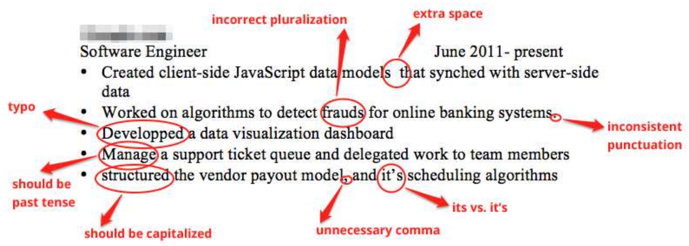
# CSCI 6350 (Barbosa)
# Project 1: Minimum Edit Distance

Due: See course calendar for due date. May not be turned in late.

Assignment ID: **proj1**
File(s) to be submitted: **proj1.py, myout.log**



[redacted]
Software Engineer June 2011- present
- Created client-side JavaScript data models  that synched with server-side data
- Worked on algorithms to detect frauds for online banking systems.
- Developped a data visualization dashboard
- Manage a support ticket queue and delegated work to team members
- structured the vendor payout model, and it's scheduling algorithms

Objective(s): 1) Implement minimum edit distance algorithm; 2) enhance the algorithm with backtracing to output edit operations; 3) provide an analysis of comparing the Levenshtein costs with those of the provided confusion matrix.

Project description:
In this assignment you will have the opportunity to implement the minimum edit distance algorithm discussed in class. You will write a small Python program that constructs the minimum cost table (using dynamic programming), and enhances it with backtracing "pointers" to output edit operations.

Requirements:

0a. All projects in this course will use file input only, and will not use interactive keyboard input, unless otherwise stated.

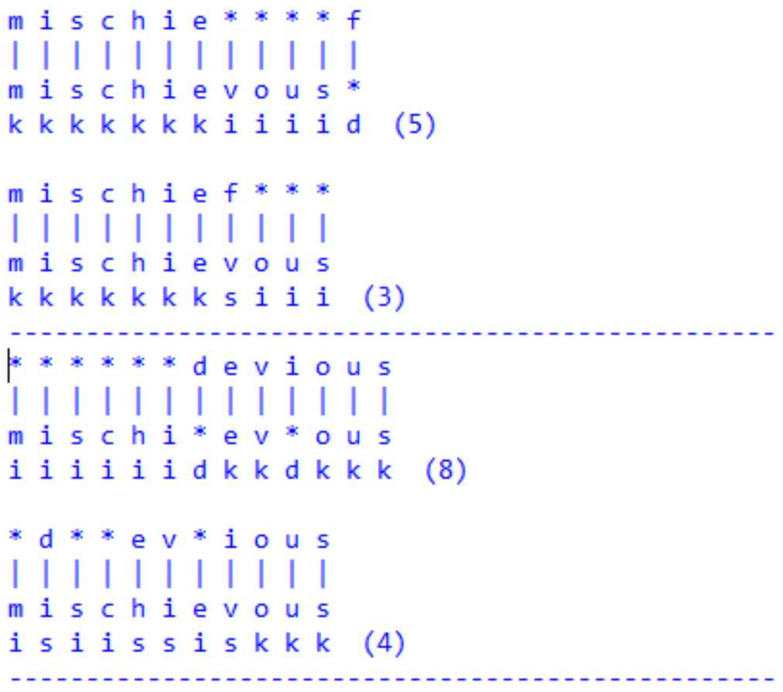
0b. All projects in this course will use a **flat directory structure** (the program and input/output files must be at the same level in the directory/folder structure), unless otherwise stated.

1. (**60%**) Your program must provide the functionality listed below, in a single file named ***proj1.py***:

- *Input* – The input consists of sets of words (one set per line in **lowercase**) for which the minimum edit distance should be calculated. The first word in each line is the **target** word. All other words in the line are **source** words that must be transformed to the target word (using the minimum edit distance algorithm). The input file must be named ***words.txt***. Two additional input files are provided (in the same format):
    - ***costs.csv*** – a comma-delimited file containing the Levenshtein <u>substitution</u> costs for lowercase alphabet
    - ***costs.csv*** – a comma-delimited file containing the confusion matrix edit <u>substitution</u> costs
- *Processing* – The processing requirements include:
    - The cost of insertions and deletions is 1 in all cases. Substitution costs will be read from input files.
    - For each pair of source and target words, calculate the minimum edit distance (using both Levenshtein and confusion matrix costs), and output the cost and backtrace of operations (see below for details).
    - The dynamic programming table must be complete and correct. The backtrace table must capture all possible sources for the minimum cost at each cell.
    - When constructing the backtrace, randomly select any one of the possible cells that provide the minimum cost to the cell being processed. This should be done by importing the *random* module, and ensuring that all possibilities have an equal probability of being selected.
    - DO NOT seed the random number generator in the submitted code. Your program will be run multiple times so that different sets of edit operations are viewed for each pair of source and target words.
- *Output* – For each pair of source and target words, the program should display the following output:
    - 4 lines <u>for each of the cost methods</u> (Levenshtein and confusion matrix) – See Fig 2.14 in the textbook
        1. Line 1 will show the **source** word
        2. Line 2 will contain a vertical bar ("|") for each operation (one per character)
        3. Line 3 will show the **target** word

4. Line 4 show the operations for each character – The only change from what's in the textbook is that you must use the letter ***k*** (for *keep*) to indicate a null substitution (rather than a space).

- A line of 50 hyphens should be used to separate the pair of words from the next pair
- Example output:



```
m i s c h i e * * * * f
| | | | | | | | | | | |
m i s c h i e v o u s *
k k k k k k k i i i i d  (5)

m i s c h i e f * * *
| | | | | | | | | | |
m i s c h i e v o u s
k k k k k k k s i i i  (3)
--------------------------------------------------
* * * * * * d e v i o u s
| | | | | | | | | | | | |
m i s c h i * e v * o u s
i i i i i i d k k d k k k  (8)

* d * * e v * i o u s
| | | | | | | | | | |
m i s c h i e v o u s
i s i i s s i s k k k  (4)
--------------------------------------------------
```

2. (**15%**) Test your program – Run it multiple times to ensure that different backtraces are selected. Assess the correctness of each backtrace.

3. (**5%**) Code comments - Add the following comments to your code:

- A section at the top of the source file(s) with the following identifying information:

  Your Name
  CSCI 6350-001
  Project #X
  Due: mm/dd/yy

- Below your name add comments in each file that gives an overview of the program or class.

- Place a one or two line comment above each function that summarizes the workings of the function, and the names and purposes of parameters .

4. (**20 %**) Analysis – Provide an answer/analysis (as commented code) for the following questions:

a) Compare and contrast the results obtained from using the different cost approaches. Is one "better" than the other? How? Why?
b) While the algorithm you implemented yields the minimum edit distance between a pair of words, it is not clear how it fits into a larger context (e.g., where does it get the words from?). Provide a description of its plausible use in a natural language application context (how does one arrive at the candidate words, how is the correct spelling selected, etc.)
c) Explain how you might devise a new set of costs: what process would you go through? What data would you use or collect? How would you arrive at final values for the table?

**Project submission**:

Once logged into *ranger*, go to the directory containing your Python program. At the Linux $ prompt, type in ls (for list) to make sure the Python script file is listed in this directory. Issue the following commands at the prompt to create a script file to be submitted with the Python code.

- *script myout.log*
- *pwd*
- *ls -l*
- *cat –n proj1.py*
- *python3 proj1.py*
- *exit*

The *handin* command allows assignments to be submitted electronically. The command handin must be followed by assignment ID and the files required. The files must be named exactly as listed here (although they may appear in any order). Submit your assignment from the directory where your source and log files are, with the following command (you may resubmit your work – it simply overwrites the previous submission. Your last submission will be graded).

```
handin proj1 proj1.py myout.log
```

Alternately, go to the course syllabus web page, and select the Gus icon near the bottom of the page's contents. Login with your class account username and password. Select the appropriate project from the dropdown menu, check the Submit button, and click the Perform Action button. Use the Choose File button to navigate to the location of the files. Please note that the filename is case-sensitive, and must be entered exactly as shown on the page. Click the Upload button to submit your assignment. You may submit a newer version of the assignment, all the way up to the deadline. It overwrites your previous submission (which is not recoverable). All project files must be included in each and every submission.