1. Foreword
We are to design a java applet that takes in the following input, runs the given algorithm and produces the stated output. We define a string as a ordered set of characters with no whitespaces e.g. ab16 or Zq? or dog or _</s>yz.

Input
1) Correct string sequence input
2) Correct string <space> incorrect string <space> output number

Algorithm
1) Run edit distance to determine likely correct strings for the incorrect string, using (1) above as a list for edit distance to compare the incorrect string with.
2) For each correct string and likely correct string pair, determine the bigram probabilities for these and return this set.

Output
Likely correct string <space> correct string pairs ordered by edit distance and with corresponding bigram probabilities.

2. Prior work on sequence correction
The MS Word Spell Checker [12] uses
- Real time dynamic checking
- Has a large dictionary in sorted format, for rapid retrieval.
- Probably uses several surrounding strings e.g. trigram or n-gram to determine best correction.
- Probably uses heuristics based on surrounding strings, e.g. tense form and verb, noun and adjective classifications.
- The exact details are kept a trade secret

Our applet uses bigram and runs a one time static check on the input, if the user were to change any input he would have to click clear and recalculate. Also the user provides the previous correct word.

3. Description
Software development steps in time order
1) We use replaceAll() to prepare for the use of the split() function on the input.
2) Use ascending insertion sort on the edit distance array with a back pointer to original order maintained.
3) Check insertion sort using a separate java sort class.
4) Output the nearest 5 edit distance matches
5) Add in corresponding bigram probabilities
6) Test on different input cases.
7) Extend to output nearest n edit distance matches, where n is given by the user.
8) If id=-1 print The correct string you gave isn’t in the given string sequence.
9) Upload applet
10) Write instructions with a test case using the online applet input and output
11) Upload instructions

NLP/AI techniques used
- Bigram counting
- Relative probability
- Edit Distance
- Pattern Matching
- Sort and Sweep

NLP/AI online tools used
- None

Area
- NLP and Pattern Matching

3.1 Initial approach
For the input processing I tried using only the split method to extract the strings, however this added extra tags to identical strings e.g. if we have 2 t’s in the input, on the second t the split method would add a marker to signal this was the second occurrence of a t. Accordingly when it came to the sort and sweep to extract the unique strings the sweep part would keep the second t.

I used an Array List to store the original string sequence and used a built in method to remove duplicates but found it to be easier to operate on the original 1 dimensional array only. I also considered using a Hash
Map and adding key/value pairs, as the add() method prevents duplicate keys from being added.

For the storage and generation of the Bigram Table, I considered creating a separate class with 4 properties i.e. the first string the second string, the count and the probability, however a 3 dimensional array proved sufficient.

4. Code Analysis

http://www.geocities.com/singer.alexander/SCorrector1/schematic.pdf contains a schematic of the entire algorithm, it would be useful to consult this before proceeding. The schematic is the main part of this section.

The Process Input stage uses the temporary string variable tn to store the correct string input and get it ready to use the split() function on. We reduce the input to one long string with no leading or trailing white spaces and only a single whitespace between each string. This allows the input to have any number of trailing or leading whitespaces
newlines also any number of whitespaces
newlines between the individual strings

t[e] stores the individual strings of the correct string sequence. We take a copy of te[] as it is and store it in seq[]. Accordingly, the original string sequence is saved in seq[]. We proceed to remove the duplicates in te[] and use this for the bigram table.

When we have the output ready we set the .setEnabled property of the last Text Area to be true and display the trace.

Algorithm 1 Count Generation

```
for(int i=0; i<tec; i++) {
    for(int j=0; j<tec; j++) { 
        for(int k=0; k<seq.length-1; k++) {  
            if(   te[i].equals(seq[k])  &&
                te[j].equals(seq[k+1])  )
                t[i][j][0]++ ;
        }
    }
}
```

In Algorithm 1, the i refers to the first string or the string on the left of the bigram table. The j is an indexer for the second string or the string on the top of the bigram table. The k refers to the string we are checking in the original string sequence.

For the array t[row][col][0 or 1] a value of 0 in the third dimension indicates the count, 1 indicates the relative probability. Resources for the code implementation are [1] through [11] inclusive.

5. Conclusion

There are several options for future additions among them are

- Customizing the edit distance insertion, deletion and substitution costs e.g. substitution a to z has a cost of 10 and a substitution a to y has a cost of 9.
- Use trigram count or n-gram count, instead of the two string bigram count in current use.
- Have a mode to display only the final output section, the edit distance cost and the relative probability. This would be used for a large input set.

The applet design was good training for a larger software development project. The applet, along with instructions, can be found at http://www.geocities.com/singer.alexander/SCorrector1/SeqCorrector.html. If you require the code contact me.

References

[9] ArrayList Class http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html