# Comp 3760 Assignment 2: Edit Steps v5

***Due date: 11:00pm Sunday November 23, 2008.***

## Background

A common word puzzle found in many newspapers and magazines is the word transformation. By taking a starting word and successively altering a single letter to make a new word, one can build a sequence of words which changes the original word to a given end word.

For instance, the word "*spice*" can be transformed in four steps to the word "*stock*" according to the following sequence: *spice, slice, slick, stick, stock*. Each successive word differs from the previous word in only a single character position - ie - one character can be changed, added, or removed.

This assignment has two parts, as follows:

**Part 1: Shortest Transformations**

Given a dictionary of words from which to make transformations, plus a list of starting and ending words, you are to write a program to determine the number of steps in the shortest possible transformation of the starting word into the ending word. For example, if the dictionary contains { *dip lip mad map may pad pip pod pop sap sip* } we can transform "*sap*" to "*pad*" in 3 steps according to the following sequence: *sap, map, mad, pad*.

**Part 2: Longest Edit Step Ladder**

Formally, each transformation is known as an "edit step". So the transformation from *dig* to *dog* or from *dog* to *do* are both edit steps. An "edit step ladder" is a lexicographically ordered sequence of words $w_1$, $w_2$, ... $w_n$ such that the transformation from $w_i$ to $w_{i+1}$ is an edit step for all i from 1 to n-1.

For a given dictionary, you are to compute the length of the longest edit step ladder.

## Input

Input is from a file. Each input file starts with a line containing a single integer **`t`**, where $1 \le t \le 100$, indicating the number of test cases that follow. Each test case begins with a single comment line the consists of two forward slashes followed by a test case name. The entire comment line (including forward slashes) will be at most 40 characters.

Each test case begins immediately after its comment line, and has two sections. The first section will be the dictionary of available words with one word per line, terminated by a line containing an asterisk (*) and an integer **`w`**, where $1 \le w \le 100$.

Following the dictionary are **`w`** pairs of words, one pair per line, with the words in the pair separated by a single space. These pairs represent the starting and ending words in a transformation. The starting and ending words will appear in the dictionary.

There can be up to 2000 words in the dictionary. Words will be alphabetic and in lower case, and no word will be longer than ten characters. Words can appear in the dictionary in any order.

## Output

Output is to the console.

The output for each test case will begin with the comment, followed by one line indicating the length of the shortest transformation for each pair of starting and ending words, followed by a line indicating the longest edit step ladder.

Each shortest transformation line must include the starting word, the ending word, and the number of steps in the shortest possible transformation, separated by single spaces. If it is not possible to transform a starting word to an ending word using the supplied dictionary, your program will output the string "(not possible)" for the number of steps.

The output line for the longest edit step ladder consists of the phrase "ladder: " followed by a single integer indicating the number of steps in the longest edit step ladder.

The output for each test case must be separated by a blank line.

## Sample Input

```
2
// Test case 1
dig
dip
lip
mad
map
maple
may
pad
pip
pod
pop
sap
sip
slice
slick
spice
stick
stock
* 2
spice stock
may pod
// Test case 2
cat
dig
dog
fig
fin
fine
fog
log
wine
* 3
dig fine
cat dog
fig fin
```

## Sample Output

```
// Test case 1
spice stock 4
may pod 3
ladder: 5

// Test case 2
dig fine 3
cat dog (not possible)
fig fin 1
ladder: 5
```

## Other Requirements and Guidelines

1. ***You may work in pairs***, or by yourself. If you work with someone else, each person is expected to contribute equally to the assignment. Reports of partner-exploitation should be directed to the course instructor. The names and ID's of both partners must be included as per the submission guidelines below.
2. For this assignment you ***must use Java***, and your program must compile using the Java 6 release.
3. Your program must read the ***input from a text file***. The name of the text file will be passed as the only ***command line argument*** (as per the following submission requirements).
4. You are free to use any of the data structures or algorithms in the ***Java Collections Framework***, or you can develop your own structures and algorithms if you prefer.
5. ***You can use any Graph class and/or graph algorithms you like as long as you cite your sources (ie: graph code from the internet is fine). This means that if you did not write it yourself you must leave the name of the original author in the file. Yes, you can get it from the internet.***
6. ***You must not copy or borrow another students work.***
7. Your program will be evaluated against the ***instructors test files***, which will not be provided in advance. It is up to you to understand the requirements and make sure your solution meets them.
8. Programming style, design, technique, and inline documentation are up to you. This is not a course in software engineering or software design, however, you are expected to write code that can be easily understood by the instructor, and adheres to ***sound principles of program design*** as taught to you in previous CST courses. Each source code file in your submission must include a ***header block*** listing the author(s) name(s), date, and description of the file contents.
9. These requirements are subject to change. Any clarifications or changes will be announced in WebCT, and documented in a new version of this file. It is your responsibility to ***monitor WebCT*** so that you are aware of any assignment updates or changes.

## Submission Details

To receive full marks, your assignment must be submitted to WebCT before ***11:00pm Sunday November 23, 2008*****.** WebCT will be configured such that assignments submitted after this time will be marked as ***late***. Late assignments will be graded as follows:

- for each day (24hrs or portion thereof) the assignment is late, 15 marks will be deducted, up to a maximum of 75. Programs that are more than 5 days late will not be graded.

There will be no exceptions to this policy, regardless of your reason. It is up to you to manage your time and get the assignment submitted ***before*** the deadline.

Your submitted assignment will consist of a single zip file. When your submitted zip file is unzipped, it should create a directory with your real name(s) (initial & last name). For example, if I was working with Albert on the assignment, unzipping will create a directory called: rneilson_awei (or something similar).

Within this directory I expect to find a directory containing all the Java source files required to build your program. Your submission must not include java SDK files, and must meet the following criteria:

a. Your "main" function must be in a class called Main, in a package called "editsteps", and in a subdirectory called "editsteps" under the directory created when your submitted file is unzipped. In the above example, this means main should be in the file rneilson_awei\editsteps\Main.java
b. Note that packages correspond to subdirectories. This means you need to put the line: package editsteps; in Main.java. You will probably need to do the same with the other source files.

*How do I test your assignment?*

1. I don't use NetBeans. I just compile your program using javac. Your program must compile fine when I use the following commands to compile it: `javac editsteps\Main.java` (assuming I am in directory rneilson_awei).
2. Next I will execute your program, passing the name of one of my input files, and redirecting the output to a file. For example: `java editsteps/Main testdata.txt > out.txt`

## Grading and Evaluation

Your program will be graded according to the marks distribution shown below. The instructor's test data is designed to make sure you understood and followed the requirements. Don't assume anything!

Within each category, marks will be assigned for choice of data structures and algorithms, as well as correctness and efficiency of the solution(s).

Programs that do not compile will receive a grade of zero.

Programs that do not read the input file (ie: crash on input) will receive a grade of zero.

Programs that do not accept the file name as a command line argument will receive a maximum grade of 75/100.

Lots of marks will be deducted for programs that are poorly designed (ie: lack modularization and/or are confusing and difficult to understand.

### Mark Distribution

Ability to follow instructions 10%

- Did you follow instructions for naming of directories, building, etc?

Program Design 20%

The program design portion of your grade will be based on (but not limited to) things like

- Choice of algorithms and data structures.
- Modularity, readability, and overall organization of the program.

Output Formatting 10%

- The output is in exactly the format specified in the assignment, with no extraneous output.

Correctness of Solution 60%

The correctness portion of your grade will be based on (***but not limited to***) things like

- Program runs without error / exception.
- Output is correct.

## Revision History

V3 changes are marked in BLUE

V4 changes are marked in RED

V5 changes are in GREEN