Fall 2016 COP 3223H Program #3: Tallying Votes
Due date: Please consult WebCourses for your section

Objective

1. To learn how to use a dictionary to efficiently store and edit information related to a number of keys.

Problem A: Vote Tally (tally.py)

All the election data comes to a centralized clearinghouse where all the votes for a state are counted. We can model this data coming in by reading a large stream of data from a file. Every time votes are reported, two tokens are added to the stream of data:

(1) The name of the candidate
(2) The number of votes recorded for that candidate.

For the purposes of this problem, we'll use only first names comprised of letters only for (1). The number of votes will be a positive integer. The sum of all of the votes in a file won't exceed 300,000,000 and the number of unique candidates in the file won't exceed 100.

Thus, the input file, **`allvotes.txt`**, is a sequence of pairs of tokens in this fashion. Here is a sample input file:

```
Allie 10   Bill 3
Allie 5 Carmen 7
Carmen 2
Bill 1
Allie 22
```

Notice that all that is guaranteed is that each token is separated by white space and that within a pair of tokens, the name comes first, followed by the number of votes.

Your job will be to print out a sorted list of candidates by number of votes. You will be guaranteed that no two candidates will have the same number of votes (we don't want a repeat of 2000!!!) Print one candidate per line.

Implementation Requirements

For this problem, you must use a dictionary with the keys being each candidate and the values they are mapped to being the total number of votes they have received. Whenever you read in a pair, you must first check to see if that candidate is in the dictionary yet. If he/she is, add the votes to their tally. If not, add the appropriate entry to the dictionary.

Please read your input from **`allvotes.txt`**.

One Method to Handle Sorting the Output

We haven't learned how to sort this sort of data in class, so here is one simple idea (not efficient, but relatively easy to implement):

After filling the dictionary completely with the final vote tally,

1) Run a loop through the dictionary n times, where n is the length of the dictionary (ie. total number of candidates receiving at least one vote)

2) In each loop iteration, find the candidate with the most votes and print out this information. (Note: the course text shows how to find the maximum in a list. So review any example where the maximum value in a list of numbers is found and apply the general concept to this problem.)

3) Set the number of votes for the candidate printed to -1, so that it is no longer the maximum.

Once this looping mechanism has finished, each candidate will be printed one by one in this double loop structure and the dictionary info will be completely destroyed storing -1s for each candidate. (This is fine because at this point the program will be completed.)

If you want to do something more efficient or fancy, feel free to do so, but you aren't required to.

One Method to Read from a File

Please follow the censor.py example shown in class. First do:

```
myfile = open('allvotes.txt','r')
mystr = myfile.read()
voteinfo = mystr.split()
```

From here, index the items in the list voteinfo in pairs, at indexes i and i+1, where i starts at 0 and jumps by 2 each time. This information will be strings, so you'll have to use the int function to convert some of those strings to the appropriate type. Remember that you can obtain the length of the list by using the len function.

Input Specification

The input file will be named **`allvotes.txt`**. It's file format is exactly as described above. Pairs of tokens will be separated with white space, where the first item in each pair is the candidate name, all upper case letters, and the second item is the number of votes that candidate is receiving.

Output Specification

Output one line for each candidate - a sorted list of all candidates by vote. For each line, list the name of the candidate, followed by a space, followed by the number of votes they have received.

Sample Input (allvotes.txt)

```
Allie 10   Bill 3
Allie 5 Carmen 7
Carmen 2
Bill 1
Allie 22
```

Sample Output

```
Allie 37
Carmen 9
Bill 4
```

Deliverables

One source files, *tally.py*, for your solution to the problem.

All files are to be submitted over WebCourses.

Restrictions

Although you may use other compilers and coding environments, your program must run in IDLE for Python 3.2.5.

Grading Details

Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

Note: As mentioned in class, the input specifications are there to help you. Those are the requirements I guarantee I'll stick to when testing your program. You don't need to check if the user enters values within those parameters.