Extended Exercise: The Mathematics of Elections

A Supplement to "How to Design Programs"

Acknowledgments

Thanks to Stephen Bloch (Adelphi) and Terry Butler (Brigham Young University) for feedback on this exercise set. Professor Bloch recommends Don Saari's book "Chaotic Elections!: A Mathematician Looks at Voting" for additional insight into this topic. He also suggests you look up the Condorcet method for counting votes.

1 Counting the Votes

Source: Wall Street Journal, Friday March 14, 2003, page B1, column 1

Counting democratic elections poses a non-trivial problem. If we allow a voter or, more likely, a group of voters (say a state) to pick a sequence of choices, there are many strategies to evaluate these sub-elections. Here are three:

1. the winner-takes-all strategy says that the winner of the most sub-elections wins the overall election;

2. the approval-rating strategy gives the first and second place finishers each one point, and the candidate with the most points wins the overall election;

3. the points-per-place strategy allocates 3, 2, and 1 point to the top three finishers, respectively. Again, the candidate with the most points wins the overall election.

Different countries have come up with additional strategies with good justification for each of them.

Not surprisingly, different strategies produce different winners. That is, given the same series of election results, the winner depends on the evaluation strategy. Indeed, any of the above strategies may produce more than one winner, so we also need a tie-breaking mechanism in the end.

1.1 The Basic Problem

PREREQUISITE: 12.3. Composing Functions, Revisited Again

Exercise 1.1.1 Make up examples of elections. Consider the following scenario. Your class wants to organize a little reception for your teacher. (Yes, the teacher did a great job, teaching you things and doing so in a nice way.) You would like to cater at least two choices of food. You ask all your classmates to rank the following foods in order of preference: chicken, steak, and (triple-flavored) tofu. Each of your classmates turns in a ranked list of these foods and you need to figure out the winner. Use all three strategies to count (imaginary) results from this food-vote. ∎

Exercise 1.1.2 Explain how real-world election laws incorporate elements from each of these strategies. Hint: Strategies 1 and 3 occur in the US in modified form. ∎

Exercise 1.1.3 Association lists are an important element of programming that helps with many tasks. It typically associates symbols or strings with numbers or other values.

A *AssocList* is either

1. empty
2. (cons (cons *String natural-number*) *AssocList*)

Develop the function *bump*. It consumes an association list *al*, a string *name*, and a natural number *i*. It produces an association list. The given and the produced association lists are alike except for *name*. If *name* is already associated with a number *j* in *al*, then the new list associates *name* with (+ *i* *j*); otherwise it associates *name* with just *i*. ∎

Exercise 1.1.4 For each of the three evaluation strategies, develop a function that computes the winner. Each function consumes a list of election results and produces a sorted list of those candidates who won the election. An election result specifies the first three candidates as strings. Think of each election result as the first three in a state's election or the ranking of three foods that the students in your school may select for a school party.

The functions must accommodate write-in candidates. That is, every voter may add a name. The functions cannot assume that there is a fixed set of candidates. Note that this does *not* make the problem more difficult.

Hints:

1. The evaluation of an election consists of two tasks: tabulating the votes and determining the winner(s).
2. Use an association list (ex. 1.1.3) to connect the two tasks.
3. Sort the list of winners with (quicksort . . . string<=?). ∎

Exercise 1.1.5 Find a list of subelection results for which the three strategies produce pairwise distinct winners. ∎

1.2 Editing Programs

PREREQUISITE: 21.2. Abstracting From Examples

Exercise 1.2.1 Compare the functions for tabulating the votes from elections. They are structurally alike except for the treatment of each election, which is determined by the election strategy. Abstract the functions that tabulate the votes over the strategy. Represent the strategies as functions with this contract:

```
;; strategy : AssocList Election → AssocList
;; to count the votes in the election-result
(define (strategy results election-result) . . . )
```

The class of *Election*s represents the results of a single sub-election according to the solution of exercise 1.1.5.

Create a single function *eval-elections* that computes the winners of an election according to all strategies that satisfy the above contracts. ∎

Exercise 1.2.2 Use the functions from figure 57 (page 313) to simplify the function from exercise 1.2.1, which tabulates the votes according to some given strategy. ∎

Exercise 1.2.3 Use the functions from figure 57 (page 313) to simplify the function from exercise 1.1.5 that determines the winner from the tabulated results. ∎