Project 2

(2015 Fall)

Course name：J1799d Instructor：Ming Li TA：Wenbo Liu

ID number	**15213796** **15213782**	Name	**Luting Wang** **Dajian Li**
Tel	**+86 15622777988** **+86 13570300549**	Email	**364706267@qq.com** **dajianl@andrew.cmu.edu**
Starting date	**2015.9.23**	Finished date	**2015.10.1**

1、 Project requirement

In this project, we are required to write a routine to compute the Levenshtein distances between two symbol strings. We should implement two different pruning strategies, one based on a maximum string edit distance of 3, and another based on a "beam" of 3 relative to the current best score in any column of the trellis and provide switch to these strategies. After that, we are required to extend the routine to simultaneously compare a given text string to multiple templates and select the best one. In the routines above we must also display the trellis, the scores and the best path in table format. We will use our routine to check and correct the spell in *story.txt*, provided a dictionary (*dict.txt*). We will then adapt our Levenshtein distance computation routine to compare word strings and use the routine to compute the total number of errors between our spell-corrected version of the story and the correct story.

2、 Design your program

In our program, we set up a dictionary array by simultaneously read the words in *dict.txt*. In order to calculate distances in sequence, the dictionary is arranged as an

array of each character in the words with common suffix in front. Then we use *levenshtein* function to calculate the Levenshtein distances between a given string to multiple templates. In the *levenshtein* function, there are three strategies – non-pruning, fixed threshold pruning and beam pruning. For non-pruning strategy, we can simply calculate the *distance* matrix column by column. For the other two strategies, we should perform pruning operation to the words in the dictionary in each column. So we have a *pruned* array that contains flags for each word to note pruned words to avoid extra calculation. In order to find the best path we also have a matrix with the same size as the distance matrix to note the parent of each node in the trellis. In order

3、 Program implementation and testing

We implement the answer to Project 2 independently. Both implementations are in C++. Both implementations use similar design described in Part 2. But details are different.

For Luting Wang's implementation, she defined a structure for nodes in trellis:

```
struct Trellis {
    int score;
    int i;
    int j;
    bool valid;
    int insertion;
    int deletion;
    int substation;
};
```

The structure contains score, parent, valid flag, total number of deletion (up to this node), total number of insertion (up to this node) and total number of substitution (up to this node).

In multiple template routine, Luting defined a structure for words in dictionary.

```
struct Dict {
    int length;
    bool valid;
    string value;
};
```

Structure Dict contains the word, length of the word and flags for its validity (true if it is not pruned).

For the fixed threshold strategy, pruning operation is done right after calculation in a column is finished. The beam pruning strategy, however, requires to review the column to find the word to be pruned. Once a word is pruned, no more calculation of distance will be made in corresponding rows.

For Dajian's implementation, he used an array of every character of each words and an array of every words. He used pruned array to flag the pruned words, a distance matrix and a parent matrix. Everything is the same as discussed in Part 2.

Both of our implementation functions.

4、 Experimental results and discussion

1. Compare two word.

   Template: elephant

   Word: eleaphent

   If we use the first pruning strategy:

   The trellis is:

```
Display the trellis:
8 7 6 5 4 4 4 4 3 2
7 6 5 4 3 3 3 3 2 3
6 5 4 3 2 3 2 2 3 4
5 4 3 2 2 2 1 2 3 4
4 3 2 1 1 1 2 3 4 5
3 2 1 0 1 2 3 4 5 6
2 1 0 1 2 3 4 5 6 7
1 0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```

The best score:

```
Display the score:
2
```

The best path:

```
Display the best path:
(0,0)->(1,1)->(2,2)->(3,3)->(4,3)->(5,4)->(6,5)->(7,6)->(8,7)->(9,8)
```

The validity of each node, applying the first strategy or second strategy(1 means valid, and 0 means invalid):

```
validity of each nodes:
0 0 0 0 0 0 0 0 1 1
0 0 0 0 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 0
0 0 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 1 0 0 0
0 1 1 1 1 1 0 0 0 0
0 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

If we use the second pruning strategy:

The trellis is:

```
Display the trellis:
8 7 6 5 4 4 4 4 3 2
7 6 5 4 3 3 3 3 2 3
6 5 4 3 2 3 2 2 3 4
5 4 3 2 2 2 1 2 3 4
4 3 2 1 1 1 2 3 4 5
3 2 1 0 1 2 3 4 5 6
2 1 0 1 2 3 4 5 6 7
1 0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```

The best score:

```
Display the score:
2
```

The best path:

```
Display the best path:
(0,0)->(1,1)->(2,2)->(3,3)->(4,3)->(5,4)->(6,5)->(7,6)->(8,7)->(9,8)
```

The validity of each node, applying the first strategy or second strategy(1 means valid, and 0 means invalid):

```
validity of each nodes:
0 0 0 0 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1
0 0 0 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 0 0
0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

And if we use the first strategy to calculate the distance between “elephant” and “monkey”, it would be wrong, because the threshold is too tight to allow for longest input. We should use the beam search to do pruning.

2. Compare one word to multiple template

   The text file “test.txt” have three template, and we want to compare the word “ eleaphant ” to them.

   The trellis and best path are as follows:

```
please input the word:eleaphent
please input the pruning strategies you want to use, 0 indicates fixed threshold
, 1 indicates beam search1
Display the selected trellis:
8 7 6 5 4 4 4 4 3 2
7 6 5 4 3 3 3 3 2 3
6 5 4 3 2 3 2 2 3 4
5 4 3 2 2 2 1 2 3 4
4 3 2 1 1 1 2 3 4 5
3 2 1 0 1 2 3 4 5 6
2 1 0 1 2 3 4 5 6 7
1 0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
Display the score:
2
Display the best path:
(0,0)->(1,1)->(2,2)->(3,3)->(4,3)->(5,4)->(6,5)->(7,6)->(8,7)->(9,8)
the selected word is:elephant
```

3. Spelling Checker

Firstly, we remove all punctuation markers from the text and convert it all to lower case, “story.txt”.

We use “story.txt” to compare with the dictionary “dict.txt”, the spellchecked version of story is shown in “spellchecked.txt”.

The total computation time is about 30 s.

4. Compare word strings

Now, we get the spellchecked version of story, we want to compare the errors between the correct story version and it. The total errors are:

```
The total insertion is:0
The total deletion is:0
The total substution is:166
The total errors are:166
```

This is the result of Dajian’s program:

```
distance between two story:143
run time: 23
Program ended with exit code: 0
```

All Output

We briefly discuss some of our experiments.

The first experiment is to use Python to implement the routine, Python code could be found in the appendix. With After we correctly implemented the routine for multiple templates, we found that the average time for Python to find the best aligned word is 2-3 seconds. Finally, to correct the story took about half an hour. That is too long to wait.

The second experiment is the efficiency of the three strategies. We use Macbook Air (1.3GHz Intel Core i5, 4G 1600MHz DDR3) to run the routine of three strategies to correct the story. It turns out that the run time is Beam-Pruning > Non-Pruning > Fixed Threshold Pruning. We believe the reason is that beam pruning has one more loop, which induces computational complexity.

Choosing the first or the last word when they have the same score also makes a difference. According to our experiment, we tend to choose the last because choosing the last one will make less correction error (9 errors less).

The final discussion is about the dictionary. While looking at the dictionary, we found that some of the words in dictionary contains punctuation markers such as periods (calif. corp. v. x.), minus (dis-), semicolon (not; see;), colon (stanzas:) and even number (c1), which may induce ambiguity to word extraction. We believe that this is the reason that our results vary.