# TENTH ANNUAL JUILFS CONTEST
# SPRING 2018



Department of Computer and Information Science
University of Oregon
*2018 June 02*

contributors: Trevor Enright, Jordan Lewis, Amber Straub, Joe Sventek, Chris Wilson

# A. Change Letter Cases

(8 points)

| | |
|---|---|
| F | a |
| E | b |
| A | c |
| D | d |
| B | e |
| C | f |

In posting to a Twitter feed, a person uses a mix of capital letters, lower-case letters, and punctuation. Sometimes one wishes to mix up the posting style. To assist this, your job is to write a program that will

- change upper-case letters to lower-case,
- change lower-case letters to upper-case,
- leave punctuation characters unchanged, and
- if there are no lower-case characters in the output string, add "...AND I MEAN IT!!" to the end.

The input will be a number n, $1 \le n \le 100$, followed by n phrases of at most 140 characters. Here a phrase is a string consisting of up to 140 characters – these will be upper case letters [A-Z], lower case letters [a-z], and other punctuation characters. The output should be n strings with the case of the letters changed, as described above.

For example, on the input

```
4
It's really that simple: Program, or be programmed.
yes! and no !!
NOPE
1234 56789
```

the desired output is

```
iT'S REALLY THAT SIMPLE: pROGRAM, OR BE PROGRAMMED.
YES! AND NO !!...AND I MEAN IT!!
nope
1234 56789...AND I MEAN IT!!
```

# C. Word Frequency

## 9 Points

**Problem**

Jordan is interested in collecting data about the frequency of words in a text file. For each word, Jordan wants to know how many times it occurred, it's first occurrence, it's last occurrence, a list of all occurrences, and the most common word. Your job is to help Jordan by writing a program that reads the words from a file via standard input, and prints data for the user.

**Input**

The file will be a list of **N** words (duplicates included), with one word on each line.

**Output**

In alphabetical order, output each word on a line containing "*w* - (*a*, *b*, *c*, [*x*, *y*, *z*, ...])" where w is the word, *a* is the number of occurrences, *b* is the line number of the first occurrence, *c* is the line number of the last occurrence, and [*x*, *y*, *z*, ...] is a list of line numbers corresponding to each occurrence (brackets required). After printing each word, print "Most common word - foo: bar" where foo is the actual word and bar is the number of occurrences. Break ties for most common word based on alphabetical order.

Print all information even if the word only occurs once.

| **Sample Input:** | **Sample Output:** |
|---|---|
| Apple<br>Orange<br>Orange<br>Pear<br>Banana<br>Apple | Apple - (2, 0, 5, [0, 5])<br>Banana - (1, 4, 4, [4])<br>Orange - (2, 1, 2, [1, 2])<br>Pear - (1, 3, 3, [3])<br>Most common word - Apple: 2 |

# D. Hangman
(10 points)

One word hangman.

You walk in on your sister and nephew playing hangman. The idea of their game is that one player picks a word and draws as many blank spaces (underscores) as there are letters in the word. They are allowed to give hints and fill in letters of the word but must fill in every occurrence of the letters they choose. Additionally, they may give zero or more letters that are *not* in the word. The other player then guesses a letter that is in the word. If the guess is correct then the letter guessed is filled into the correct places in the word and the game continues. If the guess is wrong then it's one step closer to the gallows. To help your little nephew out you decide to write a script to help him guess a letter that is in the word. The script should output a single letter that is next best guess for your nephew. A dictionary file will be provided consisting of alphabetized words, one word per line.

Input:
The input will consist of a line with a single number $n$ with $0 < n < 100$, which gives the number of words that need a letter to be guessed. Each of the $n$ words will consist of exactly two lines of input. The first line consists of the *input word* with $k$ underscores with $0 < k < 23$. Where $k$-1 underscores may be replaced with lowercase letters in the alphabet or a hyphen ("-"). The second line will consists of $j$ letters with $0 < j < 27$, this string of characters that are *not* used in the word and therefore do not repeat or contain any of the letters that have replaced underscores.

Output:
Let the best next guess be determined by the most common letter of the words in the dictionary that could possibly be the *input word*. For each word, the output should be either a single letter that is the best next guess or "word not in dictionary".

**Note**: You will be provided with a dictionary named "words10k.txt". Use the name of that file in your program. The judges will run your program in a directory containing it.

For example:

_e_t ← first line (*input word*)
ansl ← second line (letters not in word)

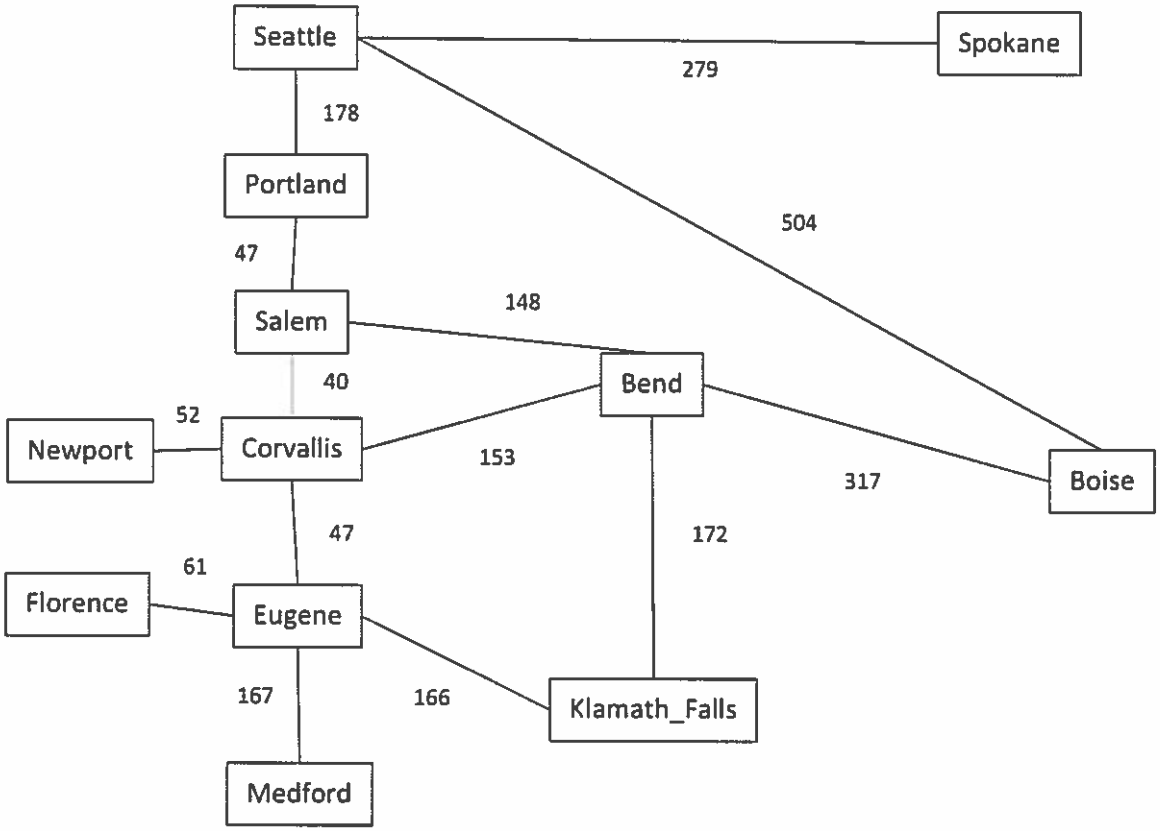
The next best letter to guess would be "p" since the only words in the dictionary that it could be are the following: 'debt', 'deft', 'heft', 'kept', 'pert', 'wept'
Where there are three words with the letter "p" in them and only two or one of any other letter.

*Sample input on back*

# E. Transportation Planning

(11 points)

A transportation planner is interested in knowing the maximum distance one must travel to move between any two points in a country (i.e., the maximum of all the shortest distances between pairs of cities) and wants you to develop a program to do so. For example, consider this simple road system, where all roads are bidirectional:




For this road system, the maximum distance would be that between Spokane and Boise: 783 miles.

The input to your program consists of a line with a single integer, $N \leq 1000$, indicating the number of road segments between cities. This is followed by $N$ lines, each of which describes a road segment, formatted as follows:

*city1 city2 distance*

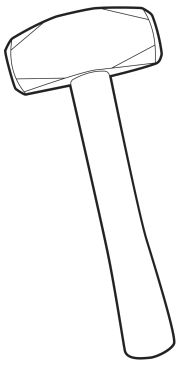
where *city1* and *city2* are the names of two cities without embedded blanks, and *distance* is an integer specifying the number of miles between the cities. Each road segment is bidirectional.

You are to output the maximum distance between any pair of cities defined by the road segments as an integer.

*Sample input on back*

# **F**. Check Secret Powers

(8 points)



The super-secret *Secret Oregon Bureau* (SOB) is committed to keep Oregon's secrets safe from its neighboring states. And because it is *the* SOB it needs a secure cryptographic protocol. A special team from the CIS department has invented a super-secure method and it needs your help. Should you choose to participate, you will be asked to help determine if certain numbers have special powers.

We say an integer $p$ has special powers with respect to $q$ and r if we can write it as $p = q^e * r^f$, for integers $e$ and $f$. For example, 200 is special for 2 and 5 since $200=2^3*5^2$, but 200 is not special for 20 and 25.

The input will be a number n, $1 \le n \le 100$, followed by n lines of the form $p\ q\ r$. These integers satisfy $0 \le p \le 2147483647$ and $2 \le q,r \le 2147483647$ If $p$ can be expressed as a power of $q$ and $r$ as described, then output "`p is a power of q and r`", otherwise say "`p is not a power of q and r`".

For example, on the input

```
9
0 2 3
1 2 3
2 2 3
3 2 3
5 2 3
200 2 5
32 4 8
2038431744 4 6
200 20 25
```

the desired output is

```
0 is not a power of 2 and 3
1 is a power of 2 and 3
2 is a power of 2 and 3
3 is a power of 2 and 3
5 is not a power of 2 and 3
200 is a power of 2 and 5
32 is a power of 4 and 8
2038431744 is a power of 4 and 6
200 is not a power of 20 and 25
```