T9

Algorithms and data structures ID1021

Johan Montelius

Fall term 2022

Introduction

In this assignment you should implement the something that you might have used when your were a kid and had your first mobile phone. At the time when you had to text messages using a regular nine digit keypad, the T9 system made life easier. If you typed "43556" the text message would probably say "hello" and not "idklo" or something else that was not a word.

T9 kept track of all words that could possibly be encoded starting in a sequence of keys. If the user has typed '3' followed by '2' it knew that this narrows the set of words down to words starting in "ge", "he", "id" and "if" (assuming that there are no words starting in "gd" nor "gf" etc). T9 would use this information to display choices for the user or, if there was only one possible word, replace the sequence with a word. Sometimes the smartness of T9 gave surprising text messages but over all it worked great.

The true implementation of T9 was very clever in how it encoded the table of words in the dictionary and this was necessary since memory was in short supply. The implementation that you should do will not be as memory efficient but you will use the same structure as the original T9 implementation.

A strange tree - a trie

In this assignment you will not follow the T9 implementation exactly. To start with you will use a list of the most common Swedish words and we will then use only the characters 'a' to 'ö' but not 'q' nor 'w'. This means that we will have 27 characters which suits us fine since this is three characters for each of the keys '1' through '9'.

a node

To represent all the words in the list we will use a tree structure called a *trie*. The idea is to construct a tree where each node has as many branches as

there are letters in the alphabet. The root of the tree thus has 27 branches representing the words that start with: 'a', 'b', 'c' etc.

A node will apart from the set of branches also have a Boolean value indicating of the path from the root to the node is a valid word. We will call this *a valid node* although all nodes are valid these are the ones that represents valid words.

```
public class Node {
  public Node[] next;
  public boolean word;

  public Node() {
    next = new Node[27];
    word = false;
  }
}
```

A leaf in the tree will have the `next` value set to null and the `word` value to true. Note that the words themselves are never explicitly represented as strings, it's the path to a valid node (`word` set to `true`) that implicitly represents a word.

You might ask yourself why we use this strange tree to represent the words but it turns out that it is a very compact form if done right. In this Java implementation the set of 27 branches will take up some space but we could have coded them as 27 bits i.e. only one word. Searching for possible words is also quite efficient even if we could find more efficient representations.

code, index and key

Some terminology and methods that could come in handy are: *code* and *index*. Implement a method that given character ('a'...'ö') returns the code: 0...26. We will use these codes since they then can be used to address the branches of an array. Also implement the reverse method that given a code returns the character.

The second thing we will need is a method that given *a key* returns *an index*. The keys are: '1', '2' etc and the indices are the integers 0 ...9. We will use indices starting with 0 since we will use them to index an array.

The last thing we need, and this is not strictly needed but could be fun to have, is a method that returns *a key* given a character. This could be used to encode words so that you can turn "toffel" into "752224". It will come in handy when you do tests where you first insert a word and then make sure that you can actually find the word given the encoded sequence.

adding words

You will populate the tree by adding all words in a list. A word is added by starting in the root and then work your way down the tree given the indices of the characters in the word. If you find a branch empty you will of course have to construct that branch. If all nodes are already present then make sure that the last node is marked as a valid node.

The `add` method is surprisingly simple and if you only draw up what exactly is to be done you will write it down in twelve lines of code.

searching for words given a sequence

The lookup procedure is slightly more tricky but only because we are looking for all possible words and not just a single word. Implement a method `search`, that takes a key sequence (”2314”) and returns an array of all possible words that could match the sequence.

You implement this simply by starting in the root node and then search all possible alternatives given the first key. If the first key is ’2’ then this corresponds to the initial letters ’d’, ’e’ or ’f’. You will find these branches if you look at branches: 3, 4 and 5. Take the key ’2’, convert it to the index 1 and then examine branches $1 * 3$, $1 * 3 + 1$ and $1 * 3 + 2$. You can examine the branches by using the `search` method recursively. The three returned values should then be combined into one value.

As an argument to `search` you also provide a string and this is the string representing the path that you have taken. When you reach the end of the sequence you then have a string that could be a valid word. Looking at the node that you have, the Boolean `word` value will tell you if this is indeed a valid word.

Vanligaste orden i svenska

The word ”svenska” is among the two thousand most common words in Swedish text. The word ”i” is in the top hundred. The words ”vanligaste” and ”orden” are not among the most common but the words ”vanlig” and ”ord” are both in the top 500.

Give to you is a file containing the eight thousand most common words. This will be your source to populate the T9 tree. The file is a version of svenska ”Kelly-listan” [1] where we have removed multiple word expressions, any words containing ’w’ (only ”webb-”, ”show” and ”clown”) or ’q’ (only ”squash”), and changed words like ”idé” to ”ide”.

To make sure that your T9 implementation work you can first populate the tree, then for each word in the list encode it as a sequence of keys and finally do a decoding of this sequence. You will of course have cases where

the encoded version of a word is decoded resulting in two or three words but most words are decoded back to the original.

References

[1] Kilgarriff, Adam; Charalabopoulou, Frieda; Gavrilidou, Maria; Johannessen, Janne Bondi; Khalil, Saussan; Kokkinakis, Sofie Johansson; Lew, Robert; Sharoff, Serge; Vadlapudi, Ravikiran & Volodina, Elena Corpus-based vocabulary lists for language learners for nine languages Language Resources and Evaluation,48:121–163 DOI 10.1007/s10579-013-9251-2 2014