How to Write a Spelling Corrector

Aug 24, 2007

This is a Java 5/6 version for Norvig's spelling corrector.

The main objective was create the smallest Java version, and not focusing on performance.

In the Norvig's article, you can find the explanation, and other implementations (your favorite language can be there!).

As you can count, this implementation has **42** lines (or **35**, if you want to follow the Norvig counting and not include the blank lines). Not so beautiful as the Norvig's python (as any statically typed language), not so huge as the C version.

In order to use, you must:

1. download the dictionary file: big.txt (sorry Norvig, hotlink!)
2. compile the code in the command line: **javac Spelling.java**
3. exec in the command line: **java Spelling**
4. example: **java Spelling speling** should output **spelling**

You can compile it using an online Java compiler, like JavaXXX Compiler Service, selecting version as 1.5 or 1.6. No errors or warnings should be generated.

This implementation also uses only JRE default libraries, and don't requires external or custom libraries.

This code has LGPL, but if you want use it, be a nice guy and send me an email (rael.gc@gmail.com), I'll be happy!
If you find a bug, or find a way to improve the performance, please, send me the code, and I'll put the credits here.

History:
2007/08/13 - Original version
2007/08/15 - Removed a useless method (remained in code from the draft version)
2007/08/16 - Changed the static initializer to a constructor, removed the temp

buffer in the file loading
2007/08/16 - Removed similar methods
2007/08/17 - Replace the tailored max method for the java.util.Collections.max()
2007/09/03 - Additional notes added
2007/12/11 - Some tricks and more one line removed
2007/12/12 - Added a Groovy version
2008/12/07 - Some tricks and more one line removed (thanks to Anil Madamala, "Passionate about programming")
2009/02/09 - Useless line removed (thanks to Sergey Mikhanov)

```
import java.io.*;
import java.util.*;
import java.util.regex.*;

class Spelling {

    private final HashMap<String, Integer> nWords = new HashMap<String, Integer>(

    public Spelling(String file) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader(file));
        Pattern p = Pattern.compile("\\w+");
        for(String temp = ""; temp != null; temp = in.readLine()){
            Matcher m = p.matcher(temp.toLowerCase());
            while(m.find()) nWords.put((temp = m.group()), nWords.containsKey(tem
        }
        in.close();
    }

    private final ArrayList<String> edits(String word) {
        ArrayList<String> result = new ArrayList<String>();
        for(int i=0; i < word.length(); ++i) result.add(word.substring(0, i) + wo
        for(int i=0; i < word.length()-1; ++i) result.add(word.substring(0, i) +
        for(int i=0; i < word.length(); ++i) for(char c='a'; c <= 'z'; ++c) resul
        for(int i=0; i <= word.length(); ++i) for(char c='a'; c <= 'z'; ++c) resu
        return result;
    }

    public final String correct(String word) {
        if(nWords.containsKey(word)) return word;
        ArrayList<String> list = edits(word);
        HashMap<Integer, String> candidates = new HashMap<Integer, String>();
        for(String s : list) if(nWords.containsKey(s)) candidates.put(nWords.get(
        if(candidates.size() > 0) return candidates.get(Collections.max(candidate
        for(String s : list) for(String w : edits(s)) if(nWords.containsKey(w)) c
        return candidates.size() > 0 ? candidates.get(Collections.max(candidates.
```

```
    }

    public static void main(String args[]) throws IOException {
        if(args.length > 0) System.out.println((new Spelling("big.txt")).correct(
    }

}
```

Additional notes:

- In a real usage, you should not read the dictionary file (big.txt) all the times.
- Furthermore, to gain performance, the nWords object can be initialized with a preallocated size: the number of words in the dictionary file.
- Another performance tooltip is use some sorted Collection class as nWords type. This can improve the **constainsKey** method call, while add some penalty in the insertion time. But remember, you should read the dictionary file just one time.

Facts:

- I already wrote a version (at least) 50% faster than the current displayed here.
- You can reduce more lines wroting a more ugly code.

Groovy Version

This is a Groovy version and exists just for fun. It's too slow!
As you can count, it has **25** lines (or **22** without the blank lines).
It requires at least the 1.5 Groovy version.

```
def train(f){
    def n = [:]
    new File(f).eachLine{it.toLowerCase().eachMatch(/\w+/){n[it]=n[it]?n[it]+1:1}
    n
}

def edits(word) {
    def result = [], n = word.length()-1
    for(i in 0..n) result.add(word[0..<i] + word.substring(i+1))
    for(i in 0..n-1) result.add(word[0..<i] + word[i+1] + word[i, i+1] + word.sub
    for(i in 0..n) for(c in 'a'..'z') result.add(word[0..<i] + c + word.substring
```

```
    for(i in 0..n) for(c in 'a'..'z') result.add(word[0..<i] + c + word.substring
    result
}

def correct(word, nWords) {
    if(nWords[word]) return word
    def list = edits(word), candidates = [:]
    for(s in list) if(nWords[s]) candidates[nWords[s]] = s
    if(candidates.size() > 0) return candidates[candidates.keySet().max()]
    for(s in list) for(w in edits(s)) if(nWords[w]) candidates[nWords[w]] = w
    return candidates.size() > 0 ? candidates[candidates.keySet().max()] : word
}

println(correct(args[0], train("big.txt")))
```

Rael Gugelmin Cunha (aka Rael GC)

Disqus seems to be taking longer than usual. Reload?