Department of Computer Science and Engineering
University of Puerto Rico
Mayagüez Campus

CIIC 4020 / ICOM 4035 – Data Structures
Spring 2019-2020
Project #1 – Alternate Voting Method

Introduction

It's election year in the country of Poor Harbor and changes are being considered for the voting system. The current system allows for the candidate with the highest amount of votes to be declared as the winner. However, in the last two elections the winners received less than 50% of the votes, causing citizens to question whether it was legitimate to have a president that was rejected by more than half of the voters. The lawmakers of Poor Harbor have since decided to implement an alternate voting method.

Using this alternate voting method, if there are *n* candidates running for a particular position, voters can vote for all *n* candidates, giving each candidate a rank of preference, from 1 to *n*. In general, an election may include several elected positions, with several candidates running for each position, and several cast ballots. However, we'll focus on an election for a single position.

You will write a program that is able to receive data about the candidates for the position and votes cast in the election, then process the votes to determine the final results based on the counting rules that we are about to discuss.

Some people might claim that the new voting system exhibits higher fairness than the current one, while others might disagree. No voting system is perfect, but it is outside the scope of this project to discuss all of the pros and cons of these systems in particular.

Valid ballots

In each ballot voters can vote for any number of candidates from 0 (blank ballot) up to *n*, where *n* is the number of candidates running for that particular position. The voter can rank each candidate according to his/her preference, assigning the value 1 to the highest preference, 2 to the second-highest preference, and so on, with a higher value indicating a less-preferred

candidate. No ranking value can be repeated in a ballot; otherwise, the ballot is invalid. Also, if the voter chooses to vote for $k < n$ candidates, the ranking values used must be the values 1 to k. In that case, if a ranking value greater than k is given, the ballot is also considered invalid.

Counting rules

The counting process may consist of several rounds or iterations. In each round, if one candidate has more than 50% of 1's, that candidate wins the election for that position and the counting process concludes. Otherwise, the candidate having the lowest amount of 1's is eliminated and will not be considered in the subsequent counting rounds. If there is a tie for lowest amount of 1's, then the decision is based on the lowest amount of 2's (between those that tied for the lowest amount of 1's). If again there is a tie, then the decision is based on the lowest amount of 3's (between those that tied for the lowest amount of 2's), and so on. If at the end no decision can be made, then, among all of those candidates that are still tied, the one having the current largest ID# is removed.

When a candidate is eliminated, the counting for the next round must take into account the voters' preferences. If the #1 candidate in a ballot is eliminated, then the higher-ranked candidates are "moved up" a position, so that the #2 candidate now becomes #1, the #3 candidate now becomes #2, and so on. In general, if the candidate being eliminated was given a rank value of k in a ballot, then all candidates on that ballot with a rank greater than k will have their rank subtracted by 1, hence giving them a lower rank (higher preference).

Election example

Consider the following election with 5 candidates and 10 ballots. Notice that no candidate has more than five (50%) 1's.

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	4	4	5	1	1	4	1	3	4
Juan	2	3	2	1	2	5	2	2	4	2
Lola	3	1	3	3	4	3	5	4	2	3
Pardeep	4	5	1	2	3	2	3	5	5	1
Pucho	5	2	5	4	5	4	1	3	1	5

Round 1: Notice that Juan and Lola both have the least amount of 1's. Then, following the counting rules, we compare the amount of 2's (only between Juan and Lola). Juan has six 2's and Lola has only one, so Lola is eliminated. The ballots are updated as follows:

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	3	3	4	1	1	4	1	2	3
Juan	2	2	2	1	2	4	2	2	3	2
Pardeep	3	4	1	2	3	2	3	4	4	1
Pucho	4	1	4	3	4	3	1	3	1	4

Round 2: Juan has the least amount of 1's, so he is eliminated. The ballots are updated as follows:

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	2	2	3	1	1	3	1	2	2
Pardeep	2	3	1	1	2	2	2	3	3	1
Pucho	3	1	3	2	3	3	1	2	1	3

Round 3: No candidate has yet reached at least 50% of 1's, so we need to eliminate another candidate. Pardeep and Pucho are tied for 1's, but Pardeep has more 2's, so Pucho is eliminated. The ballots are updated as follows:

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	1	2	2	1	1	2	1	1	2
Pardeep	2	2	1	1	2	2	1	2	2	1

Final result: Pepe is the winner!

Input data

The input for the program is to be read from two data files located in the same directory as your program.

`candidates.csv`: Contains the data about the candidates. Each line contains two fields of information separated by a comma: The name of the candidate and the ID# assigned to that candidate. *Note that the ID numbers are 1-based* (lawmakers don't know about 0-based indexing). The candidates appear in the file in ascending order **by ID #**, starting at 1 and in sequential order (e.g. 1, 2, 3, etc.).

`ballots.csv`: Contains the data about the ballots and the votes cast by the voters. Each line starts with the ballot number followed by the votes in that ballot (in ascending order **by rank**), in the following format:

ballot#,c_1:1,c_2:2, ...,c_k:k

where every c_i:i combination represents the candidate ID (c_i) to which the voter assigned the rank value i. Note that you must validate each ballot (See "Valid ballots" section).

Output

You must save the results in the current directory in a file named `results.txt` (which your program must create). The results must include:

- Number of ballots received
- Number of blank ballots (no candidates were selected)
- Number of invalid ballots
- Round results. For each eliminated candidate you must store:
  - Number of 1's that the candidate received at the moment of elimination
  - Round in which the candidate was eliminated.
  - Use the following format: "Round <*num*>: <*Candidate name*> was eliminated with <*num*> #1's".
- Winner, in the following format: "Winner: <*Candidate name*> wins with <*num*> #1's".

Sample data

You will be provided with sample input files with the corresponding output file so that you can test your program. Remember that one successful test case does NOT guarantee that your program is correct, but it should at least help you to find some of the most common errors. Additionally, the sample output file will show you **exactly** how your output should be formatted.

Technical details

Ballots

You must create a Ballot class, such that an instance of this class can store all of the information regarding a single ballot. It must contain the ballot number as well as a List to contain the votes that were cast in that ballot (you can use more than one List if you wish, but technically you can do it with only one). How you use the list(s) and what you store in them is up to you. Your Ballot class must contain *at least* the following methods:

```
public int getBallotNum(); // returns the ballot number
public int getRankByCandidate(int candidateId); // rank for that candidate
public int getCandidateByRank(int rank); // candidate with that rank
public boolean eliminate(int candidateId); // eliminates a candidate
```

Create a constructor such that you could pass the line of text from the input file and then the constructor would take care of storing the information accordingly. You may create as many additional methods as you deem necessary and/or helpful.

Storing the ballots

The ballots must be stored in *n* Sets, where *n* is the number of candidates. The best way to collect and manipulate this collection of sets is, of course, another List. The set S_i must contain all of the ballots where candidate with ID *i*+1 has a rank value of 1. As you read the input file, you must create Ballot objects and store them in the appropriate set depending on which candidate has a rank value of 1 in that ballot (every valid ballot must have such a candidate). Note that empty ballots and invalid ballots should not be stored, only accounted for.

Removing candidates (Design vs. Implementation)

Although in the explanation of the voting method a candidate disappears from the ballots once he/she is eliminated, that would require going through **all** of the ballots and performing a search for that candidate. The reason we are using separate sets for the candidates is to avoid that overhead. Instead, we're only going to modify the ballots where the candidate being removed has a rank value of 1 (the "top candidate"), so that we only have to loop over that particular set. The only downside to this strategy is that when we remove the top candidate from a ballot, it's possible that the new top candidate might have also been previously eliminated and we hadn't removed him/her from that ballot. You must create another Set for the eliminated candidates, so that when you are modifying a ballot, you keep removing the top candidate in that ballot until you find one that is still active (or the ballot is empty).

Using the same example as before, suppose we start as follows:

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	4	4	5	1	1	4	1	3	4
Juan	2	3	2	1	2	5	2	2	4	2
Lola	3	1	3	3	4	3	5	4	2	3
Pardeep	4	5	1	2	3	2	3	5	5	1
Pucho	5	2	5	4	5	4	1	3	1	5

Once we eliminate Lola, we have:

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	3	4	5	1	1	4	1	3	4
Juan	2	2	2	1	2	5	2	2	4	2
Lola	3		3	3	4	3	5	4	2	3

eliminated = { Lola }

Pardeep	4	4	1	2	3	2	3	5	5	1
Pucho	5	1	5	4	5	4	1	3	1	5

Note that we only had to modify one ballot (b2)!
In this round, Juan gets eliminated, but fortunately we only have to modify b4.

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	3	4	4	1	1	4	1	3	4
Juan	2	2	2		2	5	2	2	4	2
Lola	3		3	2	4	3	5	4	2	3
Pardeep	4	4	1	1	3	2	3	5	5	1
Pucho	5	1	5	3	5	4	1	3	1	5

eliminated = { Lola, Juan }

Now Pucho gets eliminated. When updating b2, we notice that Juan moves up to be the top candidate, but Juan was already eliminated, so we modify the ballot again. The same thing happens with b7 and b9, and we end up with:

Candidate	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
Pepe	1	1	4	4	1	1	2	1	1	4
Juan	2		2		2	5		2	2	2
Lola	3		3	2	4	3	3	4		3
Pardeep	4	2	1	1	3	2	1	5	3	1
Pucho	5		5	3	5	4		3		5

eliminated = { Lola, Juan, Pucho }

And Pepe wins (again).

Note that *half of the ballots were never modified*, thanks to the strategy of separating the ballots by top candidate and only modifying when necessary. Otherwise, we would have had to modify all 10 ballots in every single round. Technically we now loop over the *eliminated* set instead, but that is a much smaller set. Another advantage is that now it's really easy to find out at any point the number of ballots in which a candidate has a rank value of 1 (think about it). However, we still loop over all ballots to count the amount of votes each candidate has for other ranks, but this is no different than before.

Election class

Your main program must be named Election.java, and it must implement all of the aforementioned logic using the Ballot class you created. *You must use only locally defined data structures; you cannot use Java's built-in List/Set structures.* However, you can be creative as to how you use them (for example, you can use a LinkedList as a Set, if you think that might be convenient).

Documentation & Comments

You must properly document your code using the Javadoc format. Include at the top of the class some comments explaining the structure of your class, particularly what type of list(s) you are using and why. For example, "An ArrayList was used to store the votes, where the top candidate is the last element of the list, so that when that top candidate is removed…". Different people will have different implementations, so it's important that your code is well documented and commented so that we may understand your intention. You will lose points if you don't provide comments or if we deem they are inadequate.

Submission

The final date to submit your program will be **Thursday, March 12, 2020** at 11:59pm. The name of your zip file must follow one of the two following formats: `P1_4020_nnnnnnnn_192.zip` (for those in CIIC4020) or `P1_4035_nnnnnnnn_192.zip` (for those in ICOM4035), where nnnnnnnn should be replaced with your student id number.

Academic Integrity

Do NOT share your code! You may discuss design/implementation strategies, but *if we find projects that are too similar for it to be a coincidence, all parties involved will receive a grade of 0.* Don't cheat yourself out of a learning experience; seek our help if you need it.

Final comments

The specifications of a project are the first, and arguably the most important, part of a software development project. Therefore, it's crucial that you read these specifications thoroughly so that you understand what is being asked of you. These are skills that you will need to succeed in your professional career, so it's imperative that you start applying and improving them now. If your program runs successfully, but does not adhere to the specifications, it is of no use. *Before you submit your project, review these specifications one last time and make sure you meet all of the requirements that have been imposed.*

If your code does not compile properly, your grade will be 0, NO EXCEPTIONS!