Department of Computing and Information Systems

COMP10002 Foundations of Algorithms
Semester 2, 2013
Assignment 2

Learning Outcomes

In this project you will demonstrate your understanding of dynamic memory and linked data structures. You will also extend your skills in terms of program design, testing, and debugging.

The Story...

Australia just went through a Federal election. A distinctive component of Australia's electoral process is the use of *preferential voting*, whereby voters rank candidates in order of preference, from most desired (with number 1) through to least desired (assigned rank m, where m is the number of candidates in that electorate). For example, one ballot paper might be represented as

```
A   B   C   D   E
1   4   2   3   5
```

where each number represents the ordering that the voter places on the corresponding candidate. That is, if we think of the candidates as being the "columns", and labeled from `A` to `E`, then this voter wants candidate `A` to win the electorate, and if that can't happen, they next prefer candidate `C`, then `D`, then `B`, and then finally, their last choice is candidate `E`. Each voter in the electorate submits a vote with their own ordering. So, if there are seven voters in total in the electorate, their votes might be represented as:

```
1   4   2   3   5
3   4   2   1   5
1   2   4   3   5
4   1   2   5   3
4   3   1   2   5
2   3   5   1   4
2   3   1   5   4
```

To compute the winning candidate in each electorate, the votes are tabulated according to their first preferences, yielding this arrangement of candidates and voter numbers:

```
A:  1  3
B:  4
C:  5  7
D:  2  6
E:
```

No candidate has more than 50% of the first preferences, and so the next phase of the counting is commenced, with candidates being removed one by one form the bottom of the list until one candidate has a clear majority. Candidate `E` has no first preference votes, and cannot be elected, so they are the first removed from consideration. No votes get redistributed at this point in time.

Next comes `B`, with one first-preference vote. This candidate is also eliminated, and that vote is redistributed according to its second preference, yielding:

```
A:  1  3
C:  4  5  7
D:  2  6
```

There is still no majority, and the process continues. Since A and D are tied with least votes, one of them is chosen at random[1] and removed from contention; all of the votes currently sitting with that candidate are redistributed to whichever of the remaining candidates is listed highest on the corresponding original ballot papers. Supposing that D is the one selected and eliminated, the situation then becomes:

```
A:  1  3  6
C:  2  4  5  7
```

There is now a clear majority, and C is declared elected, with a "two-party preferred" vote of 57.1% (even though they only had 28.6% of the first preferences). On the other hand, if A had been eliminated at the tied step, candidate D would have been elected as a result of the preference flows.

There is a range of on-line information about preferential voting, including:

- `http://en.wikipedia.org/wiki/Instant-runoff_voting`
- `http://www.austlii.edu.au/au/legis/cth/consol_act/cea1918233/s273a.html`.

Input Format

The expected input format for all stages is identical, and consists of a single integer indicating the number of candidates; followed by that many candidates' names, one per line, as single words; followed by an arbitrary number of votes, each of which consists of a row of candidate preferences of the type already discussed. For example,

```
5
Alice
Bob
Catherine
Dilbert
Enrico
1  4  2  3  5
3  4  2  1  5
1  2  4  3  5
4  1  2  5  3
4  3  1  2  5
2  3  5  1  4
2  3  1  5  4
```

is an input file that describes the situation in the example, with names used instead of A to E.

The *only* assumption that you may make about the input data is that the length of each candidate's name will be thirty characters long, or less. In particular, you may *not* embed any assumptions about the maximum number of candidates, or the maximum number of votes being cast, into your program – the data structures for these must be dynamic.

You *may* assume that the input data is completely correct, and that every voter has lodged a full set of valid preferences.

[1] In Australia, when votes are tied and action is required, the Returning Officer for the electorate makes a decision by exercising a casting vote. In this project we will regard that as being a "random" outcome, and simulate it via a call to `rand()`.

Stage 1 – Getting Started (marks up to 8/15)

Write a program that reads data in the format specified above (reading from `stdin`, with no prompt required) and generates a tabulation of the first preferences. Expected output on the example data is:

```
5 candidates, 7 votes

Round 1 ...
        Alice     :    2 votes, 28.6%
        Bob       :    1 votes, 14.3%
        Catherine :    2 votes, 28.6%
        Dilbert   :    2 votes, 28.6%
        Enrico    :    0 votes,  0.0%
```

Stage 2 – Preferential Voting (marks up to 13/15)

Now apply the elimination/preferencing process to that initial arrangement, to generate:

```
Round 2 ...
        Alice     :    2 votes, 28.6%
        Bob       :    1 votes, 14.3%
        Catherine :    2 votes, 28.6%
        Dilbert   :    2 votes, 28.6%

Round 3 ...
        Alice     :    2 votes, 28.6%
        Catherine :    3 votes, 42.9%
        Dilbert   :    2 votes, 28.6%
        2 equal last candidates, outcome determined randomly

Round 4 ...
        Alice     :    3 votes, 42.9%
        Catherine :    4 votes, 57.1%   *** elected
```

You may use whatever techniques you wish, including linked lists and/or resizing arrays (you probably don't need trees for this one), and may make use of code from the textbook, provided it is properly attributed.

You should aim for running time that is $O(mn)$, where m is the number of candidates, and n the number of votes, and it is assumed that $m > n$ (after all, every candidate will vote). That is, your running time should be linear in the size of the input. But note that any *correct* solution will get the majority of the "execution" component of the marks, even if it does not achieve $O(mn)$ time.

In order for the markers to assess whether your solution does indeed require $O(mn)$ time you should ensure that comments in your program clearly explain the computation, and clearly establish the basis for any claimed performance. There will be marks allocated to the clarity of your argument about running time, as well as to the simple fact of having achieved $O(mn)$ performance.

In both Stage 1 and Stage 2 the output in each round of voting may be in any convenient order – the examples here use the order the candidate names were input.

Stage 3 – Sorting (marks up to 15/15)

Now alter your program so that each block of the output is in decreasing vote count order, with ties resolved alphabetically. For example, the first output block would become

```
Round 1 ...
        Alice     :    2 votes, 28.6%
        Catherine :    2 votes, 28.6%
        Dilbert   :    2 votes, 28.6%
        Bob       :    1 votes, 14.3%
        Enrico    :    0 votes,  0.0%
```

Note that the addition of the sorting step means that your running time may no longer be in $O(mn)$; no need to worry about that. You may use any sorting algorithm you wish, including calling the system `qsort()` if you want to.

Stage 4 – Complicated!! (marks up to 15/15)

Keep track of every point at which a tie-breaking decision is made during the preferencing process. Rather than randomly choosing one of the alternatives for elimination, and just going on from there, record what eventuates with *every* alternative choice, *every* time there is a tie. Assign overall probabilities of winning to candidates, based on the outcomes, and print out (as an additional output after the standard presentation shown above) a final table of probabilities of winners. In the example, `Catherine` is the winner with 100% probability, because regardless of what happens at the tie-breaking point, she is the eventual winner. In other situations there might be more than one candidate with non-zero probability of winning. You will need to track possible combination of tie-breaking choices recursively, splitting the residual probability at each decision point according to the number of equal choices.

The boring stuff...

This project is worth 15% of your final mark. You don't have to do Stage 4 in order to get 15/15, and really truly should only tackle it if you really truly want to have an exciting time doing some fun programming.

You need to submit your program for assessment; detailed instructions on how to do that will be posted on the LMS once submissions are opened. You will need to log in to a Unix server and submit your files to a software system known as `submit`. You can (and should) use `submit` **both early and often** – to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. Only the last submission that you make before the deadline will be marked.

You may discuss your design/plans during your workshop, and with others in the class, but what gets typed into your program must be individual work, not from anyone else. So, do **not** give hard copy or soft copy of your work to anyone else; do **not** "lend" your memory stick to others; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say "**are you out of your mind?**" if they ask for a copy of, or to see, your program, pointing out that your refusal, and their acceptance of it, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode.*

Deadline: Programs not submitted by **6:00pm on Friday 18 October** will lose penalty marks at the rate of two marks per day or part day late. Students seeking extensions for medical or other "outside my control" reasons should email Alistair, `ammoffat@unimelb.edu.au`.

And remember, algorithms are fun!