Fuat Yiğit Koçyiğit – 16429085948
Barış Sinaplı - 38039235216
CMPE343 Section 2 & 1
09.01.2022

CMPE343 Programming Homework 5 REPORT

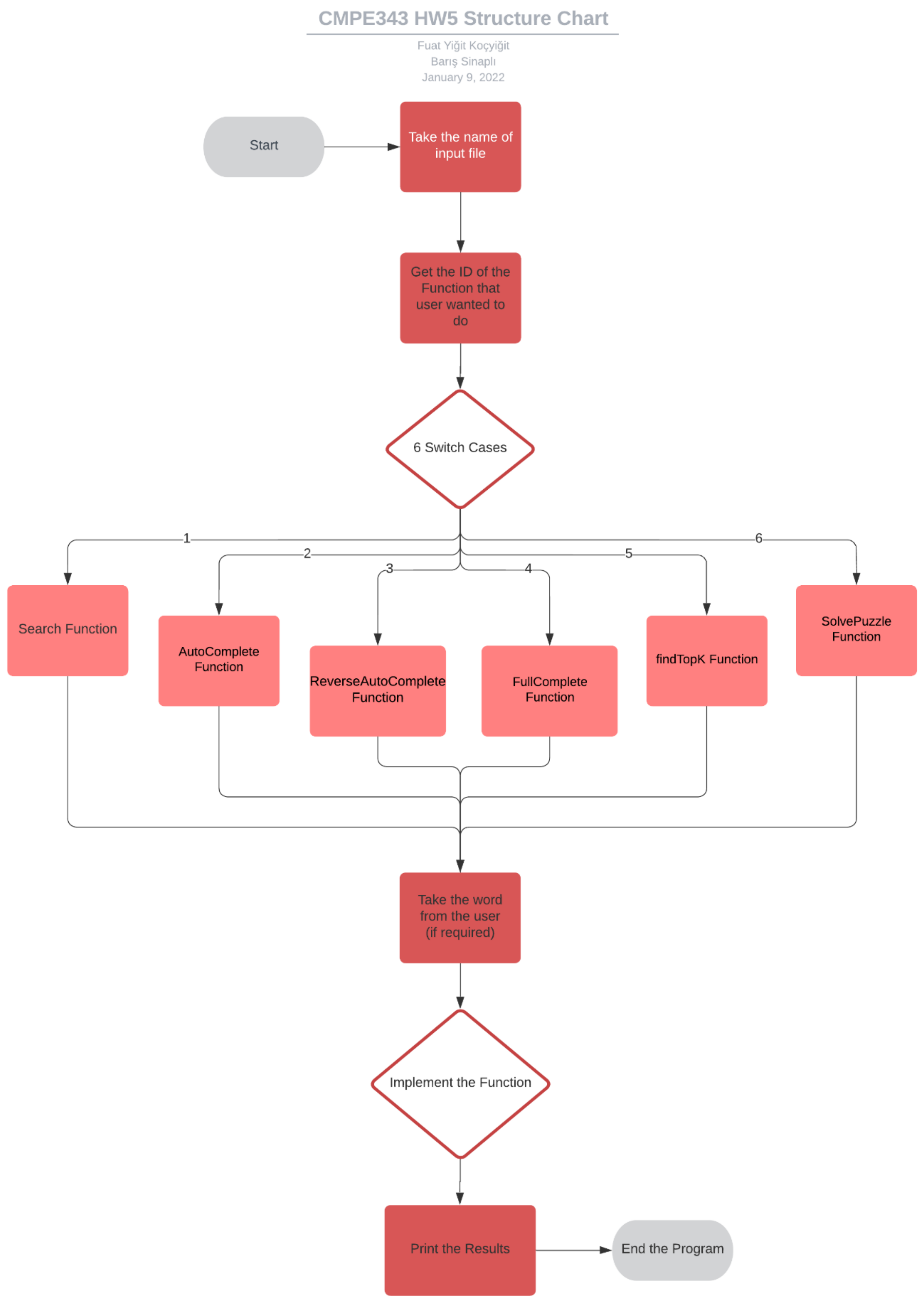
Problem Statement and Code Design

In this assignment, it is expected to implement a trie data structure and 6 different functions. These 6 tasks are Search, autoComplete, reverseAutoComplete, FullAutoComplete, findTopK and SolvePuzzle. The instructions that the functions required to do is:

- Boolean Search(String arg): This function should return true if the given argument is in our trie.
- Void autoComplete(String prefix): This function should print all strings start with given prefix in the trie, lexicographically.
- Void reverseAutoComplete(String suffix): This function should print all strings end with given suffix in our trie, lexicographically.
- Void FullAutoComplete(String prefix, String suffix): This function should print all strings start with given prefix and end with given suffix in our trie, lexicographically.
- Void findTopK(int k): This function should print top k words that have most occurrences, lexicographically.
- Void SolvePuzzle(String filepath): This function should read input from the given filepath and print all possible words in our trie.

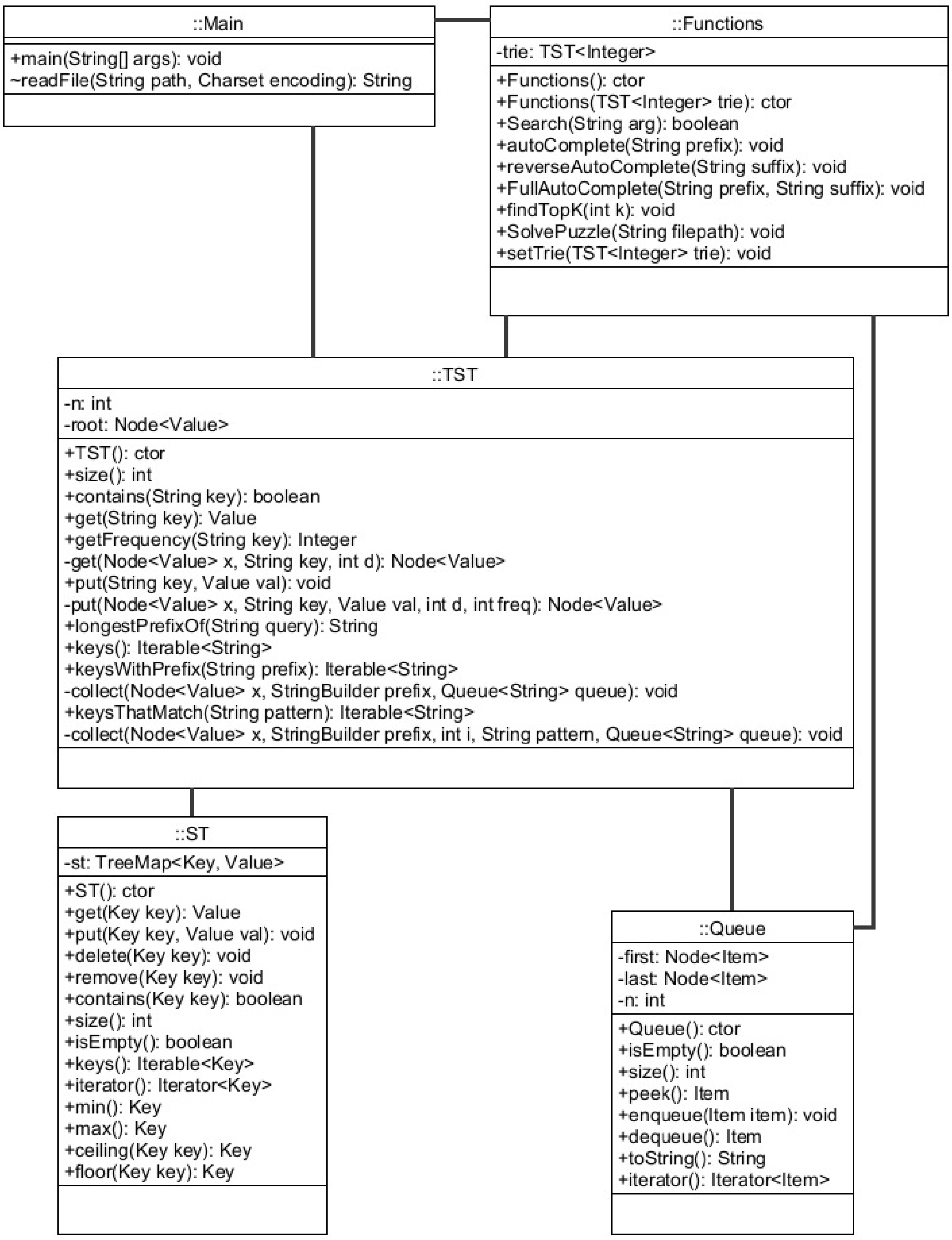
The program firstly will create a trie structure and then will take 3 inputs from the user. First one is the file name, second one is the function id and last one is given word. After that, our program will insert all the words in the given input file to our trie data structure and execute the function that is given as input.

To create the program, we used Trie Data Structure to put insert all the words in the given input to our program. The Trie Data Structure have been implemented with the TST and ST classes. Then, in the main class and Functions class, the input has been taken from the input file and required functions have been implemented which are in our Function class.



Structure Chart of the Project

(If it is not visible, there is a pdf version of it in our rar file too)


::Main
+main(String[] args): void
~readFile(String path, Charset encoding): String
::Functions
-trie: TST<Integer>
+Functions(): ctor
+Functions(TST<Integer> trie): ctor
+Search(String arg): boolean
+autoComplete(String prefix): void
+reverseAutoComplete(String suffix): void
+FullAutoComplete(String prefix, String suffix): void
+findTopK(int k): void
+SolvePuzzle(String filepath): void
+setTrie(TST<Integer> trie): void
::TST
-n: int
-root: Node<Value>
+TST(): ctor
+size(): int
+contains(String key): boolean
+get(String key): Value
+getFrequency(String key): Integer
-get(Node<Value> x, String key, int d): Node<Value>
+put(String key, Value val): void
-put(Node<Value> x, String key, Value val, int d, int freq): Node<Value>
+longestPrefixOf(String query): String
+keys(): Iterable<String>
+keysWithPrefix(String prefix): Iterable<String>
-collect(Node<Value> x, StringBuilder prefix, Queue<String> queue): void
+keysThatMatch(String pattern): Iterable<String>
-collect(Node<Value> x, StringBuilder prefix, int i, String pattern, Queue<String> queue): void
::ST
-st: TreeMap<Key, Value>
+ST(): ctor
+get(Key key): Value
+put(Key key, Value val): void
+delete(Key key): void
+remove(Key key): void
+contains(Key key): boolean
+size(): int
+isEmpty(): boolean
+keys(): Iterable<Key>
+iterator(): Iterator<Key>
+min(): Key
+max(): Key
+ceiling(Key key): Key
+floor(Key key): Key
::Queue
-first: Node<Item>
-last: Node<Item>
-n: int
+Queue(): ctor
+isEmpty(): boolean
+size(): int
+peek(): Item
+enqueue(Item item): void
+dequeue(): Item
+toString(): String
+iterator(): Iterator<Item>

UML Diagram of the Project

(If it is not visible, there is a pdf version of it in my rar file too)

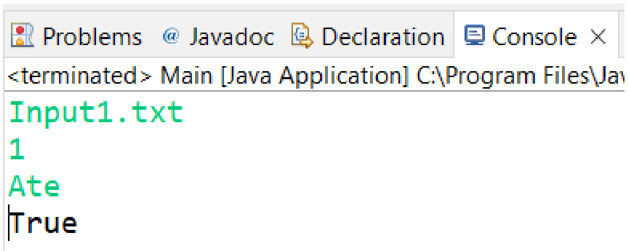
Implementation, Functionality and Performance Comparison

In the code, there is 6 different classes. 1 main class and 4 implementation classes. We implemented the Trie Data Structure with the TST and ST classes. The implementation classes have methods such as put, get, contains keysWithPrefix and longestPrefixOf. **put** methods are used for inserting a key-value pair into the symbol table. **get** method is used to get the value associated for the given key. **contains** method checks if the symbol table is containing the given key. **keysWithPrefix** method returns all keys in the set that is starting with the given prefix. And the **longestPrefixOf** method returns the string in the symbol table that is the longest prefix of the query.

Then, we created the main and Functions class. In main class, we are creating our arrays, TST, and Function object. Then, with switch case we are implementing the required function that the input requested (required functions are in Functions class). In Functions class we have the functions that the program needs to do. There are 6 methods in this class which are Search, autoComplete, reverseAutoComplete, FullAutoComplete, findTopK and SolvePuzzle. **Search** method is for checking if the given argument is in our trie. **autoComplete** method is for printing all strings that starts with given prefix in the trie, lexicographically. **reverseAutoComplete** method is for printing all strings that end with given suffix in our trie, lexicographically. **FullAutoComplete** method is to print all strings that start with given prefix and end with given suffix in our trie, lexicographically. **findTopK** method prints top k words that have most occurrences, lexicographically. And the **SolvePuzzle** function is for reading input from the given filepath and printing all possible words in our trie.

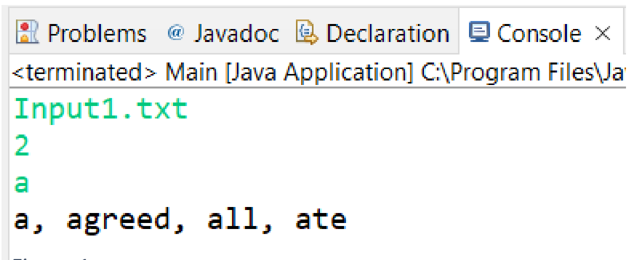
Testing

To test our codes, we used the sample inputs that are given to us. The interface of our program and the outputs was like:



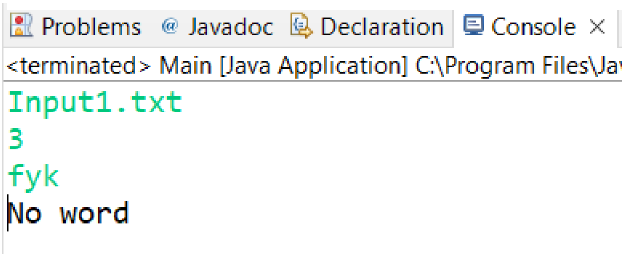
```
<terminated> Main [Java Application] C:\Program Files\Ja
Input1.txt
1
Ate
True
```

Figure 2



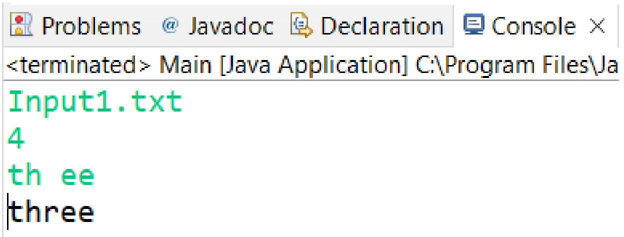
```
<terminated> Main [Java Application] C:\Program Files\Ja
Input1.txt
2
a
a, agreed, all, ate
```

Figure 1



```
<terminated> Main [Java Application] C:\Program Files\Ja
Input1.txt
3
fyk
No word
```

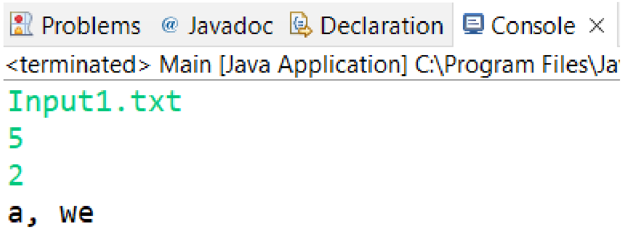
Figure 3



```
<terminated> Main [Java Application] C:\Program Files\Ja
Input1.txt
4
th ee
three
```

Figure 4

Since there isn’t a performance comparison, we did not measured the runtimes.



```
<terminated> Main [Java Application] C:\Program Files\Ja
Input1.txt
5
2
a, we
```

Figure 5

Final Assessments

- The trouble points that we experienced in this assignment was implementing the trie data structure and creating the functions.
- The implementation/modifying the data structure and reading/inserting every word correctly were the most challenging parts for us because it was hard to create trie data structure and use it in functions.
- We learned how to use the trie data structure and how it can be very used in our daily lives.