Project 7.6 Voting methods

Although most of us are used to simple plurality-based elections in which the candidate with the most votes wins, there are other voting methods that have been proven to be fairer according to various criteria. In this project, we will investigate two other such voting methods. For all of the parts of the project, we will assume that there are four candidates named Amelia, Beth, Caroline, and David (abbreviated A, B, C, and D).

Part 1: Get the data

The voting results for an election are stored in a data file containing one ballot per line. Each ballot consists of one voter's ranking of the candidates. For example, a small file might look like:

```
B A D C
D B A C
B A C D
```

To begin, write a function

```
readVotes(fileName)
```

that returns these voting results as a list containing one list for each ballot. For example, the file above should be stored in a list that looks like this:

```
[['B', 'A', 'D', 'C'], ['D', 'B', 'A', 'C'], ['A', 'B', 'C', 'D']]
```

There are three sample voting result files on the book website. Also feel free to create your own.

Part 2: Plurality voting

First, we will implement basic plurality voting. Write a function

```
plurality(ballots)
```

that prints the winner (or winners if there is a tie) of the election based on a plurality count. The parameter of the function is a list of ballots like that returned by the `readVotes` function. Your function should first iterate over all of the ballots and count the number of first-place votes won by each candidate. Store these votes in a dictionary containing one entry for each candidate. To break the problem into more manageable pieces, write a "helper function"

```
printWinners(points)
```

that determines the winner (or winners if there is a tie) based on this dictionary (named `points`), and then prints the outcome. Call this function from your `plurality` function.

Part 3: Borda count

Next, we will implement a vote counting system known as a Borda count, named after Jean-Charles de Borda, a mathematician and political scientist who lived in 18th century France. For each ballot in a Borda count, a candidate receives a number

of points equal to the number of lower-ranked candidates on the ballot. In other words, with four candidates, the first place candidate on each ballot is awarded three points, the second place candidate is awarded two points, the third place candidate is awarded one point, and the fourth place candidate receives no points. In the example ballot above, candidate B is the winner because candidate A receives $2 + 1 + 2 = 5$ points, candidate B receives $3 + 2 + 3 = 8$ points, candidate C receives $0 + 0 + 1 = 1$ points, and candidate D receives $1 + 3 + 0 = 4$ points. Note that, like a plurality count, it is possible to have a tie with a Borda count.

Write a function

`borda(ballots)`

that prints the winner (or winners if there is a tie) of the election based on a Borda count. Like the `plurality` function, this function should first iterate over all of the ballots and count the number of points won by each candidate. To make this more manageable, write another "helper function" to call from within your loop named

`processBallot(points, ballot)`

that processes each individual ballot and adds the appropriate points to the dictionary of accumulated points named `points`. Once all of the points have been accumulated, call the `printWinners` above to determine the winner(s) and print the outcome.

Part 4: Condorcet voting

Marie Jean Antoine Nicolas de Caritat, Marquis de Condorcet was another mathematician and political scientist who lived about the same time as Borda. Condorcet proposed a voting method that he considered to be superior to the Borda count. In this method, every candidate participates in a virtual head-to-head election between herself and every other candidate. For each ballot, the candidate who is ranked higher wins. If a candidate wins every one of these head-to-head contests, she is determined to be the Condorcet winner. Although this method favors the candidate who is most highly rated by the majority of voters, it is also possible for there to be no winner.

Write a function

`condorcet(ballots)`

that prints the Condorcet winner of the election or indicates that there is none. (If there is a winner, there can only be one.)

Suppose that the list of candidates is assigned to `candidates`. (Think about how you can get this list.) To simulate all head-to-head contests between one candidate named `candidate1` and all of the rest, we can use the following `for` loop:

```
for candidate2 in candidates:
    if candidate2 != candidate1:
        # head-to-head between candidate1 and candidate2 here
```

This loop iterates over all of the candidates and sets up a head-to-head contest between each one and `candidate1`, as long as they are not the same candidate.

Question 7.6.1 *How can we now use this loop to generate contests between all pairs of candidates?*

To generate all of the contests with all possible values of `candidate1`, we can nest this `for` loop in the body of another `for` loop that also iterates over all of the candidates, but assigns them to `candidate1` instead:

```
for candidate1 in candidates:
    for candidate2 in candidates:
        if candidate2 != candidate1:
            # head-to-head between candidate1 and candidate2 here
```

Question 7.6.2 *This nested* `for` *loop actually generates too many pairs of candidates. Can you see why?*

To simplify the body of the nested loop (where the comment is currently), write another "helper function"

```
head2head(ballots, candidate1, candidate2)
```

that returns the candidate that wins in a head-to-head vote between `candidate1` and `candidate2`, or `None` is there is a tie. Your `condorcet` function should call this function for every pair of different candidates. For each candidate, keep track of the number of head-to-head wins in a dictionary with one entry per candidate.

Question 7.6.3 *The most straightforward algorithm to decide whether* `candidate1` *beats* `candidate2` *on a particular ballot iterates over all of the candidates on the ballot. Can you think of a way to reorganize the ballot data before calling* `head2head` *so that the* `head2head` *function can decide who wins each ballot in only one step instead?*

Part 5: Compare the three methods

Execute your three functions on each of the three data files on the book website and compare the results.