DSA Assignment Documentation

Prassana Kamalakannan 19097457

Date Modified: 29/10/2018

Class Descriptions

The Main class contains the main method, which calls the mainMenu() method, and this initiates the whole program.

The Menu class was created to encapsulate all the methods that makes up the menu. The mainMenu() first gets an array of lines of nominee details from the FileIO class's read method, and then outputs a a menu showing the list of options the user can choose from. A switch statement is used to for the for the menu. If the user chooses 1 then the filterNominees() method is called, if 2 then nomineeSearch is called, if 3 then listByMargin() is called and if 4 then The ItineraryDisplay method is called. This menu will continue to loop until the user enters 5 which is the option to quit. A switch statement is used for the menu because it is faster than using if-else statements.

The filterNominees() method imports a a string array where each element is a line containing the details of a nominee. Each of the details is split by a comma using SplitFile's splitCandidateFile() method. After the lines have been split, each element of the line is used to create an object called nominee. Each nominee object, created, is then added to a linked list. This linkedList is then exported from splitCandidateFile() to filterNominees. A linked list iterator is used to add each nominee object into an array which is then imported into a sort method. After the sorted array had been returned, each element of the array is then added to another linked list, now in sorted order and is then filtered. Another menu is then displayed, this time showing a list of options on how you want to filter the list. After choosing an option the list will filter to show only the objects that contain the attribute or attributes you want enter.

The nomineeSearch() method gets the user to enter in the surname or part of a surname they want to find and displays the details of the nominees that start with the search term after filtering just like in the filterNominees. First the method takes in a list of nominees, that have been read in from the same file that filterNominees() uses, then it prompts the user to enter a search which is then matched with the surname of each nominee to see if any of them starts with the entered substring. The filtering of objects work in the same way that filterNominees() does, but this time with the added condition that the nominee's surname must start with the entered substring. The added condition is the reason why nomineeSearch uses it's own filter method rather than calling a generic filter method.

The listByMargin() method reads in the lines from all House State First Pref files and uses each line to create a FirstPref object. These objects are then added into a linked list. This way all first prefs can be used by accessing their getters. It was done this way because data from mutiple files had to be used. The user can then enter the party they want to see and then, using each linked list element's getters, the margin is calculated and then displayed for that particular party.

The Sorts class contains methods on how you want to sort the list of nominees. It imports an array of Nominee Objects. A menu is then displayed to the user, giving them options on what attribute they want to sort by. If option surname is chosen then, using a switch statement, sortBySurname() is called. The sortBySurname method, compares the surnames of each nominee object, lexicographically, and uses a sort algorithm on the imported array based on the comparison, this is the same with the other sortBy methods as well. The bubble sort algorithm was used for this assignment because, although it has a worst case time complexity of O(n^2), it also has the best case complexity of O(n).

The SplitFile class holds methods that split the read in lines depending on their format and attributes. The splitCandidateFile() method splits the lines that have been read in from the HouseCandidates csv file. Each line is usually split on the comma but for some of the lines, one of

the attributes, which is the party name, has a comma in the middle of it, so simply splitting on comma wouldn't work. However, when there is a comma in the middle of a party name, that name is also surrounded by double apostrophes. This allows me to split on the apostrophes first, which creates three separate arrays which can then each be split on the comma. This gives us each individual attribute of a nominee which can then be used to te create a nominee object. Each nominee object is then added to a linked list which is then exported to be used elsewhere.

The splitForMargin() method imports the array of lines,read in from the FirstPref files, and splits them into their individual elements. The same issue with the comma, that was present in the previous method, is present here as well. Some lines have a comma in the middle of the party name which doesn't make it possible to simply split on the commas. If the party name has a comma in it then it also has double apotrophes around it. The lines, are then, split on the apostrophes first to get three separate arrays and then each array is split on the comma to get the attributes of each first pref object. Once again each object is added to a linked list and then returned to be used by the listByMargin() method in the Menu class.

The ReadMarginFile contains only one method, it imports a file name and calls the readFile() method which returns the lines of the file. The string array of lines was then imported into the splitForMargin() method in the SplitFile class to get a linked list of objects that each represents a FirstPref. This linked list is then exported to be used in the listByMargin() method in the Menu class.

The DSAGraph class contains many methods and private inner classes which come together to define a graph data structure. The graph is implented as an adjacency list. The private inner node class contains a label, a value, links to other nodes and a flag that tells you whether it's been visited or not. The getAdjacent() method returns a linked list of nodes which are adjacent to it. The addEdge() method iterates through the list of adjacent nodes and if the node you want add already exists, the methods will do nothing, otherwise it will insert as the last element of the list of adjacent nodes.

The DSAGraph class itself has a linked list of all the nodes it contains. The addVertex() method imports a label and a value and creates a node from that, and then inserts that node into the linked list. The addEdge() method imports two DSAGraphNode objects. It then iterates through both of their lists of adjacent nodes and if they both exist in each other's adjacent links then their addEdge methods are called. This creates an implicit edge between two vertices.

Justifications

The Nominee class is a basic container class which contains methods and class fields that defines a Nominee object. The reason for creating this class was so that it would be easier to use a nominees properties which wre required for filtering and searching. This is the same for the creation of the FirstPref class. Although they both have comman class fields, I decided to keep them separate rather than have them both inherit from a super class. The reason for this is because they are completely entities that have no relation to aech other with respect to the code, therefore it was conceptually easier to have these two classes separate.

I wasn't able to complete part 4 of the assignment which was the list itinerary section due to time constraints. If I had the time I would read in the files Airport distances and Electoral distances and create objects from both of them. Then I would use a the DSAGraph class, that has and inner edge to output the itinerary. The Airport and Elctoral distances would be the represented as by the nodes and the distances would be represented by the edges. The reason for using a graph is because it is

good for modelling real life situations and has a time complexity of O(number of vertices + number of edges) at most. After the graph for this particular concept is created it can be used to find the shortest paths to the different locations.

For most of the assignment I used linked lists to store objects. The reason for this is because, when inserting into a linked list, the worst case time complexity is O(1) because you don't have to iterate over a linked list if you want which is better than and array's worst case for insertion which is O(n).

I didn't write test harnesses for the DSAStackList and DSAQueueList, because theoretically, If the DFS and BFS work properly then the stack and queue also works properly.