T9

Mahad Ahmed

Fall 2022

Introduction

Do you remember the old cellphones when we used to click the buttons several times to get a specific letter we wanted when we where trying to send a message? In the beginning it was a bit difficult because you had to write each single letter to make a word. But when the T9 system became available it made life so much easier and texting so much better. For example if you typed the key sequence "43556" the text message would probably say "hello" and not something else that is not a word. The T9 system was implemented in a very clever way. For this assignment we have done our own version of the T9 system. It will surely not be as memory efficient as the original T9, but the structure is the same.

Trie

A trie is a different kind of a tree structure. The word trie in of itself is a excerpt from the word "retrieval". Trie is a sorted tree structure the stores strings or set of strings. Each node in the tree has as many branches as there are letters in the alphabet. In the Swedish language we have 29 letters but we want to have 3 letters for each key so we only use 'a' to 'ö' without 'q' and 'w'.

A node will also have a boolean value. This boolean value is like a path setter, which means if it is true then we have a word. ö

code,index and key

One of the first task that needed solving was that to implement a method whereby when we enter a character ('a'....'ö') it gives back code from 0-26. So given 'c' it will return the code "2". we implemented it using the ASCII values. So each letter we move on step to the left so that 'a' is 0 and 26 is 'ö'. A very simple method to implement.

The next method is the opposite of the previous, now instead of giving a character we give a code that then returns the character. So we basically did the opposite of the method above.

Another extra method we implemented was, given a character then return a key. This came in handy when doing some tests to see if everything was working. To be frank this was very simple to implement and only to us one line of code to do.

```
public static int getKeyFromChar(char letter) {
   return (findCharacterCode(letter) / 3) + 1;
 }
```

What we did was to divide the character code by three and adding a one after. So for example if have 'a' as a parameter then the result will be 1. Because the character code for 'a' is 0 and 0/3 is 0 and the plus 1 gives us 1. So a will go to key 1 on the phone. Another example if we have 'g' as a parameter, the character code will be 6. So 6/3 is 2 then plus 1 gives us 3. That means the letter 'g' can be accessed by pressing down key 3 on the phone.

adding words

To populate the tree we need to add some words. Which words? The top 8000 words used in the Swedish language given to us by our teacher. Firstly you will need to read in all of the words from a text-file. This was done using a similar method used in another assignment.

```
public Trie(){
   root = new Node (' ');
   try (BufferedReader bufReader = new BufferedReader(new
       FileReader("kelly.txt"))){
     String line = bufReader.readLine();
     while (line != null) {
     add(line);
     line = bufReader.readLine();
   }
   bufReader.close();
   }
   catch (Exception e) {
```

```
      e.printStackTrace();
    }
  }
```

This is how we read in a file using the "BufferedReader" which is a class object in java. We also call our add function.

So how did we go about adding a word in our tree? The teachers implementation was to do it in maybe 12 lines of code, but we did it only using 5. It works like this, we first check if the first position in our array is empty. This array is located inside of the node. If it is empty create a new node that has the an array and a boolean and put the first character of the word we want to add. Then we loop to the next character and do the same until we get to the last character. When we get to the last character we set the boolean variable *word* to true. Now we have a word inside of the tree. But know that this word is not located together like an array but more like a linked list where the on before a character references the character that comes after.

```
public void add(String word){
   Node current = root;
   for (int i = 0; i < word.length(); i++){
     if (current.arrayCharacters[findCharacterCode(word.charAt(i))]
         == null){
       current.arrayCharacters[findCharacterCode(word.charAt(i))] =
           new Node(word.charAt(i));
     }
     current =
         current.arrayCharacters[findCharacterCode(word.charAt(i))];
   }
   current.word = true;
 }
```

searching for words given a sequence

If we right a key sequence let's say "1234", then we want to see all of the possible words that matches that key sequence. This is what we implemented, a sort of a auto complete. We have a method called search that takes in a key sequence and gives back an array with all of the possible words. To implement this was a bit tricky.

We have two methods called *search*, one is i out trie object and the other one in our node class. Let's do an example. If we have the key sequence "324" we then want all the possible words that match this specific sequence. What we do is then take every key separately and do one by one. Then we check for every possible character in every key and match it with every other keys character. In this example the first letter in key 3 is 'g'. Then we do another search and see what character that works with g in the next

key which is 'd'. Then it checks does a word in our txt-file begin with the sequence "gd" which it does not. Then we go to next character in the same key which was 2. Now it becomes "ge" which is a word an other words begin with. Then you have to check for every combination. After you have checked for every combination then you return those that were a word in an array.

code for search method

```
//This method is inside our Trie object
   public String[] search(String key) {
        if (key != null && key.length() > 0)
            return root.search(key, "").split(" ");
        else
            return new String[0];
    }

//This is inside of our node class
    private String search(String key, String word) {
            if (key.equals("")) {
                if (this.word)
                    return word;
                else
                    return "";
            }

            int firstIndex = getIndexPerKey(key.charAt(0) - '0') * 3;
            StringBuilder sb = new StringBuilder();

            for (int i = 0; i < 3; i++) {
                String possibleWord = "";
                if (arrayCharacters[firstIndex+i] != null)
                    possibleWord =
                        arrayCharacters[firstIndex+i].search(key.substring(1),
                        word + findCharacter(firstIndex + i));
                sb.append(possibleWord);
                if (!possibleWord.endsWith(" "))
                    sb.append(' ');
            }
            return sb.toString();
        }
```