Haskell Programming Assignment

Functional Programming

March 9, 2020

Contents

1 Introduction

Your Functional Programming module marks are made up of 50% Final Exam and 50% Continuous Assessment. The continuous assessment is made up of

- In-class lab test (Week 9) - 5%
- Programming Assignment (due week 13) - 40%
- In-class lab test - 5% - more details later

Your Programming assignment can be one of

1. Your own choice.. You will need to discuss this with me and could be, for instance, part of your final year project. (If you choose to do this, and this work is evaluated towards this module, this will be mentioned in your Final Year Project final report as the work cannot be double marked)
2. The (default) assignment as specified in the remainder of this document.

2 Single Transferable Vote

The single transferable vote (STV) is a voting system designed to achieve proportional representation through ranked voting in multi-seat constituencies. You can see a good video here. You can look up lots on STV but from the point of view of the assignment, we are interested in the preparation and counting of the votes.

2.1 Rules for counting votes

Specifically the rules for counting the votes for our election are:

1. All valid papers are grouped by first preference votes and candidates are ordered in descending order of first preferences. Each vote is given a weight of 1000.

2. The quota is calculated

$$quota = (\frac{number\ of\ valid\ votes}{number\ of\ seats\ +\ 1})\ +\ 1$$

3. If, at the end of any count, a candidate has a total weight of votes greater than the quota, then this candidate should be elected.

4. Calculate the total weight of transferable votes. If the total weight of transferable votes is greater than the surplus, then transfer each of the votes (to the continuing candidate indicated as the next available preference) with a reduced weight,

$$newweight = old\ weight * (\frac{surplus}{total\ weight\ of\ transferable\ papers})$$

   If the total weigh of transferable votes is less than or equal to the surplus, transfer all votes in the bundle, leaving the weights unchanged.

5. If, at the end of any count, no candidate has reached the quota and there are more continuing candidates than vacancies, a candidate must be eliminated. Working backwards from the candidates with the lowest weight, distribute the (eliminated) candidate's votes, leaving the weights unchanged.

6. **STOP** when the number of elected candidates + then number of candidates remaining (uneliminated) = the number of seats. At this point the remaining (unelected and uneliminated) candidates are deemed to be elected 'without reaching the quota'.

2.2 Dealing with Invalid or Partially Invalid votes

A valid vote is a ordered list of preferences, starting at one, with each preference contiguous and with no duplicates. So the following are not allowed:

1. Two candidates with the same preference (e,g vote → 1,2,3,3,4 - this should be reduced to 1,2)
2. A break in preferences, e.g. vote → 1,2,4,5 (this should be reduced to 1,2)

Note that a 'cleaned' vote using these rules may result in a reduction to an empty vote. A fully empty vote is called a 'spoiled vote'

3 Assignment Details

You are asked to

1. Take in data of the form as per this file (votes.csv) and clean and rearrange the data "as necessary".
2. Write a Haskell program to count votes in an election (e.g. the sample data) using
   - Alternative Vote (as per Hutton). The output should be, (at least), the winning candidate.
   - Single Transferable Vote (as per Section 2). The output should be, (at least), the list of elected candidates.
   - As we introduce new concepts, these may be included, but the main bulk of the work is defined in this document.

4 Some notes on the program using the STV)

1. Note that the input from the user will include both the votes (e.g. the sample data given) and the number of candidates to be elected.

5 Indicative Marking Scheme

Functionality	%
Cleaned Data coming from file	20
Alternative Vote working (winner correct)	5
STV - Elected Candidates Corrrect	50
Extra Information	
More comprehensive information per count	10
Using a package appropriately	5C
Miscellaneous extras (aka very impressive stuff!)	10

Quality of Haskell Code	%
Haskell Style	50
Elegance including naming	30
Use of appropriate libraries, structures	20

The final weighting between functionality and quality of Haskell Code is 70 (functionality) 30 (Quality of Haskell code).
It is, of course, necessary that the work submitted by you is your own. You may be asked to explain your work and apart from Prelude functions, and clearly imported libraries, any other work not entirely your own must be clearly referenced. I may interview any student to check that the code is their own (if not fully referenced) and that you fully understand the code and how it works.

The interview mark will be a multiplier with the Functionality/Quality Mark. So, if your Functionality Mark = 70% and Haskell Code Quality = 70% this gives you an average of 70%. If your interview score was %50, then your overall mark would be 70 * 50 / 100 = 35%.

This assignment should be submitted through Moodle by Week 13.