CSCI-E250 Final Project

By: Ilya Levin

YouTube: http://youtu.be/UF3fMbAi2g8

-

Instructions

1. Unzip the file submitted
2. run “make all” in the directory
3. run “./spellchecker”
4. You will be give 5 options:
    a. Press 1 for the Edit Distance Alogortim -> See Spelling Mode
    b. Press 2 to use the Editex Distance Algorithm-> See Spelling Mode
        i. Note: (10 minutes to load the dictionary!)
    c. Press 3 to run a performance test Edit Distance. (~5-10 mins test time)
    d. Press 4 to run a performance test on the Editex Algorithm (~3 hours+ test time!!)

Spelling Checking Mode: After the dictionary is loaded, you can type in word, the output is a set of potential matches to misspelled word.

Performance Test: This loads the “misspelledwords.txt” which contains a list of misspelled words along with the correct spelling. The test runs the misspelled word through the spell check to see if the correct work is in the results set. After the test is done you’ll get a output of how many tests pass (found the misspelling), failed (didn’t find the misspelling), and the percentage of pass tests.

Brief Overview

The OCAML Spell Checker is a command-line spell checker. The goal of this project is to

implement various spell checking algorithms. The basic data structure I will use will be a BK-Tree. I will be comparing two different algorithms, Edit Distance, and Editex Distance. The basic problem I'm am trying to solve is given a misspelled word x, which algorithms performs better.

Edit Distance is the distances between two words by methods of insertions, deletions, and replacements. The distances between "boy" and "soy" is one since it just requires the replacement of "b" to "s". [1]

Editex Distance is similar to Edit Distance except it is based on sound. Figure 1 shows all the sound groups. As an example the distance between "par" and "far" would have a distance of 1 since "p" and "f" are in the same sound group. However "far" and "car" will have a distance of 2 since "f" and "c" are in different groups. See reference two for the actual implementation of this algorithm.

0	1	2	3	4	5	6	7	8	9
aeiouy	bp	ckq	dt	lr	mn	gj	fpv	sxz	csz

Figure 1: Sound Groups for Editex[2]

Planning

Overall I feel like my planning was pretty good. I got the most basic stuff out of the way very quickly. The BK-Trees and the Edit Distance function implementation was done on schedule and had it working within a few weeks. However when I switched to focusing on the Editex, I underestimated how long it would take.

Design and implementation

Design the spell checker application was very helpful. Original I wanted to create one functor that would take in a algorithm such as Edit Distance and simple return a Spelling Checking

---

[1] V. I. Levenshtein (1966) Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics, Doklady 10(8), 707-710

[2] Zobel, Justin and Dart, Philip. 1996. Phonetic string matching: Lessons from information retrieval.in Proceedings of the Eighteenth ACM SIGIR International Conference on Research and Development in Information Retrieval, Zurich, Switzerland, August 1996, pp. 166-173.

module.

However after talking to my TF (Perry Green) he advised me to create a few modules, one for the BK-Tree operations, another for the Spell Checking operations. This made a lot more sense and helped me gasp how to separate specific tasks in a clear way.

The Editex algorithm was the most difficult part of the assignment. Most of my time was spent researching the actually algorithm. There wasn't a lot of information about it and when I did find something the description or the pseudo code was either confusing or just plain wrong. I'm still not 100% sure I've implemented this correctly, but for the most part it seems to give a smaller number for words that sound are alike and a bigger number for words that sound different.

Also I wasn't expecting the algorithm to be so slow which made my comparison testing take a very long time.

Reflection

Looking back, I am actually surprised I was able to research and program all this project. At first writing a spell checker application in 3 weeks seemed like an impossible task. However once I started researching it, and figuring it out the basic implementation wasn't so bad. I feel that I've become more confident in reading research papers, and implementing complex data structures.

I was however frustrated that many research papers use very vague definitions in their description and pseudo code. One example is the pesudeo code "s_i" where s is a string. Many times the paper doesn't mention if the string starts with a 0 or a 1 which makes a huge difference in how the string comparison works. They also reference other algorithms assume the reader already knows them.

From the advice of my TF, the decision to split out the BK-Tree into a module and the SpellChecker into another module worked out great. It made the code easier to read and understand.

I also added another module called WordDistance that encapsulates the implementation of finding the distance between two words. This worked out really well since when I was done with my Editex function all I had to turn it into a WordDistance Module, and I could plug that in to my SpellChecker Module.

If I had more time, I would improve the performance of the application. It might of made more sense to create objects instead of modules. Also the application could go a lot faster if it was multi-trended. The only time the application needs to write to RAM is when it's building it's BK-Tree into memory. Once it's done I could create a new thread for each spell check request. Another way to improve the speed would be to serialize the BK-Tree into a file and deserialize it on load. Currently the BK-Tree is being recalculated every single time the application loads which is a waste of CPU cycles when you can just save it.

If I had to do this from scratch I think I would've implemented using objects instead of modules. This would avoid passing the dictionary around to all the methods, and I could just save that

information in a instance variable. I would of also stayed away from the Editex function and perhaps try to implement something that had more information on it.

Advice for future students

Get advice from your TF, they are a good resource to have, and could save you a lot of headache in the future.

Make sure you pick something you're interested in doing

Also make sure they is enough information out there for you to understand it. Not everything is on Google.

Original Spec

BKTREE = Leaf | Node String * INT * Node List (Updated this, decided to add an Edge type)

- BK Tree Module Function Signatures (Updated this a lot in the final spec, I needed a lot more functions)
    - loadFile (wordMap: (fun word -> int)) (fileName:String):BKTree
    - addword (wordMap: (fun word -> int)) (tree:BKTREE) (word:String):BKTree
    - spellCheck (search:function) (tree:BKTREE) (misspelledWord:string):String Option

- Levnshtein Functor- Input a BK Tree Module, outputs a Levenstein distance BK Tree Module
    - loadFile (fileName:String)
    - addWord (tree:BKTree) (word:String):BKTREE
    - spellCheck (tree:BKTree) (misspelledWord:String):String Option

- Editex Functor- Input a BKTREE Module, ouptuts a Editex distance BK Tree Module
- loadFile (fileName:String)
- addWord (tree:BKTree) (word:String):BKTREE
- spellCheck (tree:BKTree) (misspelledWord:String):String Option

The Main.ml file:

- The program will start by asking the user which algorithm they would like to use.
- Next, the program will open a file containing all the English words into the BKTree
- Once the file is loaded the command prompt will ask the user to type in a word. If the word can be found in the BK-Tree that word will be return. If it can't be founded it will return the word it thinks you are mean. If no words are found the spellCheck function will return None, and the user will see a friendly error message.

Final Spec

BKTREE = Node String * EDGE List
EDGE = (INT,BKTREE)

BK-Tree Interface - Public

- create ()
- addWord bktree word distanceFunction

Private functions

getElements edge - Converts an edge to a tuple
getList edge - gets the list of sub-nodes
getWord edge - gets the word within an edge
addEdge list word distance - adds a word to an existing list of edges.

distanceFunction must be Metric Space
(**http://en.wikipedia.org/wiki/Metric_space**)

WordDistance Module

val wordDistance : string -> string -> int

SpellCheck Interface - Public

loadDict fileName
spellCheck misspelledWord threshold
performanceTest

Private functions

spellCheck (toSearch:bktree list) misspelledWord threshold:string list
addToSearch listofEdges low high

Spell checker algorithm description: The search works similar to a breadth-first-search tree traversal, except if the edges values are not between the low and high values then the path will be ignored. The low and high values are calculated by editDistance(misspelledWord,currentNode)+threshold and editDistance(misspelledWord,currentNode)-threshold.

Pseudocode for spellcheck:

```
spellCheck (toSearch:bktree list) misspelledWord threshold
        bk::tail = take off head and tail from toSearch list
        d = editDistance (getRoot bk) misspelledWord
        toSearch = addToSearch tail (getList bk) (d-threshold) (d+threshold)
        if d is within threshold then
                (* add it to results and keep searching *)
                getWord bk::spellcheck toSearch misspelledWord threshold
        else if (List.length tail > 0)
                (* keep searching *)
                spellchecker toSearch misspelledWord threshold
        else
                (* we're done, return what we have so far *)
                []
```

Version Control:

I'm more familiar with SVN than GIT so I'm using Google Code:
http://code.google.com/p/ocaml-spellchecker/

The Main.ml file:

- The program will start by asking the user which algorithm they would like to use.
- Next, the program will open a file containing all the English words into the BKTree
- Once the file is loaded the command prompt will ask the user to type in a word. If the word can be found in the BK-Tree that word will be return. If it can't be founded it will return the word it thinks you are mean. If no words are found the spellCheck function will return None, and the user will see a friendly error message.