last character leads the traversing to a node whose value attribute is 1. it means that the word is present in the file.

4. if the words last character does not lead us to a node with value attribute set to 1.

it means that the word does not exist in the file.

Advantages

- Using memory efficiently.
- Complexity of word search is O(L) where L is length of word.
- Another advantage of Trie is, we can easily print all words in alphabetical order which is not easily possible with hashing.
- We can efficiently do prefix search (or auto-complete) with Trie.

Disadvantages

Memory for a trie is a little more involved. In the worst case, there is one node per *character*. All those little node allocations start adding up.

Linked List

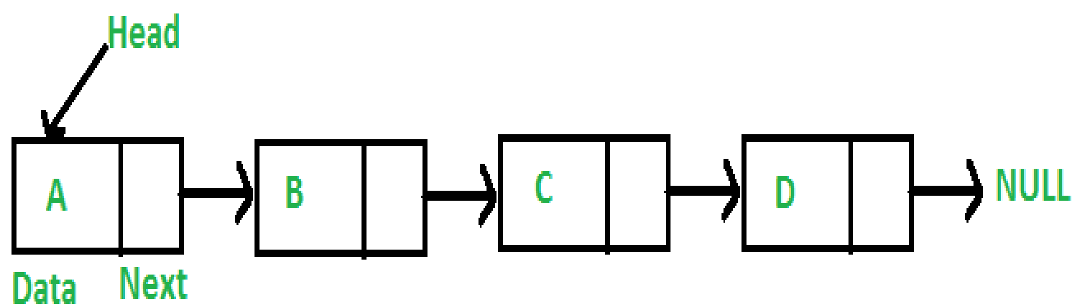
Definition

Linked List is a data structure type that presents data in a sequence of nodes that are linked together. Complexity for insertion front is O (1), back is O(n).

When considering lists, we can speak about-them on different levels - on a very abstract level (on which we can define what we mean by a list), on a level on which we can depict lists and communicate as humans about them, on a level on which computers can communicate, or on a machine level in which they can be implemented.

Graphical Representation

Non-empty lists can be represented by two-cells, in each of which the first cell contains a pointer to a list element and the second cell contains a pointer to either the empty list or another two-cell. We can depict a pointer to the empty list by a diagonal bar or cross through the cell.



Advantages

- No need for a free block of memory to store an array.
- Linked List is Dynamic data Structure.
- Linked List can grow and shrink during run time.
- Insertion and Deletion Operations are Easier
- Efficient Memory Utilization, i.e. no need to pre-allocate memory
- Faster Access time, can be expanded in constant time without memory overhead
- Linear Data Structures such as Stack, Queue can be easily implemented using Linked lists.

Disadvantages

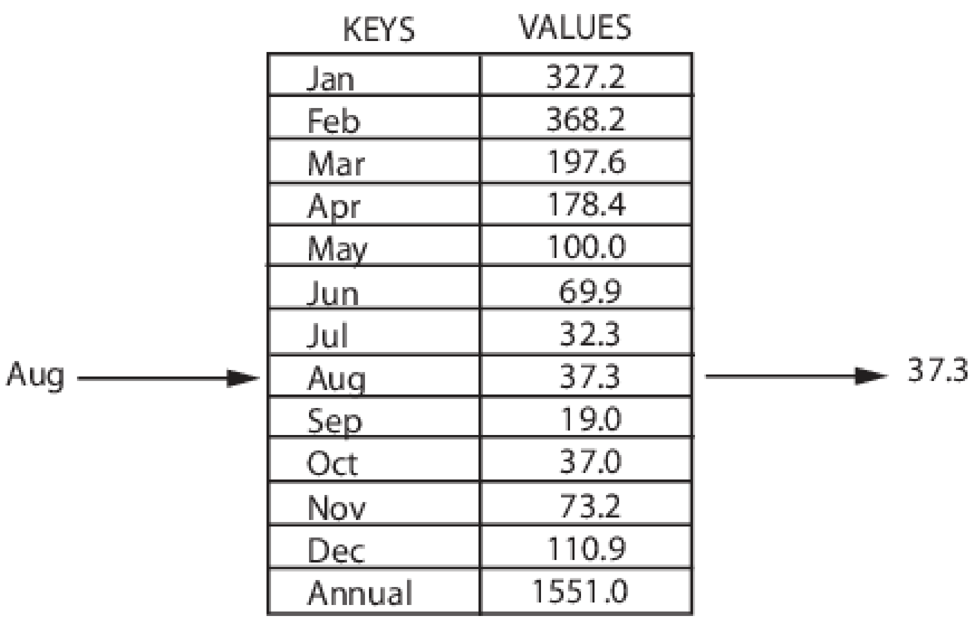
- Each node contains data and pointer to next node, so memory doesn't used efficiently
- Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back-pointer.
- Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
- Nodes aren't stored contiguously, greatly increasing the time required to access individual elements within the list, especially with a CPU cache.

Unordered map

definition

Unordered maps are associative containers that store elements formed by the combination of a *key value* and a *mapped value*, and which allows for fast retrieval of individual elements based on their keys. In an unordered map, the *key value* is generally used to uniquely identify the element, while the *mapped value* is an object with the content associated to this *key*. Types of *key* and *mapped* value may differ. Complexity Average case is O (1) and worst case is O(n).

An unordered map is an ADT (Abstract Data Type) where key-value pairs (k-v) are stored in an array. The 'key' is an identifier for some kind of data, and the 'value' is the content that is being identified or saved. Each locker in your school has a unique key or a unique combination lock. If you were to list the locker number and the key for each locker, the 'key' would be the locker number and the 'value' would be the number of the combination lock or the number of the key that belongs to each locker.



When to use the unordered map Data Type?

unordered map data type is ideal to use in lookup type situations where there is an identifying value and an actual value that is represented by the identifying value. A few examples where the map data type can be used are:

- Student ID numbers and last names
- House numbers on a street and the number of pets in each house
- Postal codes and names of cities
- Driving licenses and last names

Methods on unordered map

A lot of functions are available which work on unordered map. Most useful of them are – operator =, operator [], empty and size for capacity, begin and end for iterator, find and count for lookup, insert and erase for modification.

The C++11 library also provides functions to see internally used bucket count, bucket size and used hash function and various hash policies but they are less useful in real application.

Advantages

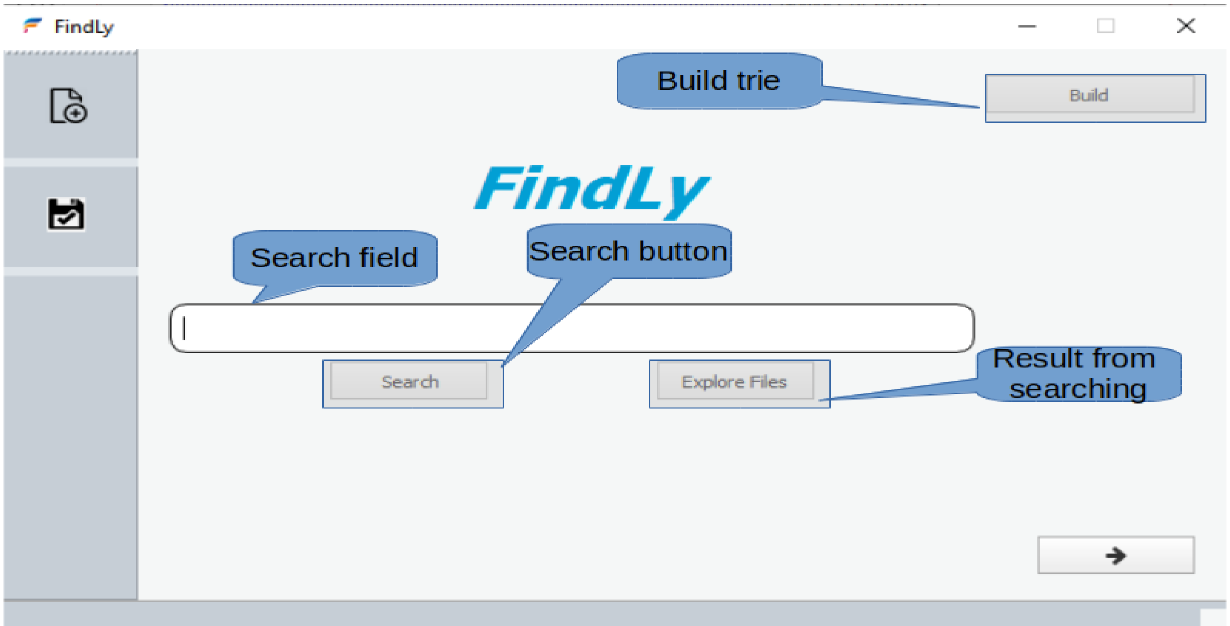
- Main advantage is synchronization.
- Insertion and Deletion is almost constant.
- In many situations, unordered map turns out to be more efficient than search trees or any other table lookup structure. For this reason, they are widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches and sets.

Disadvantages

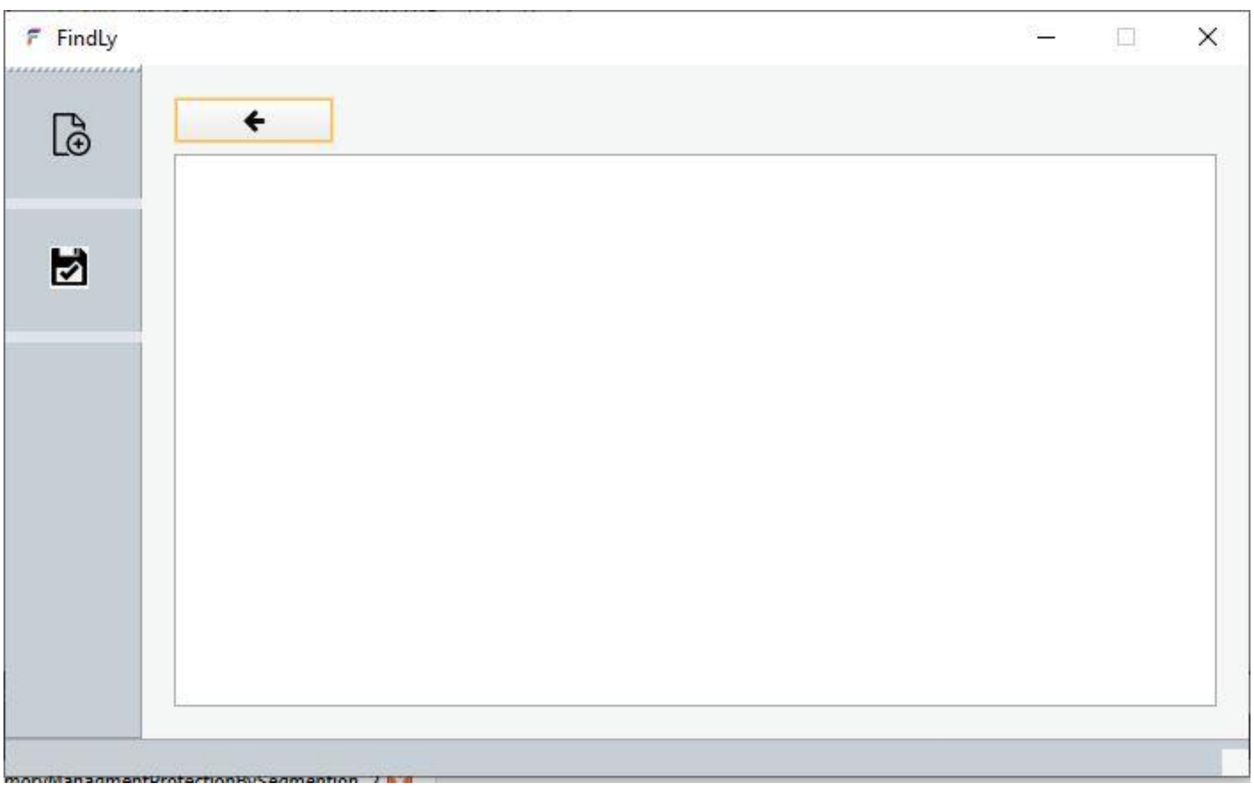
- Unordered maps are practically unavoidable. when hashing a random subset of a large set of possible keys.
- Unordered map becomes quite inefficient when there are many collisions.
- Unordered map does not allow null values.

Pictures of what our GUI looks like

- Main Section

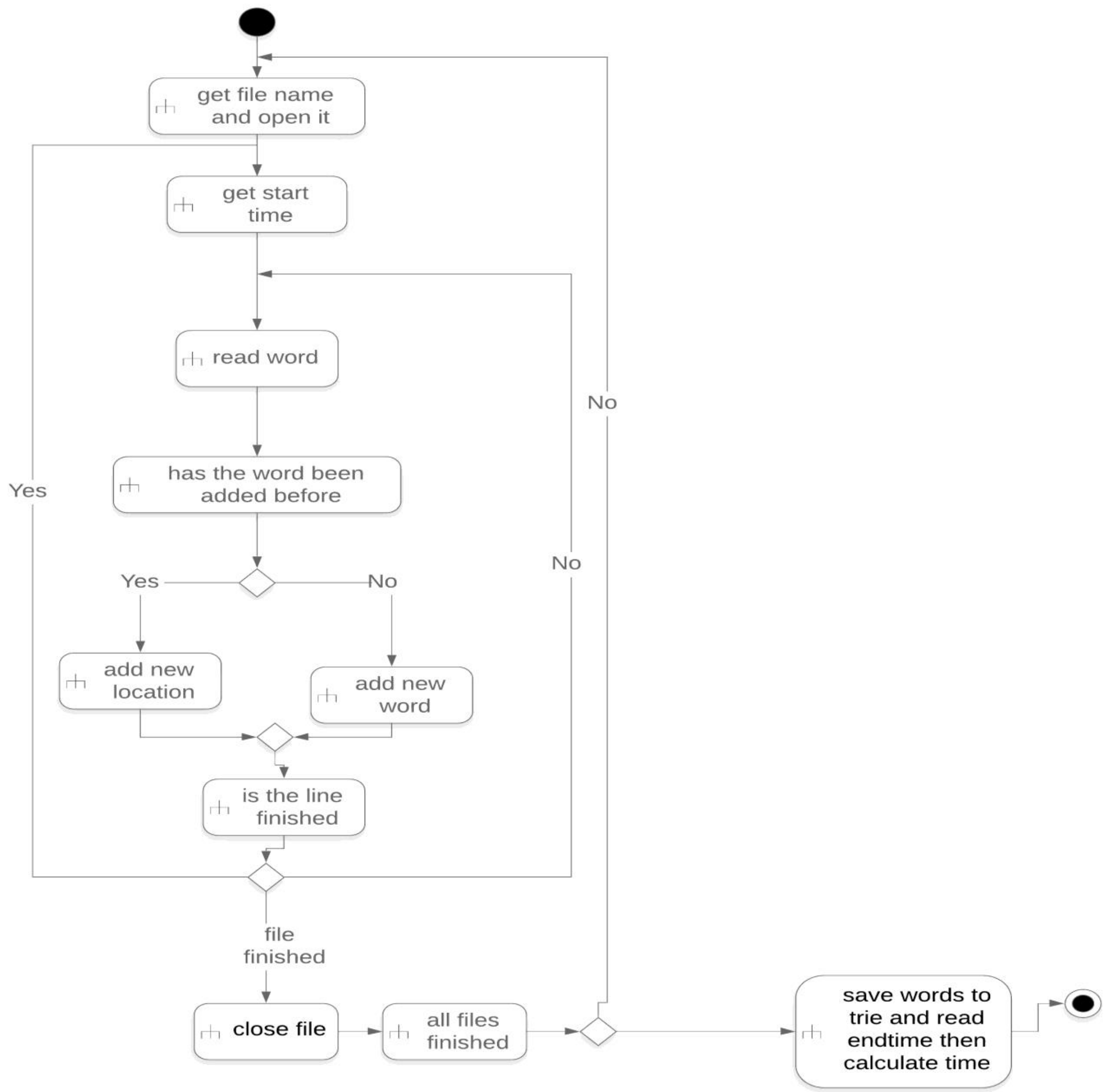


- Secondary Section

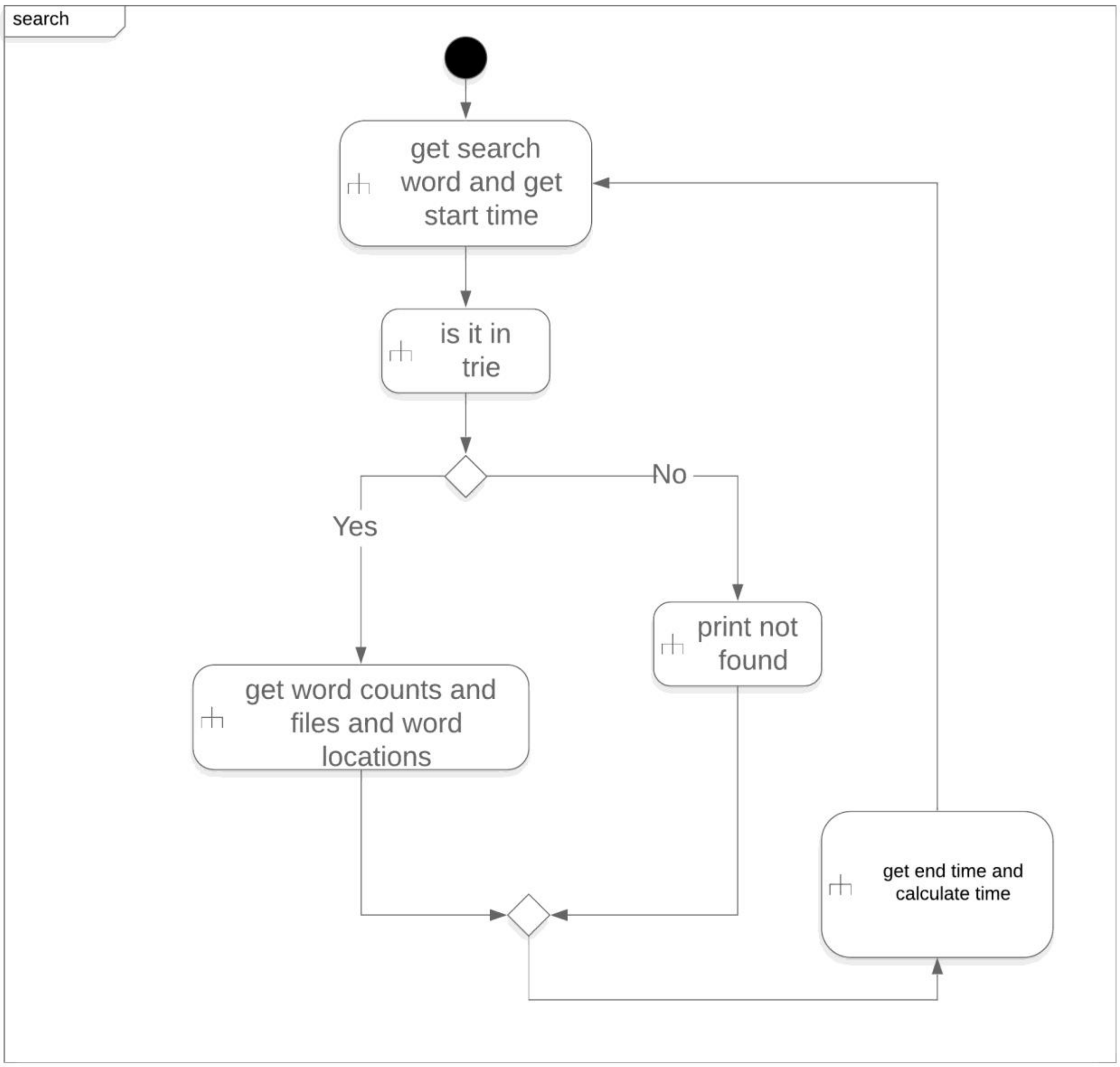


That's the UML activity Diagram of our project to explain the flow of program

- Building Process

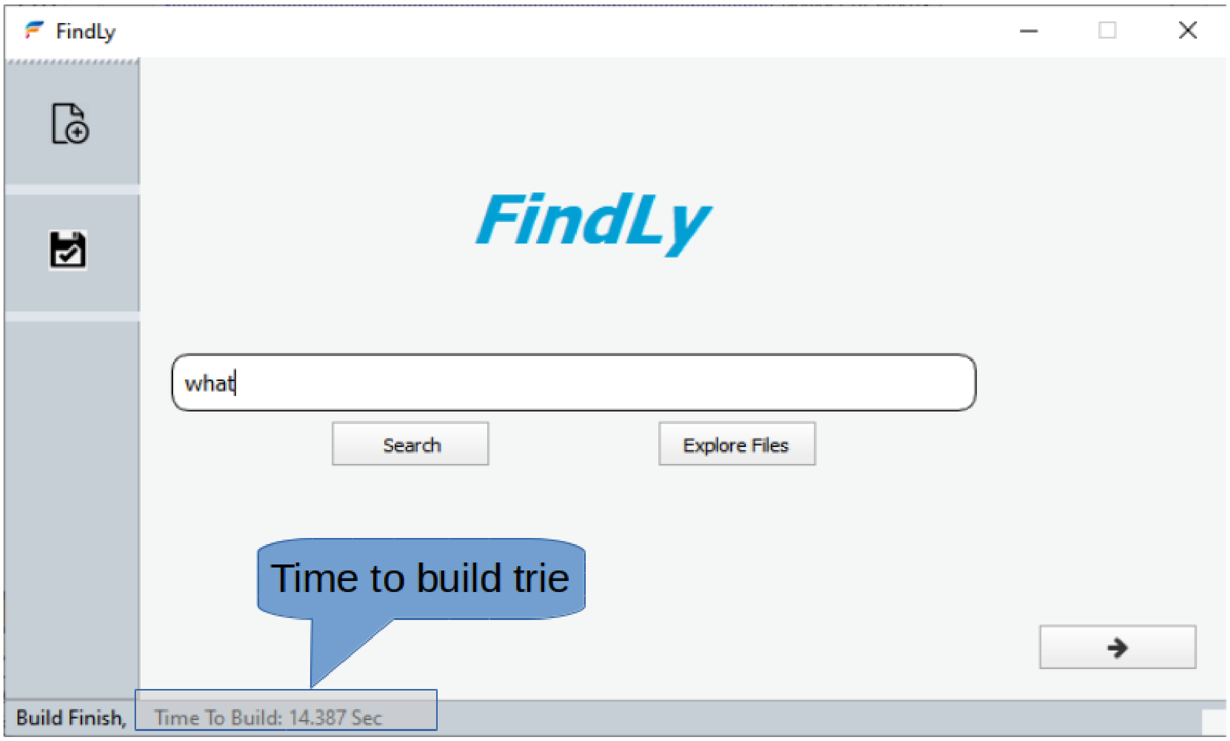


- Searching Process

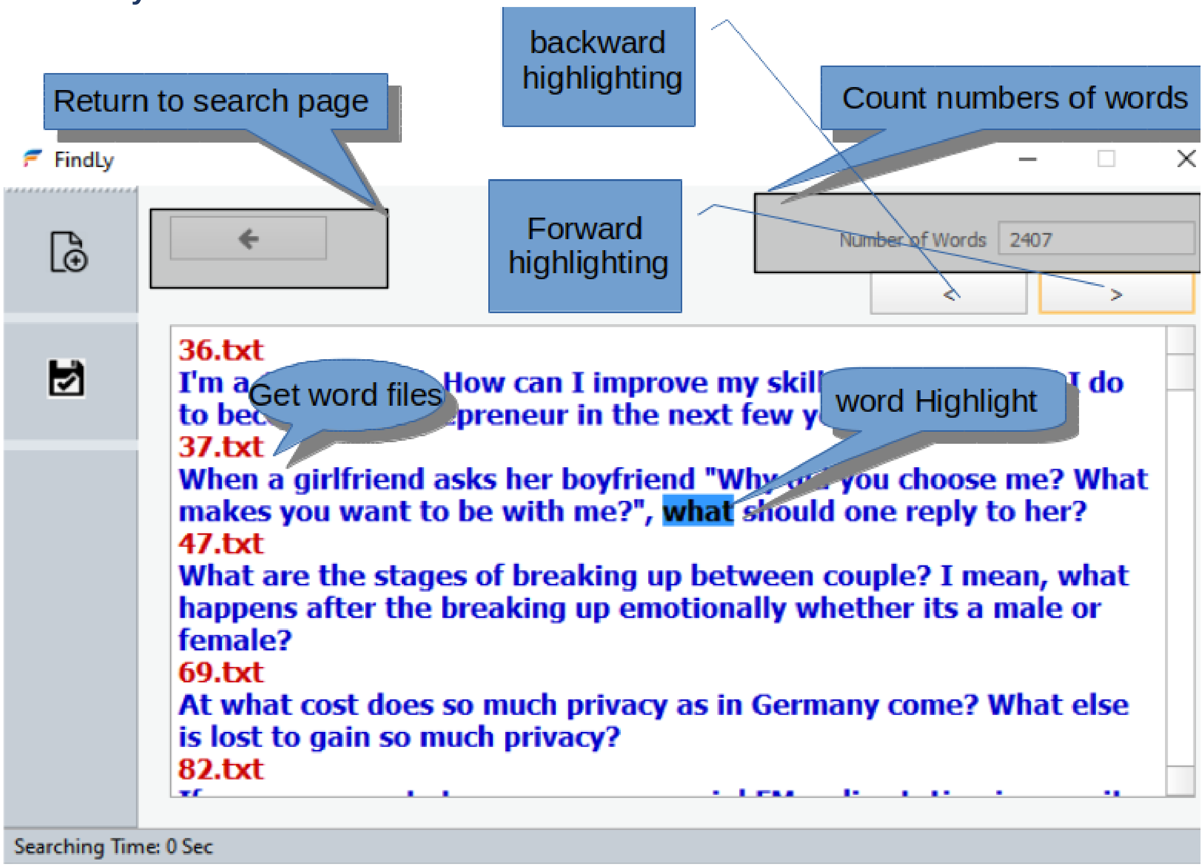


Finally explain of our UML activity diagram in real example from program

- Main Section



- Secondary Section

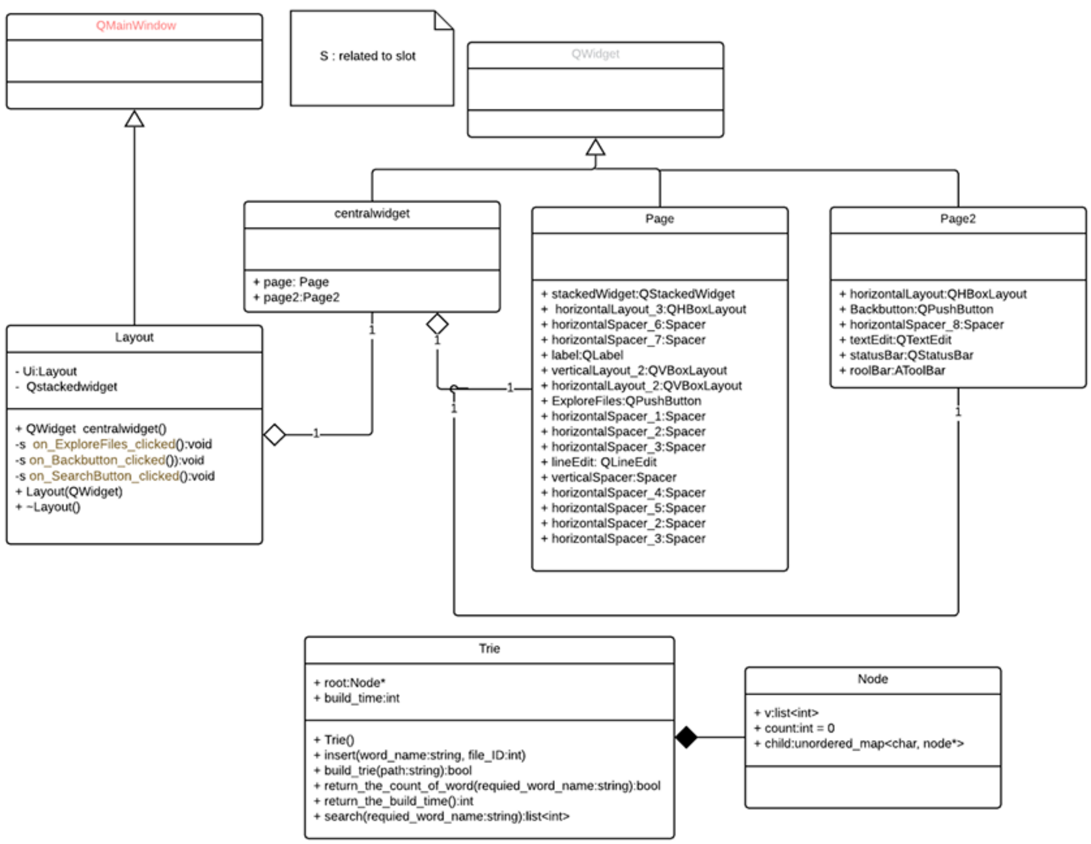


02

Second Topic

Implementation Details

First we introduce the UML Class Diagram of whole project



Explaining Diagram in Front and Backend Code.

- **Backend section**

  The implementation backend code was build using C++ programming language.

The backend consists of Trie tree class which store the words in the text file letter by letter , and the complexity of the search in the Trie as O(length of word), and structure represent the node of the Trie.

- The node structure

```
struct node
{

    list<int> file_id;
    int count = 0;
    unordered_map<char, node*> child;


};
```

The node structure contains:

1- **Child: this is unordered map point to the next item in the Tree.**
2- **Count (integer type): this valuable store the count of the character in the Trie.**
3- **List of file id: this list store the file IDs which contains this word.**

The Trie class contains several functions:

1) **Insertion function**

```
void trie::insert(string word_name, int file_ID)
{

    if (root == nullptr)
        root = getNewTrieNode();//create the root node

    node* temp = root;
    for (int i = 0; i < word_name.length(); i++) {
        char x = word_name[i];


        if (temp->child.find(x) == temp->child.end())
            temp->child[x] = getNewTrieNode();
        temp->child[x]->count++;
        temp->child[x]->file_id.push_back(file_ID);// push the file id in the node list
        temp = temp->child[x];
    }


}
```

The insertion function receive two parameters:

1- **The first parameter is the word name which we want to store it in the tree.**

2- **The second parameter is the file ID which include this word.**

The insertion function build the tree and store file IDs in the last character in the word node.

2) **Build Trie function**

```
bool trie::build_trie(string path)
{
    clock_t starting_build_time = clock();
    for (int i = 0; i < 100001; i++) {
        string name = path+ to_string(i) + ".txt";
        ifstream file(name, ifstream::in);
        for (string word; file >> word; )
            insert(word, i);
    }
    clock_t end_build_time = clock();
    build_time = (end_build_time - starting_build_time) / CLOCKS_PER_SEC;
    return true;
}
```

The build Trie function receive one parameters:

1- The path of the folder which include the files.

The Trie function read 100000 file and send the words in the files to the insertion function to store the words in the Trie.

3) **The build time function**

```
int trie::return_the_build_time()
{
    return build_time;
}
```

The build time function returns the time of reading 100000 text file.

It is in the range of seconds.

4) **Search function**

```
list<int> trie::search(string requied_word_name)
{
    node* temp = root;
    list<int>empty_list{};
    for (int i = 0; i < requied_word_name.length(); i++) {


        temp = temp->child[requied_word_name[i]];
        if (temp == NULL) return empty_list;

    }
    temp->v.unique();
    return temp->v;

}
```

The search function receive one parameters:

1- **The required word to search**

The search function return list of integer of files IDs which contains this word.

If the word does not exist, the function returns empty list.

5) **The count function**

```
int trie::return_the_count_of_word(string requied_word_name)
{
    node* temp = root;

    for (int i = 0; i < requied_word_name.length(); i++) {


        temp = temp->child[requied_word_name[i]];
        if (temp == NULL) return 0;

    }

    return temp->count;
}
```

The count function receive one parameters:

1- The required word to search

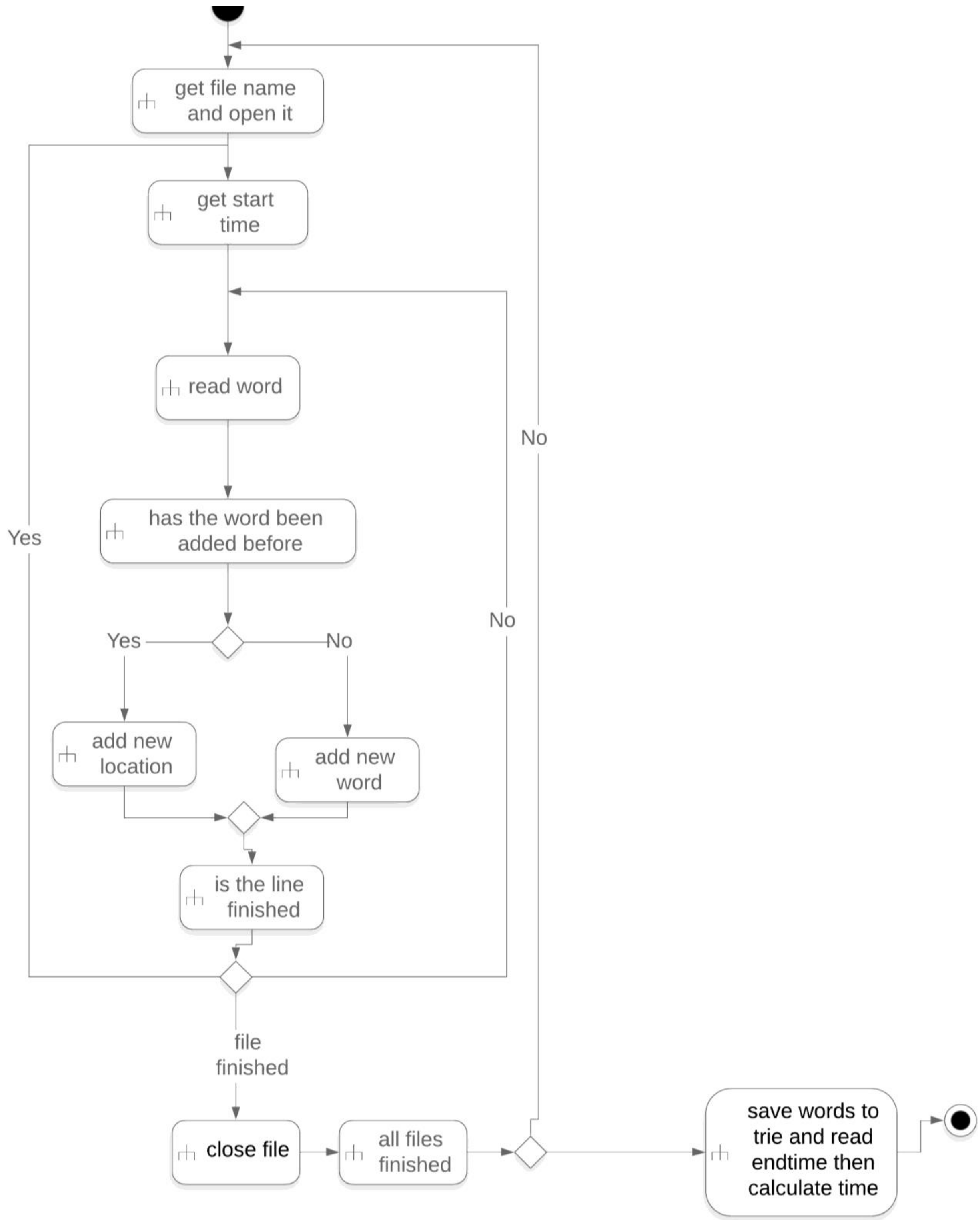
The count function return the Repetition of this word.

If the word does not exist, the function returns zero.

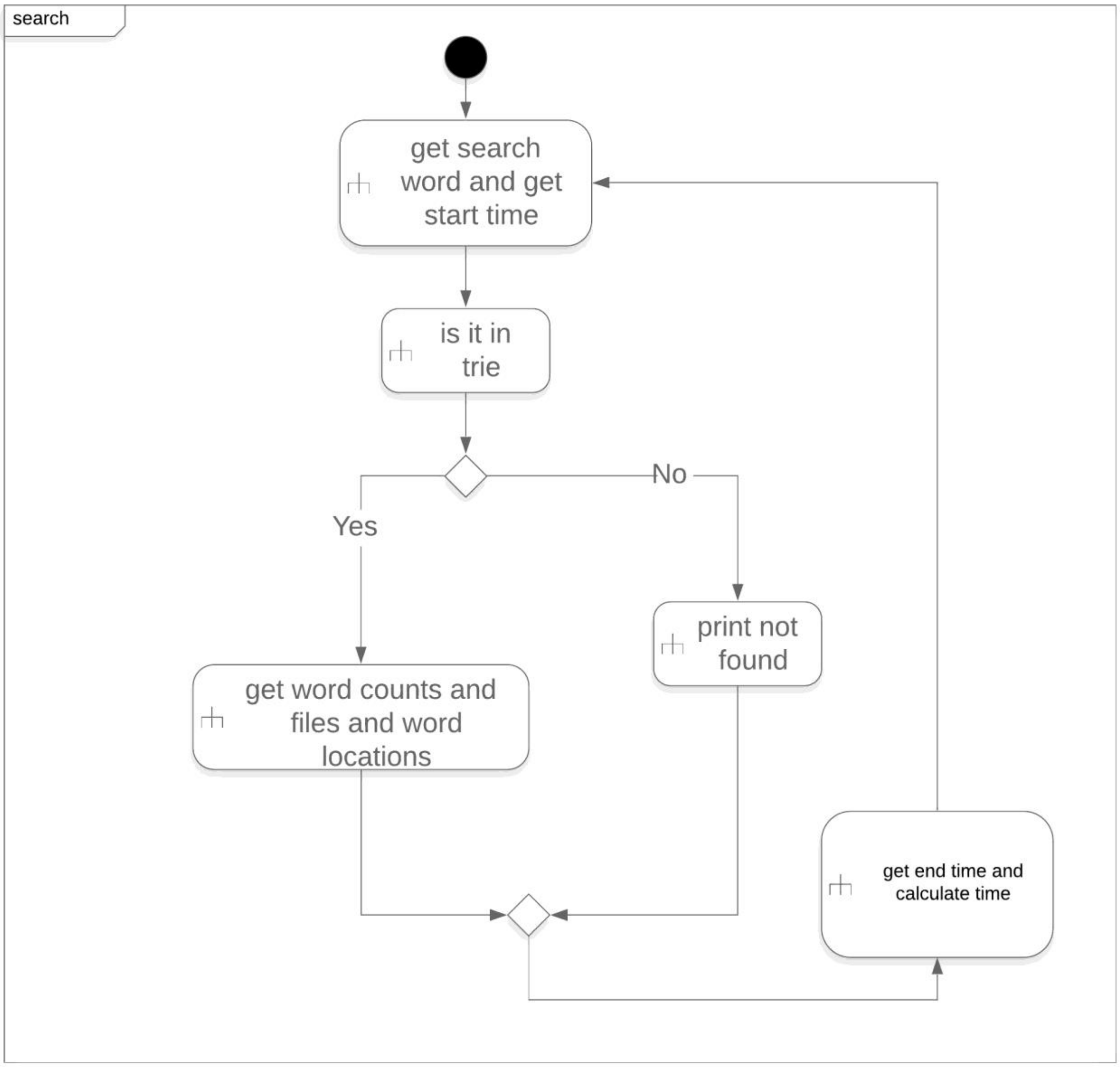
• The frontend section

- Program flow

Building



Searching


search
get search word and get start time
is it in trie
No
Yes
print not found
get word counts and files and word locations
get end time and calculate time

03

Complexity of Operations

Third Topic

Notation we used

Word length	W
number of files	F
number of words	N

Trie class

operation	**complexity**
When we start program when called function (build_trie) then a Build Trie Button hide.	O(N*W*F)
Search for words in Trie.	O(W)
Insert element into Trie when clicked on build Trie button on time.	O(W)
return_the_count_of_word	O(W)
Getbuildtime returning Building time.	O (1)
Getsearchtime returning Searching time.	O (1)

Layout class

operation	complexity
When Complete search searching we called a (GetFiles) Function.	O(F)

Project Memorization

1- GitHub

Link:

https://github.com/Mostafa ashraf19/SearchEngine?fbclid=IwAR1AKNhi9fyA3RVm9ouRlqGAeMQ-LUVhNoGL3fgr2vUEjlrUEQZkHR935WU

2- YouTube video

Link:
https://www.youtube.com/watch?v=vwEmJUe6WSY&feature=youtu.be&fbclid=IwAR2uFfqpzE7hEk-lbaj0zPsg_ot3RCTWgX4OoPfVwkMLRfxrXdkR8nh8k6o

Reference

1- Un ordered Map, CPP Reference Site

2- Inverted Index vs. Forward Index, Tutorials point Site

3- Trie Data Structure