Information:

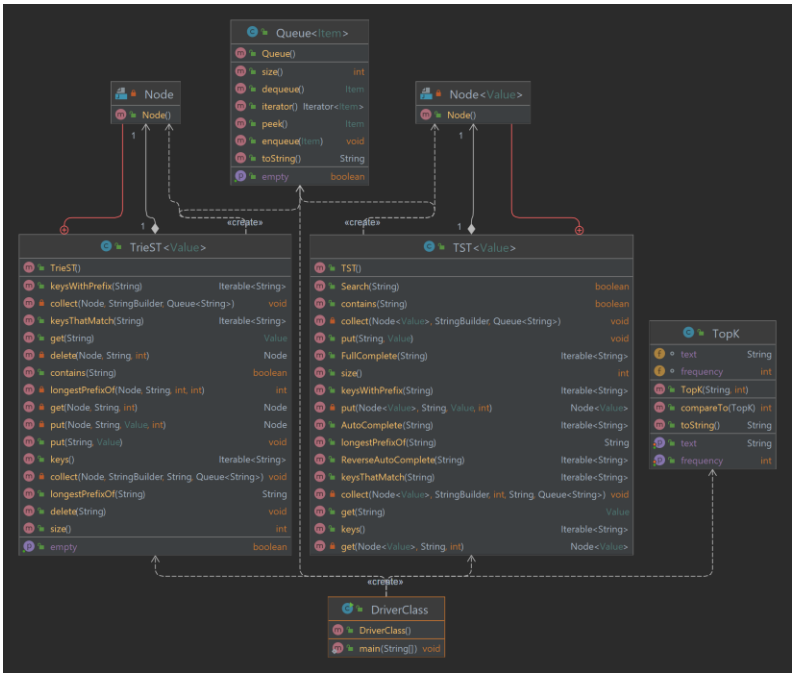
CMPE343/224 Programming Homework5 Tries;

Problem Statement and Code Design:

Açıklamalı [AK1]:

Part1: **In this assignment we need to o implement Trie Data Structure and belonging functions by using Symbol Table and Tree implementations of the Trie Data Structure. We need to establish the fundamental operation of a Trie which are "Searching,Auto-Complete-Reverse Auto-Complete,Full Auto-Complete and after that we add some additional facilities like FindTopK and PuzzleSolving by using HashMaps and multi-trie and 8-way 2D array search via dynamic programming. We design the programme which gets a operation number from user and than get operation parameter such as "prefix,suffix,filepath etc.".Eventually do corresponding job about it.**

Here is the General Top-down UML structure of our assignment:



Implementation, Functionality:

In this assignment have some sub-modules to complete this task.

Queue Class: Ordinary Queue implementation in order to use it "FIFO" (First In-First Out) manner.It has enqueue(Item item), Item dequeue() and boolean isEmpty() which we need for determining Sequence of the Tasks.

TopK Class: This class represents the TopK object and it's attributes which are "text" which holds string itself and "Frequency" which holds frequency of given string". We need this class for "storing each string's frequency" in a one instance. It has String toString() and Getter Setter methods for it's attributes.

TrieST Class: The one of the Fundamental aspects for this task is "Trie Data Structure". This class used to store the" String stream" in an efficient way which is Linear Time (for most of the operations). An R-way Trie Class which every node have alphabet lenght Node next() array to indicate the childs of them (with severel null ones) in tree. It has

- Value get(String key) (Returns the value associated with the given key.)
- Put (String key,Value val) Inserts the key-value pair into the symbol table overwriting the old value with the new value if the key is already in the symbol table)
- int size() (Returns the number of key-value pairs in this symbol table.) Iterable<String> keys() (Returns all keys in the symbol table)
- Iterable<String> keysWithPrefix(String prefix) (Returns all of the keys in the set that start with given prefix)
- Iterable<String> keysThatMatch(String pattern) Returns all of the keys in the symbol table that match given pattern where the character '.' is interpreted as a wildcard character.
- String longestPrefixOf(String query) (Returns the length of the longest string key in the subtrie rooted at x that is a prefix of the query string, assuming the first d character match and we have already found a prefix match of given length *(-1 if no such match)*).

TST Class: The one of the Fundamental aspects for this task is "Trie Data Structure". This class used to store the" String stream" with only examined just enogh key characters (for most of the operations). An 3-way Trie Class which have only 3 child for each node in tree.They are storing less equal and greater nodes of the parent node.It has

- Value Search(String key) (Returns the value associated with the given key.)
- Put (String key,Value val) Inserts the key-value pair into the symbol table overwriting the old value with the new value if the key is already in the symbol table)
- Iterable<String> ReverseAutoComplete(String prefix) (Returns all of the keys in the set that start with given suffix)
- delete(String key) (Removes the key from the set if the key is present)
- Iterable<String> FullComplete(String pattern) (Returns all of the keys in the set that start and end with given pattern with considering both prefix and suffix.

- **int size() (Returns the number of key-value pairs in this symbol table.)**
  Iterable<String> keys() (Returns all keys in the symbol table)
- **Iterable<String> AutoComplete(String prefix) (Returns all of the keys in the set that start with given prefix)**
- **String longestPrefixOf(String query) (Returns the length of the longest string key in the subtrie rooted at x that is a prefix of the query string, assuming the first d character match and we have already found a prefix match of given length *(-1 if no such match)*).**

PseudoCodes for Trie Algorithms

PseudoCodes for R-Way Trie Algorithm:

1) put (Node node, String key, Value value) // PUT METHOD

```
"""Insert key/value pair into node."""
  for char in key:
     if char not in node.children:
        node.children[char] = Node()
     node = node.children[char]
  node.value = value
```

2) search (Node<Value> x, String key, int d) // SEARCH METHOD

```
if x is null return null;
char c= key.charAt(d);
if processing char is less than root node(which is x)
     search(x.left)
Else if(processing char is greater than root node(which is x)
     search(x.right)
Else if( d is in the key length range)
     Search(x.mid)
Else return x
```

3) AutoComplete-keyswithprefix-

```
Take input string and add to queue.
Call TrieAutoComplete Method and send the created queue.
  Statement =Trie.AutoComplete(CreatedQueue)
Then build an string with the output of the codes.
```

BuildedString =””;
For string in statement
BuildedString = Builded +”, ”+string
Then print the builded string.
System.out.println(BuildedString);

4) ReverseAutoComplete
private String reverseString(String str) {
create a string builder object and call its reverseString Function.
StringBuilder sb = new StringBuilder(str);
sb.reverse();
Then return Reversed String
return reversedString;

ReverseStringSearch(Tries tries,String strt)

Reverse every string in the trie by calling reverseString function and add it to a new trie.

For every string in tries:

String =reverseString(string)

buildedtrie.add(string)

search for the string that starts with ‘strt’ , reverse it and add to a new trie.

For every string in buildedtrie

If string start with ‘strt’{

String = reverseString(string)

Newbuildedtrie.add(String)

Else {

Pass to next string; } }

Print every element in the last created trie.

For String in NewbuildedTrie{

System.Out.Print(String)

}

5) **FindTopK Pseudo Code:**

```
Read required top elements in the trie.
Scanner scan = nextInt ;
Create a Hashmap that maps strings with their occurrences.
Hashmap stringmap = new hashmap(String ,int);
Search for every string in the trie.
For string in the trie{
If Stringmap includes string {
        Stringmap<String,int>.increase int by 1;
        }
Else    {
        Stringmap.add<string,1>
        }
}
Sort the stringmap according to their integers.
Stringmap.sortbyint();
Print as much as the user want.
For 0 to scan {
System.out.print(stringmap.pop())
}
```

6) **SolvePuzle Pseudo Code:**

```
Void solve Puzzle (Tries tries,char [][] puzzle){
```

Search for every string in the trie in the Solvedpuzzle.
For string in tries {
Call the method that solves the puzzle and search for words in the solved puzzle and returns whether the string is in the puzzle or not.
If SolvePuzzle.includes(puzzle,string) {
Print the result if string is found.
System.out.print(string)}
Else {Pass to next string;}}}

```java
int[] x = {-1, -1, -1, 0, 0, 1, 1, 1};
int[] y = {-1, 0, 1, -1, 1, -1, 0, 1};
// This function searches in all
// 8-direction from point
// (row, col) in grid[][]
boolean search2D(char[][] grid, int row, int col, String word) {
    // If first character of word
    // doesn't match with
    // given starting point in grid.
    if (grid[row][col] != word.charAt(0)) {
        return false;
    }
    int len = word.length();

    // Search word in all 8 directions
    // starting from (row, col)
    for (int dir = 0; dir < 8; dir++) {
        // Initialize starting point
        // for current direction
        int k, rd = row + x[dir], cd = col + y[dir];
        // First character is already checked,
        // match remaining characters
        for (k = 1; k < len; k++) {
            // If out of bound break
            if (rd >= R || rd < 0 || cd >= C || cd < 0) break;
            // If not matched, break
            if (grid[rd][cd] != word.charAt(k)) break;
            // Moving in particular direction
            rd += x[dir];
            cd += y[dir];
        }
        // If all character matched,
        // then value of must
        // be equal to length of word
        if (k == len) return true;
    }
    return false;
}
```

```java
public void solvePuzzle(TrieST<Integer> Try, String path) throws FileNotFoundException {
    Scanner sc = new Scanner(new File(path));
    ArrayList<String> list = new ArrayList<>();
    char[][] arr;
    int counter2 = 0;
    String next = sc.nextLine();
    list.add(next);
    while (sc.hasNext()) {
        next = sc.nextLine();
        list.add(next);
    }
    counter2 = 0;
    arr = new char[list.size()][list.size()];
    for (int i = 0; i < list.size(); i++) {
        for (int j = 0; j < list.get(i).length(); j = j + 2) {
            arr[i][counter2] = list.get(i).charAt(j);
            counter2++;
        }
        counter2 = 0;
    }
    createArray(Try, arr);
}
```

Testing:

I created tester class "TesterClass.java" which takes one input file named 'input2.txt' (Attached into .zip file) to create the trie structure. After creating the structure, it works like the actual program but takes concrete inputs to test the functions. The tester class provides test for different inputs and tests all the functionality of the program. I created another file to test the puzzle named 'puzzle2.txt' (Attached into .zip file). It tests puzzle function and

prints results as expected. At the end of testing all results are overlap with the expected inputs. If the given input format is as expected, the program does not have any bugs according to test cases.

Final Assessments:

What were the trouble points in completing this assignment?

In my point of view, one of the trouble points of this assignment is constructing the Trie itself.We need to consider how we can divide the Strings as a character that it will be stored in efficient and supportşve for Symbol table operations.

Strings into number-based vertex and forming edges between them. Also for Prim Algorithm we should consider the lexicographical order of the vertices in case of equal edge weights.

Which parts were the most challenging for you?

Most Challenging part of this assignment is searching 2D arrays dynamically and putting possible Strings into the Trie in order to compare the input1.txt and puzzle.txt. I worked on it several hours and try to create the 2D array search solution to solve given puzzle. After solving this problem our life became easy. We put the decrypted strings into another trie then compare with the given input1.txt file. After the main logic is completed. I tried to figure out how can i apply this procedure in to the Java code.

What did you like about the assignment? What did you learn from it?

The Logic of the Storing Strings in an efficient way is vital for most of the technological area and also used widely in contemporary software solutions even in our Daily life Apps such as Finder facility in "Word text editors". Because of that a developer should have clear understanding on this topic.With this assignment ı examine the fundamental requirements and implementation keystones of the R-way and 3-way Trie Implementations. I believe that,it will be helpful for me in terms of both Data Structure knowledge and algorithmic improvement.