Data Structures

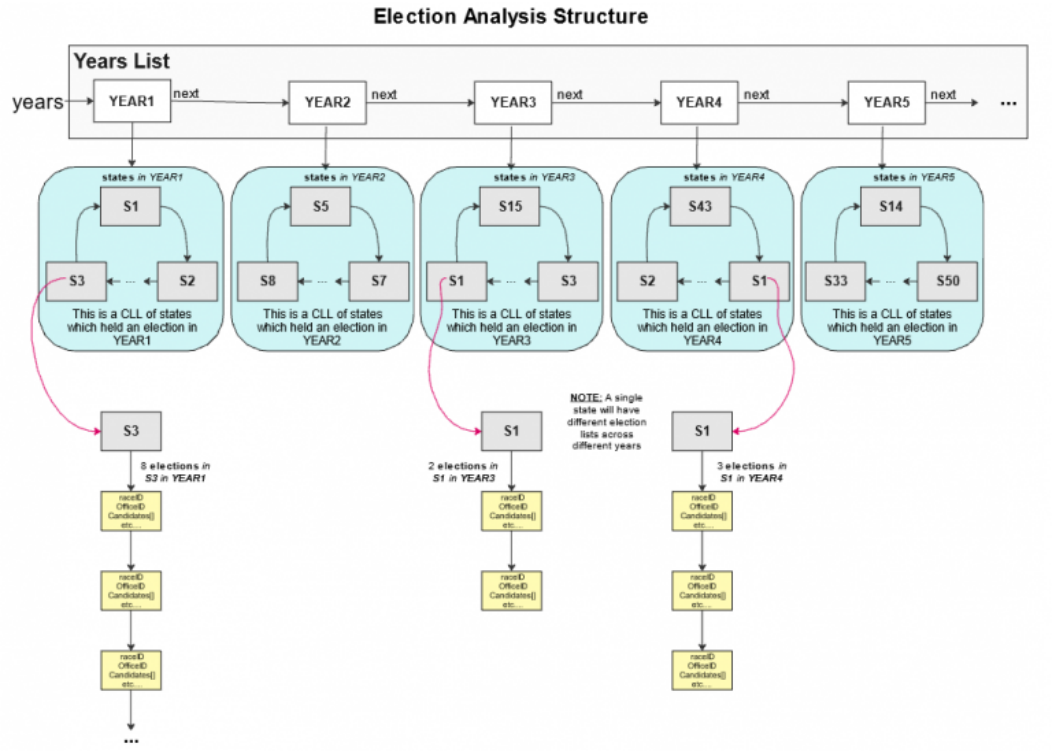
Overview

Voter participation and demographics is an important aspect of any democracy, both at the local and national scales.

In this assignment, you will read real election data for the US House of Representatives and US Senate from a Comma Separated Values (CSV) file. You will then store that data using a nested Linked List/Circularly Linked List structure. Then, we will analyze this data by year, state, and party results to gain insight into voter trends and the change in voter participation over the years.

The ElectionAnalysis class has a Linked List of YearNodes called “years”, which represent the years in which elections have taken place. Each YearNode has a Circularly Linked List of StateNodes called “states”, which represents the states which held elections in that year. Each StateNode has a reference to a Singularly Linked List of ElectionNodes, with each representing a unique election for that State in that year.

After storing this data, we can display it with the Driver to see how voter turnout and party performance have shifted in different states throughout the years.



Implementation

Overview of files provided

- The **ElectionAnalysis** class contains the method stubs you will implement. **This is the file you submit.** This class reads in CSV files for the US Senate/House and creates a list of YearNodes with election lists inside States, as defined above. Then the methods help analyze election results to gather data about elections across states and years. See the method instructions below for more details. DO NOT edit the provided ones or the method signatures of any method.
- The **Driver** class is used to test your methods interactively. Follow the prompts in the terminal to use the driver. This file is provided only to help you test your code. It is not submitted and it is not used to grade your code. **Do not submit to Autolab.**
- Two input files are included: *senate.csv* and *house.csv*. They store information about election in the US House of Representatives and US Senate. You are welcome to create your own input files, as these will not be submitted to Autolab.
    - Additionally, one test input file is included. This has very little Data in it. You can add new lines to this input file to use with JUnit tests in order to test your code.
- The **YearNode** class stores information about a year’s election. It has a field that stores the year and a **StateNode** pointer, which is a circularly linked list representing the states which had an election in that year.
- The **StateNode** class stores information about a state. It has a field for the State’s name as well as an **ElectionNode** field, which is a singularly linked list of elections which occured in that state in that year. Each StateNode has a next field that points to another state node, which should form into a circularly linked list.

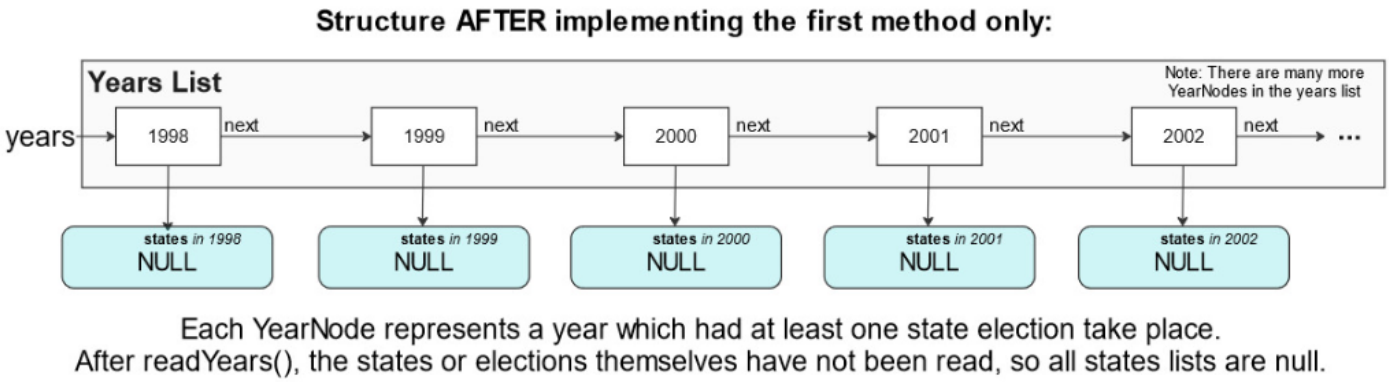
- The **ElectionNode** class stores information about a single election. It has three parallel ArrayLists which store information about a candidate's name, votes, and party. Each **ElectionNode** has a next field that points to another election node, which should create a singularly linked list.

BEFORE YOU BEGIN: Read the constructors and methods for each given **Node** class, to understand how they can be used, and how they connect to each other.

- Need help with understanding constructors, getters, and setters? Read this OOP guide.

readYears(String file)

Here is what the Linked Structure would look like **after** implementing this method:



This method reads a given CSV file of election data using StdIn, and inserts a new YearNode for each unique year found. The "years" field in ElectionAnalysis.java is a reference to the front of this years singly linked list.

To complete this method:

1. Use StdIn.setFile(file) to open the file to be read.
2. Use **StdIn.hasNextLine()** to check if there is another line to be read. You should call this a loop conditions to iterate over the CSVs lines.
3. Read and split the next line:
    - To read and parse a line, you can use String[] line = StdIn.readLine().split(',');
        - This is a String array, where each index will correspond to a piece of information (name, state, year, votes, etc..) about one candidate in a certain election.
        - You can then use this data to build your lists. The correct array indices will be given in the instructions when you need them.
    - Get the year from with int year = Integer.parseInt(split[4])
4. If that year does not have an existing YearNode in the list, create a new YearNode with the given year. Then, insert that YearNode at the END of the list.

Our goal is to insert a new YearNode in the Singly Linked List of years. Remember, ElectionAnalysis.java has a field called 'years' which is a reference to the front of this SLL.

First, check if the years list is empty (aka if the 'years' field is null). If it is, create a new YearNode() and set its year to be the year you read in from the file. Then, set 'years' equals to that YearNode, and you're done.

If the list is not empty, then we want to check if a YearNode containing the read-in year already exists, as we do not want duplicates in the list.

- To traverse the list, declare a YearNode variable ptr and set it to the "years" field. This grabs a secondary reference to the front of the years list.
- Then, while your ptr.getNext() is not equal to null AND your ptr.getYear() is not the read-in year, traverse to the next node.
    - This will traverse you to either the found duplicate node, or to the last node in the list.
- After your traversal loop ends, check the value of ptr:
    - If it's year is equal to the read-in year, continue onto reading the next line of the CSV.
    - If it's year is not equal to the read-in year, then it should be the end of the list. ptr.getNext() should be null.
        - Create a new YearNode with the read-in year
        - Insert that new node at the end of the list, by setting the 'next' field of the last node in the list equal to your new node

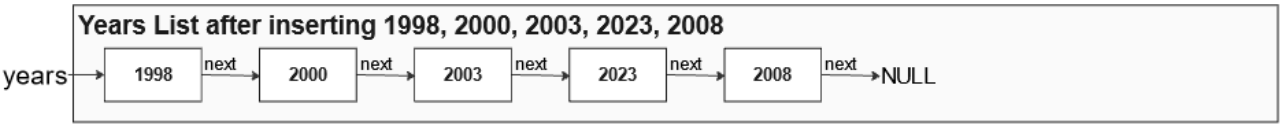
Inserting into the years list:

The Years list will start off empty (as null). When it is, simply insert a new YearNode at the front reference.

If the list is not empty and the year has not already been inserted, insert a new YearNode at the end of the list

If the target year *has* already been inserted, then do not insert a new node.

The Years list will start off empty (as null)



Submit ElectionAnalysis.java with this method completed under Early Submission for extra credit.

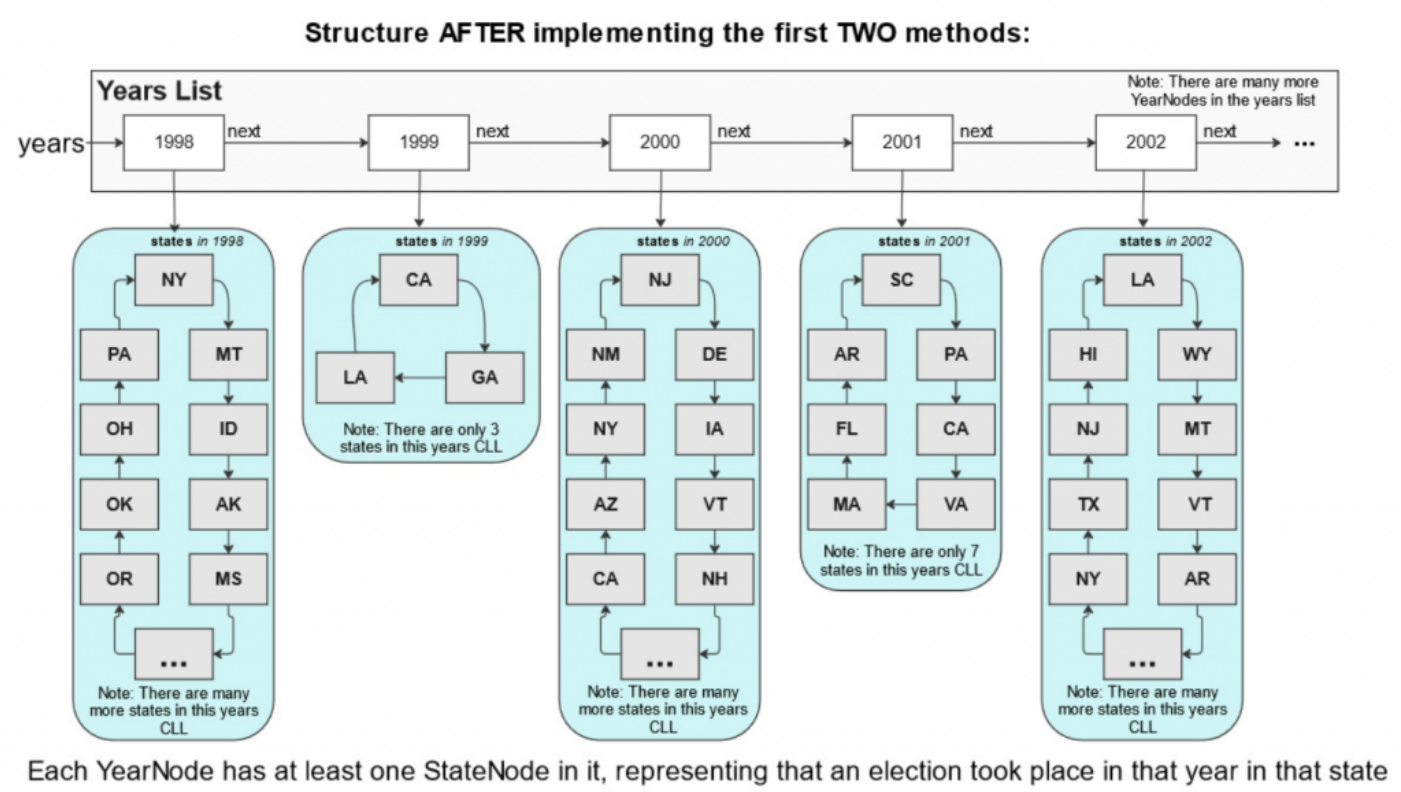
To test this method: use the *ElectionAnalysisTest.java* test class found under the "test" directory in the project folder. The test for this method has been provided for you.

If you've installed TestRunner for Java, you can run this test to evaluate your method's code.

For other methods, you will need to write each test yourself, as well as expand the test input file data.

readStates(String file)

Here is what the structure will look like **after** implementing this method and the previous method.



This method reads over the same CSV file of election data as before.
Instead now, you will read in the Year AND State for each line. You will then **find the correct YearNode**, and insert a new StateNode in it (if one does not exist).

- Read, split, and parse each line as shown in **readYears()** method instructions.

For each line in the given CSV file:
1) Read and split the line (same as in readYears()), and get the state name and year.
– Get the year from the line with **int year = Integer.parseInt(split[4])**
– Get the state name from the line with **String state = split[1]**
3) Search through the "years" linked list for the correct year:
-The correct year is guaranteed to be somewhere in the "years" list, if your **readYears()** method works properly.
4) Inside that YearNode, search through the "states" Circularly Linked List for the above state and if you don't find a corresponding StateNode for the read-in state, insert a new one at the END of the circularly linked list.

Our goal is to insert a new StateNode in the Circularly Linked List of states. Remember, each YearNode in the years list have a field called 'states' which is a reference to the **last** node of this CLL.

First, check if the states list is empty (aka if the 'states' field is null). If it is, create a new StateNode() and set its state name to be the state name you read in from the file. Then, set the containing YearNodes 'states' field to that new StateNode. Finally, set the StateNodes next field to be itself (since it should be circularly linked). Then you're done, so continue onto the next CSV line.

If the CLL is not empty, then we want to check if a StateNode containing the read-in state already exists, as we do not want duplicates in the list.

- To traverse the list, declare a StateNode variable and set it to the year.getStates(), for the corresponding YearNode with the specified year. This grabs a secondary reference to the end of the states list.
- To traverse, instead of using a while loop, you should use a **do-while** loop. This has the loop condition be checked AFTER the code is run.
- After your traversal loop ends, check the value of ptr:
  - If it's year is equal to the read-in year, continue onto reading the next line of the CSV.
  - If it's year is not equal to the read-in year, then it should be the end of the list. ptr.getNext() should be null.
    - Create a new YearNode with the read-in year
    - Insert that new node at the end of the list, by setting the 'next' field of the last node in the list equal to your new node.

readElections(String file)

This method reads over the same CSV files as before. But now we use all the data read, and insert ElectionNodes inside the correct StateNodes. Each line is a candidate in a specific election.
Each StateNode contains an ElectionNode "elections", which is the head of a Singularly Linked List of elections. You will insert these ElectionNodes for each unique raceID, and then insert candidates.

You can use the following to read and fully parse the line:

```
String line = StdIn.readLine();

String[] split = line.split(“,”);

int raceID = Integer.parseInt(split[0]);

String stateName = split[1];

int seat = Integer.parseInt(split[2]);

boolean senate = split[3].equals(“U.S. Senate”);

int year = Integer.parseInt(split[4]);

String canName = split[5];

String party = split[6];

int votes = Integer.parseInt(split[7]);

boolean winner = split[8].toLowerCase().equals(“true”);
```

To complete this method:
1) Search the “years” list for the read-in year.
2) IN that year, search that YearNode’s “states” CLL for the correct state.

- Reference the previous methods for the above steps.

3) In that state, search the state’s “elections” ElectionNode SLL for the read-in election’s raceID.
a) If we find it:
– Use election.getCandidates().contains(canName)) to check if a candidate has already been inserted. If it has, modify that candidate with election.modifyCandidate(canName, votes, party).
– These are stored in three parallel ArrayLists, meaning that each index of the three ArrayLists corresponds to one single candidate. This way we can group information about candidates (name, votes, party) by index, through three ArrayLists. ArrayLists are automatically resizing arrays, to access you can use .add(index) and .remove(index)/.get(index).
b) If we don’t find it, create a new Election Node.
-You can use election.addCandidate(canName, votes, party).
-If this candidate is the winner, set the ElectionNode’s “winner” field to the index you insert them to (election.getCandidates().indexOf(canName)).
-Also set the ElectionNode fields for “raceID” and “officeSeat”.
-If officeSeat.substring(0,5).equals(“Class”), then the election is senate, so set the election’s field as such.
After setting the data, you can insert it at the front of the states elections singularly linked list.

Our goal is to insert a new ElectionNode in the singularly linked list of elections (within the correct StateNode, in the correct YearNode). Remember, each StateNode in a states list have a field called ‘elections’ which is a reference to the front of this SLL.

First, check if that elections list is empty (aka if the ‘elections’ field within that StateNode is null). If it is, create a new ElectionNode() and set its information with the info you read from the CSV line, and add the read-in candidate to that election node with .addCandidate( ).

Then, set the containing StateNodes ‘elections’ field to that new ElectionNode. Then you’re done, so continue onto the next CSV line.

If the SLL is not empty, then we want to check if an ElectionNode containing the read-in election already exists, as we do not want duplicates in the list.

- To traverse the list, declare a ElectionNode variable and set it to the year.getStates(), for the corresponding YearNode with the specified year. This grabs a secondary reference to the front of the states list.
- Traverse the list using a while loop, similar to traversing the years list. The condition should check if the ptr.getNext() is null, OR the ptr.getRaceID() is equal to your read in raceID.
- After your traversal loop ends, check the value of ptr:
  - If it’s raceID is equal to the read-in year, continue onto reading the next line of the CSV.
  - If it’s year is not equal to the read-in year, then it should be the end of the list. So ptr.getNext() should be null.
    - Create a new ElectionNode with the read-in information
    - Insert that new node at the end of the list, by setting the ‘next’ field of the last node in the list equal to your new node
    - Add the read-in candidate to that election node with .addCandidate( )

PROVIDED – changeInTurnout(int firstYear, int secondYear, String state)

This method is provided for you. DO NOT modify this method in any way.

This will call the next method twice, to return the change in total votes between two years.

totalVotes(int year, String state)

This method returns an integer corresponding to the total number of votes cast in all elections in a certain state in a given year.
You are given the target year and state name as parameters.

To complete this method, search the years list for the YearNode corresponding to the given year.

Then, search the states list for the given state name.

Traverse that states “elections” singly linked list and add up the votes for all elections in that state in that year.
You can use elec.getVotes() to get the total votes of an election.

Finally, return the total votes in all elections.

If the year does not exist, or the state does not exist within that year, return 0.

averageVotes(int year, String stateName)

This method returns an integer corresponding to the average number of votes cast in all elections in a certain state in a given year.
You are given the target year and state name as parameters.

To complete this method, search the years list for the YearNode corresponding to the given year.

Then, search the states list for the given state name.

Traverse that states “elections” list, and add up the votes for the elections in that state in that year, and ALSO count the total number of elections.

Then, use integer division to return the avg votes of elections that occurred.
You can use elec.getVotes() to get the total votes of an election.

If the year does not exist, or the state does not exist within that year, return 0.

candidatesParty(String candidateName)

This method searches through all elections and returns the political party which a candidate most recently ran with.

You are given the target candidates name as a String parameter.

To complete this method, search through the years list.
In each year, search each election in every state for an election containing the given candidate.

- If you find one, store the value of the party they ran with and continue onto the next year.
  - You can get a candidates party with “election.getCandidates().get(election.getCandidates().indexOf(CANDIDATENAME))”
    - replace “election” with whatever variable holds the election, and CANDIDATENAME with the string candidate name.
    - Each election has three lists of data (one for names, one for #votes, one for party). Each index across the three lists corresponds to one candidate.
      - i.e. candidates.get(0), votes.get(0), and party.get(0) all correspond to one candidate, and together can give you the candidates name, number votes received, and party they ran with.
- If you were already storing a party value, you can overwrite it (since we want the most recent one, and the years list is in ascending order)

Finally, return the last political party the candidate ran with, or null if that candidate was never found in any election.

Implementation Notes

- YOU MAY only update the methods with the WRITE YOUR CODE HERE line.
- **COMMENT all print statements you have written from ElectionAnalysis.java**
- DO NOT add any instance variables to the **ElectionAnalysis** class.
- DO NOT add any public methods to the **ElectionAnalysis** class.
- DO NOT add/rename the project or package statements.
- DO NOT change the class **ElectionAnalysis** name.
- YOU MAY add private methods to the **ElectionAnalysis** class.
- YOU MAY use any of the libraries provided in the zip file.
- **DO NOT** use System.exit()

VSCode Extensions

You can install VSCode extension packs for Java. Take a look at this tutorial. We suggest:

- Extension Pack for Java
- Project Manager for Java
- Debugger for Java
- Test Runner for Java

Importing VSCode Project

1. Download ElectionAnalysis.zip from Autolab Attachments.
2. Unzip the file by double clicking.
3. Open VSCode
   - Import the folder to a workspace through **File** > **Open**

Executing and Debugging

- You can run your program through VSCode or you can use the Terminal to compile and execute. We suggest running through VSCode because it will give you the option to debug.
- How to debug your code
- If you choose the Terminal:
    - first navigate to **ElectionAnalysis** directory/folder (the one that directly contains the src, lib and bin folders).
        - to compile: **javac -d bin src/election/*.java**
        - to execute: **java -cp bin election.Driver**

Before submission

COMMENT all printing statements you have written from ElectionAnalysis.java

Collaboration policy. Read our collaboration policy here.

Submitting the assignment. Submit *ElectionAnalysis.java* separately via the web submission system called Autolab. To do this, click the *Assignments* link from the course website; click the *Submit* link for that assignment.

Getting help

If anything is unclear, don't hesitate to drop by office hours or post a question on Piazza.

- Find instructors office hours *here*
- Find tutors office hours on Canvas -> Tutoring
- Find head TAs office hours *here*
- In addition to office hours we have the CAVE (Collaborative Academic Versatile Environment), a community space staffed with lab assistants which are undergraduate students further along the CS major to answer questions.