Extra Credit Report on CS 572 Apache Tika Project

About Problem

While reading the text from the parsed FBI document we encountered a problem where the OCR quality of scan was not perfect there were many conversion from scan to text that were misspelled. There were few occurrences where an alphabet is rendered as a random special character. So, even if the word was ***ufo*** in the document it was rendered as ***ueo*** or ***ufo*** which may be undetected by a program while searching for ***ufo***. The file became seemingly impossible and difficult to read and interpret accurately by a human reader.

Although we were able to read text from the parsed document, still we were not getting a perfect result due to bad OCR reading.

Additional Feature Implemented

To solve the above problem and get a better result set, a feature was added to the current program. This feature sets a threshold for **Edit distance** for every keyword and thus matches a keyword with a token within an edit distance.

Algorithm Used

The algorithm used is as follow:

1. While we have a Keyword, For every keyword we calculate keyword length
2. Calculate `keyThreshold` for every keyword using the formula :

```
int keyThreshold=keyLength/3;
```

3. Tokenize the stream containing text of the PDF file from corpus for which we have called the function `processfile`
4. For every Word token in the Tokenizer stream, calculate edit distance between keyword and the token using the formula

```
editDistance=getLevenshteinDistance(keyword,st.sval);
```

   For checking if we have a word token we use **StreamTokenizer** object ***st***

```
st.ttype == StreamTokenizer.TT_WORD
```

5. If `editDistance` value lies between keyword and token is less than or equal to threshold value for that keyword, it is considered as a match and we update the log file and increment the counter for the number of matches for that keyword.

Impact of Additional Features

This feature will detect the words even if the word was ***ufo*** in the document it was rendered as ***ueo*** or ***ufo*** which were undetected earlier thus leading us towards a perfect match. These additional keyword search definitely improves my program's ability to find UFO-related documents since those keywords which were missed earlier due to fact they were scanned bad, are now considered as well thus giving us much better results.

Vasu Jain vasujain@usc.edu 5019-3028-37

Additional Information

Since it was already mentioned at Piazaa forums, that we may borrow a standard code from internet for ***Levenshtein Distance***, I have used code for this available at http://www.java2s.com/Code/Java/Data-Type/FindtheLevenshteindistancebetweentwoStrings.htm

I have mentioned the credits to original author in the code and have used the following function to implement the Levenshtein Distance.

```
public static int getLevenshteinDistance(String s, String t)
```

Results

The additional usage of the ***Levenshtein Distance*** algorithm improved the efficiency of our program. Though, there is additional scope of improving efficiency of program using other methods where we may consider different forms of a keyword.

Although result now captures few words which possibly have been read wrong by OCR, usage of threshold of as low as one for ***Levenshtein distance*** for smaller keywords like ***UFO, Disc, UAP*** resulted in crediting many false hits as well. This has significantly increased the total number of matches for keywords. Thus using ***Edit distance algorithms*** for smaller keyword has increased inconsistency and redundancy instead of giving us more perfect result. However, edit distance algorithm result for keywords with greater length provided us some better results and keywords which were missed earlier while not using this algorithm.

Notes

Using the following program took over four hours in my laptop, being the complexity of this program high since we are using nested loops inside a loop. Thus it may take a while to get the results.

The number of extra keywords used in this program is total of 11 (6 given + 5 new) unlike 20 in the normal part of the project (one not using algorithm for extra credit).