



DEPARTMENT OF ELECTRONICS

DATA STRUCTURES AND ALGORITHMS ASSESSMENT

## Predictive Text

*Y3839090*

January 14, 2017

## Contents

1	How the program works	2
1.1	How users interact with the program . . . . .	2
1.2	How the prediction function works . . . . .	2
2	Data Structures	2
2.1	Trie . . . . .	2
2.1.1	What is a trie . . . . .	2
2.1.2	Time complexity of a trie . . . . .	3
2.1.3	Space complexity of a trie . . . . .	3
2.1.4	How I have implemented the trie . . . . .	3
2.2	Alternative Data structures . . . . .	3
2.2.1	Binary Trees vs Tries . . . . .	3
2.2.2	Hash map vs Tries . . . . .	3
2.2.3	List vs Tries . . . . .	3
3	Testing	3
3.1	Complexity . . . . .	3
3.1.1	Time Complexity of Tries . . . . .	3
3.2	Unit Testing . . . . .	3
3.3	User interactions . . . . .	3

# 1 How the program works

dummy text

## 1.1 How users interact with the program

dummy text

## 1.2 How the prediction function works

dummy text

# 2 Data Structures

A predictive text program needs a to be able to access a set of common words. Storing and accessing these words efficiently is one of the challenges in creating a fast predictive text engine. I decided to use a trie[2] structure.

## 2.1 Trie

### 2.1.1 What is a trie

The trie is a tree-like data structure often used for storing strings. Each node in the trie represents a single letter in a string. Each node in a trie has a fixed number of child nodes like a binary search tree. Unlike the binary search tree the trie does not have two child nodes, it has one for each letter of the alphabet. This allows the trie nodes to not store their value because their position determines it. A node's children share a common prefix, that prefix is the value of there parent node.

There are many features of tries that make them a good choice for a predictive text system. One of them is that values can be look up by their prefixes. Predictive text systems use a partial word (a prefix) to search for possible full words. So it's important that prefix lookups are fast and efficient. [TODO:: ADD MORE STUFF HERE]

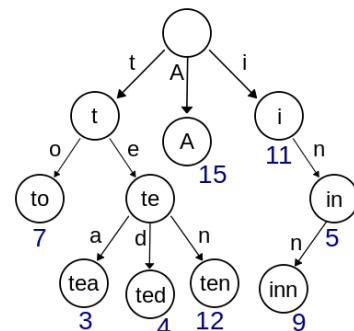


Figure 1: A trie for keys "A", "to", "tea", "ted", "ten", "i", "in", and "inn"[1].

### 2.1.2 Time complexity of a trie

The search time for a value in a trie is  $O(\text{len}(\text{string}))$ , where  $\text{len}(\text{string})$  is the length of the word to look up[3]. This means that searches in a trie are linear with respect to the number of items in the trie. A predictive text function relies upon looks. Every time a user presses a key the system will perform at least one lookup and so it is vital that these searches be fast.

The time complexity of deletion in a trie is also linear. Deletions time complexity is  $O(|A|\text{len}(\text{string}))$ , where  $\text{len}(\text{string})$  is the length of the word to look up and  $|A|$  is the size of the alphabet[3]. In the use case as a Predictive text engine deletions will be very rare or non-existent. A simple implementation of predictive text may not allow the user to delete words from the dictionary at all.

The time complexity of insertion in a trie is the same as for deletions. Insertion time complexity is  $O(|A|\text{len}(\text{string}))$ , where  $\text{len}(\text{string})$  is the length of the word to look up and  $|A|$  is the size of the alphabet[3]. When used in a predictive text engine most of the insertion into the trie will be made when loading the dictionary from file.

### 2.1.3 Space complexity of a trie

### 2.1.4 How I have implemented the trie

## 2.2 Alternative Data structures

### 2.2.1 Binary Trees vs Tries

### 2.2.2 Hash map vs Tries

### 2.2.3 List vs Tries

## 3 Testing

### 3.1 Complexity

#### 3.1.1 Time Complexity of Tries

### 3.2 Unit Testing

### 3.3 User interactions

## References

- [1] Booyabazooka (based on PNG image by Deco). Modifications by Superm401. *Example of a trie*. [Online; accessed 13-January-2017]. 2006. URL: [https://en.wikipedia.org/wiki/File:Trie\\_example.svg](https://en.wikipedia.org/wiki/File:Trie_example.svg).
- [2] Peter Brass. *Advanced Data Structures*. Cambridge University Press, 2008. Chap. 8.1, pp. 336–356.
- [3] Peter Brass. *Advanced Data Structures*. Cambridge University Press, 2008. Chap. 8.1, p. 341.