COMP1029 Case Study 4
The Winner Takes It All

Abstract

The United Kingdom recently held a nationwide referendum to decide whether to replace the simple *first past the post* system for general elections with the more refined *alternative vote* system. The decision was strongly in favour of retaining the current system.

Your script should begin with the following declarations

```
import Data.List (sort)
type Party = String
type Ballot = [Party]
```

The first line imports the library function

$$sort \;\; :: \;\; Ord \; a \Rightarrow [a] \rightarrow [a]$$

that sorts a list of values, provided that the values have an ordering. The remaining lines declare a party name as a string, and a ballot paper as a list of preferences (first choice, second choice, etc.)

First Past The Post

In this voting system, each person has one vote, and the party with the largest number of votes is the winner. We will build up to implementing this system in a number of steps.

• **Exercise:** Define a function

$$count \;\; :: \;\; Eq \; a \Rightarrow a \rightarrow [a] \rightarrow Int$$

that counts the number of occurrences of a value in a list, provided that the values support equality. For example,

```
> count 'a' "ababca"
3
```

• **Exercise:** Define a function

$$rmdups \;\; :: \;\; Eq \; a \Rightarrow [a] \rightarrow [a]$$

that removes duplicates from a list. For example,

```
> rmdups "ababca"
"abc"
```

• **Exercise:** Using *count* and *rmdups*, define a function

$$frequency \;\; :: \;\; Eq \; a \Rightarrow [a] \rightarrow [(Int, a)]$$

that counts how many times each distinct value in a list occurs in that list. For example,

```
> frequency "ababca"
[(3,'a'),(2,'b'),(1,'c')]
```

• **Exercise:** Using *sort* and *frequency*, define a function

$$results \;\; :: \;\; [Party] \rightarrow [(Int, Party)]$$

that takes a list of all the votes that were cast (one per person) and returns the results of the election in increasing order of the number of votes. For example, if we define

```
votes :: [Party]
votes = ["Red","Blue","Green","Blue","Blue","Red"]
```

then

```
> results votes
[(1,"Green"),(2,"Red"),(3,"Blue")]
```

If more than one party has the same number of votes, they should be returned in alphabetical order. *Hint*: try some experiments to see what *sort* does on lists of pairs.

• **Exercise:** Using *results*, define a function

$$winner \;\; :: \;\; [Party] \rightarrow Party$$

that takes a list of votes and returns the winner under the first past the post scheme. For example,

```
> winner votes
"Blue"
```

Alternative Vote

In this voting system, each person can vote for as many or as few parties as they like, listing them in preference order on their ballot paper (first choice, second choice, etc.) To decide which party wins, we begin by eliminating the party with the smallest number of first preference votes, and then repeat this process until only one party remains. This party is the winner.

• **Exercise:** Define a function

$$rmempty \quad :: \quad Eq\ a \Rightarrow [[a]] \rightarrow [[a]]$$

that removes all occurrences of the empty list from within a list of lists. For example,

```
> rmempty ["abc","","bc",""]
["abc","bc"]
```

• **Exercise:** Define a function

$$remove \quad :: \quad Eq\ a \Rightarrow a \rightarrow [[a]] \rightarrow [[a]]$$

that removes all occurrences of a given value from within a list of lists. For example,

```
> remove 'a' ["abc","bc","aa"]
["bc","bc",""]
```

• **Exercise:** Using *results* from the previous page, define a function

$$rank \;::\; [Ballot] \to [Party]$$

that takes a list of all the ballot papers that were submitted (one per person), selects the first preference vote on each ballot, and returns the result of a first past the post election based upon these votes, in increasing order of the number of votes. You may assume that each ballot is non-empty, i.e. contains at least one vote. For example, if we define:

$$\begin{array}{lcl} ballots & :: & [Ballot] \\ ballots & = & [b1, b2, b3, b4, b5, b6] \end{array}$$

$$\begin{array}{l} b1 = \texttt{["Blue","Green"]} \\ b2 = \texttt{["Green","Blue","Red"]} \\ b3 = \texttt{["Blue"]} \\ b4 = \texttt{["Red","Green"]} \\ b5 = \texttt{["Blue","Red","Green"]} \\ b6 = \texttt{["Green","Red"]} \end{array}$$

then

```
> rank ballots
["Red","Green","Blue"]
```

because in terms of first preferences, `"Red"` has one vote, `"Green"` has two, and `"Blue"` has three.

• **Exercise:** Using *rmempty*, *remove*, and *rank*, define a function

$$election \;::\; [Ballot] \to Party$$

that takes a list of ballot papers and returns the winner under the alternative vote scheme. For example,

```
> election ballots
"Green"
```

Hint: make sure to remove empty ballots using *rmempty* before calling the *rank* function. You may want to use `case..of` in the function.