CMPE343 HW -4
Burak SAĞLAM : 13760307838 Section 2
Sarp ARSLAN :11458145526 - Section03

HOMEWORK REPORT of QUESTION 1

Problem Statement and Code Design

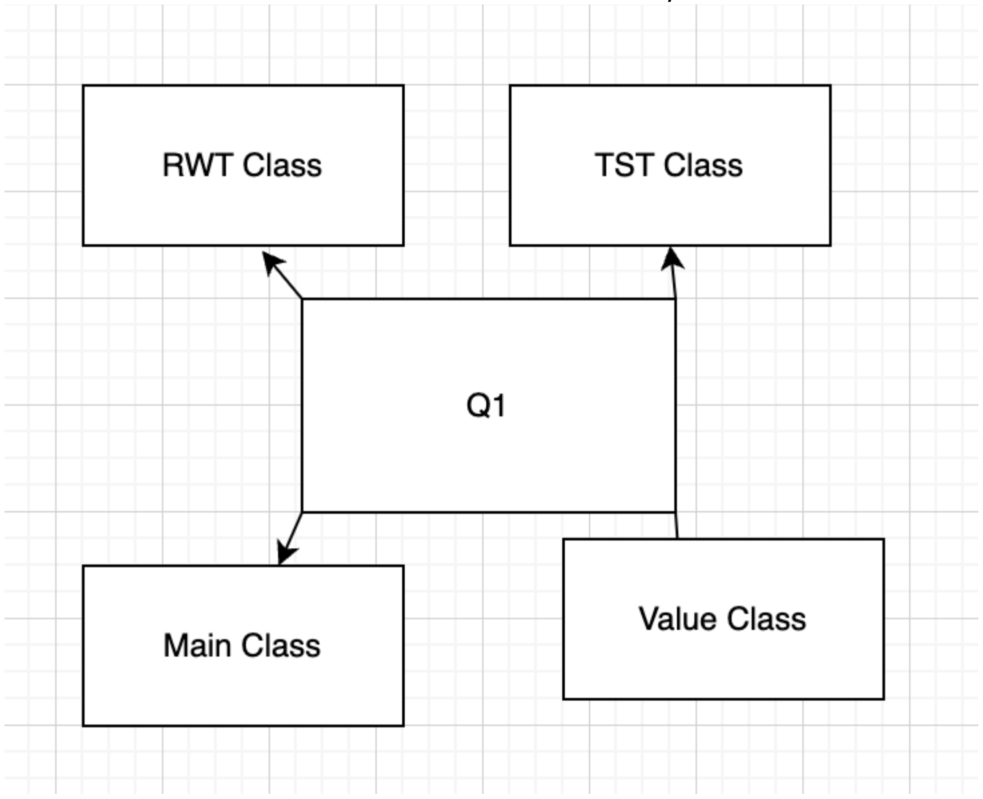
In this report, we are going to implent trie structure. We are getting information from the user and inserting. These are the functions:

bool search (String arg) determines whether the supplied variable is present in Trie and returns true or false.

void numberPrefix(Trie trie) retrieves every word in Trie and determines whether a string is prefixed with another string. For each knee, there are no prints visible.

void reverseFind(String suffix): This function prints all strings in Trie that finish with the specified suffix lexicographically.

These sub-module shows to structure that used in the system

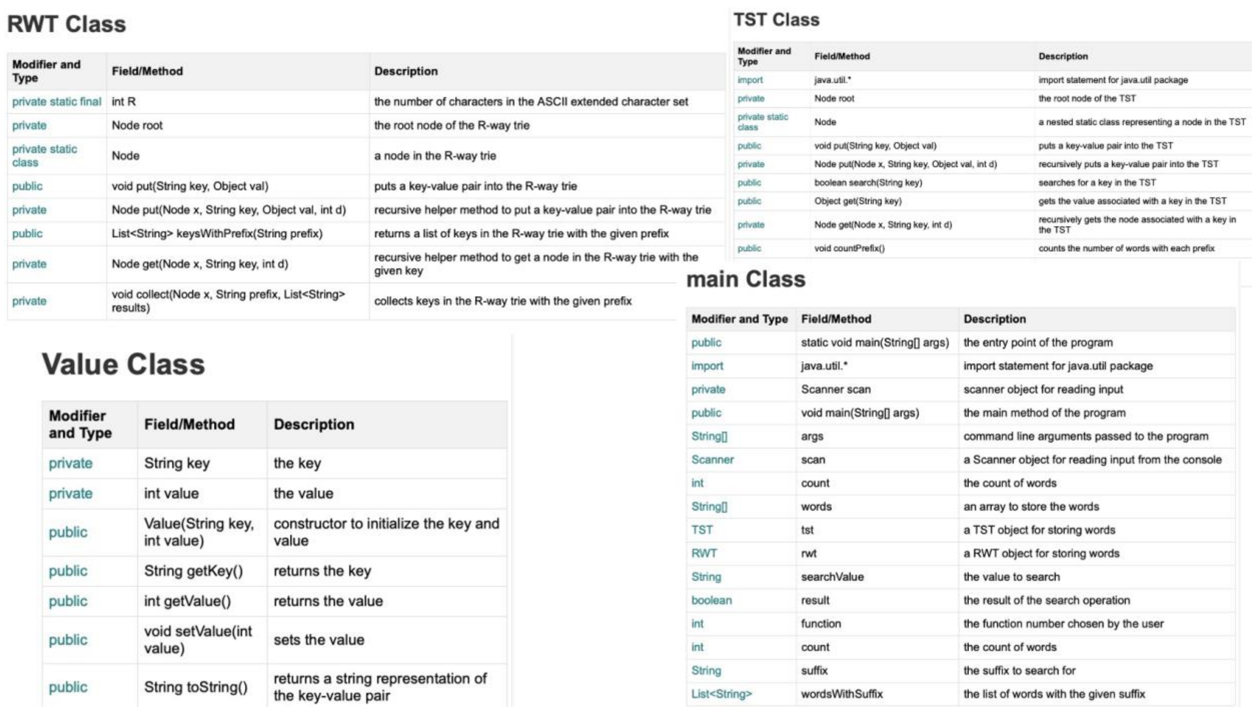


Implementation, Functionality

This code design allows you to implement the Trie data structure and the desired functions in Java. The TrieNode class creates a Trie data structure in which each node has characters and links to other nodes.

The search function checks whether the specified word is found in Dec. The countPrefix function checks whether other words appear in front of the words in the Trie and prints the number they appear.

Using this design, we can create a Trie data structure, implement the desired functions, and solve Trie-related problems.
The final function will lexicographically output all strings in your Trie that end with the specified suffix. Consider adopting a multi-trie solution for this function, or look at more sophisticated data structures such suffix arrays.
As the implementation is showed in the previous part. The application has 4 main sub – modules.



RWT Class

Modifier and Type	Field/Method	Description
private static final	int R	the number of characters in the ASCII extended character set
private	Node root	the root node of the R-way trie
private static class	Node	a node in the R-way trie
public	void put(String key, Object val)	puts a key-value pair into the R-way trie
private	Node put(Node x, String key, Object val, int d)	recursive helper method to put a key-value pair into the R-way trie
public	List<String> keysWithPrefix(String prefix)	returns a list of keys in the R-way trie with the given prefix
private	Node get(Node x, String key, int d)	recursive helper method to get a node in the R-way trie with the given key
private	void collect(Node x, String prefix, List<String> results)	collects keys in the R-way trie with the given prefix

TST Class

Modifier and Type	Field/Method	Description
import	java.util.*	import statement for java.util package
private	Node root	the root node of the TST
private static class	Node	a nested static class representing a node in the TST
public	void put(String key, Object val)	puts a key-value pair into the TST
private	Node put(Node x, String key, Object val, int d)	recursively puts a key-value pair into the TST
public	boolean search(String key)	searches for a key in the TST
public	Object get(String key)	gets the value associated with a key in the TST
private	Node get(Node x, String key, int d)	recursively gets the node associated with a key in the TST
public	void countPrefix()	counts the number of words with each prefix

main Class

Modifier and Type	Field/Method	Description
public	static void main(String[] args)	the entry point of the program
import	java.util.*	import statement for java.util package
private	Scanner scan	scanner object for reading input
public	void main(String[] args)	the main method of the program
String[]	args	command line arguments passed to the program
Scanner	scan	a Scanner object for reading input from the console
int	count	the count of words
String[]	words	an array to store the words
TST	tst	a TST object for storing words
RWT	rwt	a RWT object for storing words
String	searchValue	the value to search
boolean	result	the result of the search operation
int	function	the function number chosen by the user
int	count	the count of words
String	suffix	the suffix to search for
List<String>	wordsWithSuffix	the list of words with the given suffix

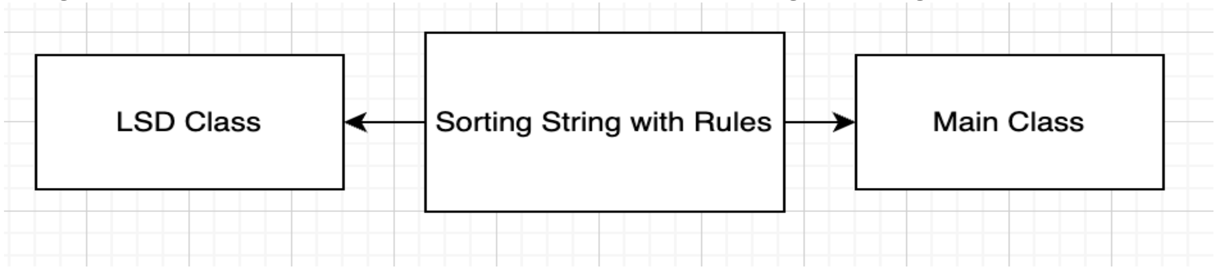
Value Class

Modifier and Type	Field/Method	Description
private	String key	the key
private	int value	the value
public	Value(String key, int value)	constructor to initialize the key and value
public	String getKey()	returns the key
public	int getValue()	returns the value
public	void setValue(int value)	sets the value
public	String toString()	returns a string representation of the key-value pair

HOMEWORK REPORT of QUESTION 2

Problem Statement and Code Design

In this report, we are going to implement String Sort Algorithm . We are going to implement this by using graph structure. In this algorith we have rules and we implemented this algorithm in accordance with these rules. We have 2 class the first one is LSD class . In that class we are sorting the string and the other class is main class. In main class we are sorting two strings with even distance.

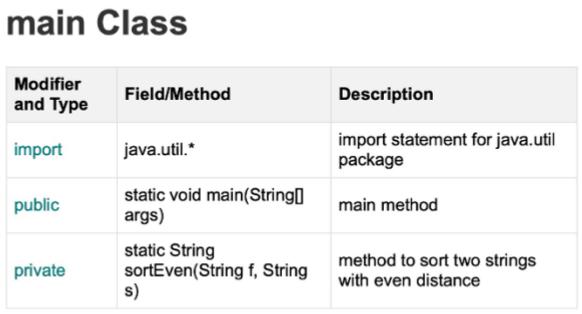


Implementation, Functionality

To explain the functionality of this program we need to explain how this code works.

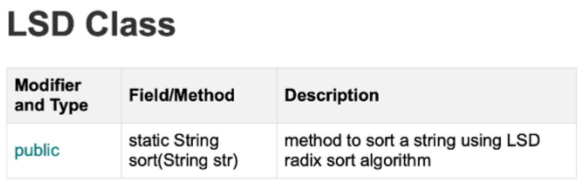
First, we pad the strings to make them the same length. Next, we group by the rightmost characters. we sort by character within the groups (counting sort). This sorting process is repeated to the left. As a result, the strings are sorted in the desired order. Strings are sorted using the LSD radix sort algorithm. After that , we have sortEven method which is used to sort two strings with even distances. After these, we get the true result.

As the implementation is showed in the previous part. The application has 2 main sub – modules.



main Class

Modifier and Type	Field/Method	Description
import	java.util.*	import statement for java.util package
public	static void main(String[] args)	main method
private	static String sortEven(String f, String s)	method to sort two strings with even distance



LSD Class

Modifier and Type	Field/Method	Description
public	static String sort(String str)	method to sort a string using LSD radix sort algorithm

TESTING FOR ALL QUESTIONS

In the first question, we encountered certain errors while implementing the countPrefix method, and we solved these errors by researching and getting help from the textbook. In the second question, we saw that the program suppresses riccrisis instead of riccss in a test case, we used the Collections library to solve this problem. We also encountered a NumberFormatException error while testing. In some tests, although the distance is not very large, we got an error for the number 13151411525 in one test. We solved the problem by changing int primitive data type to long primitive data type. In this way, we have understood the importance of always making the code design for the worst possible scenario, regardless of the input given.