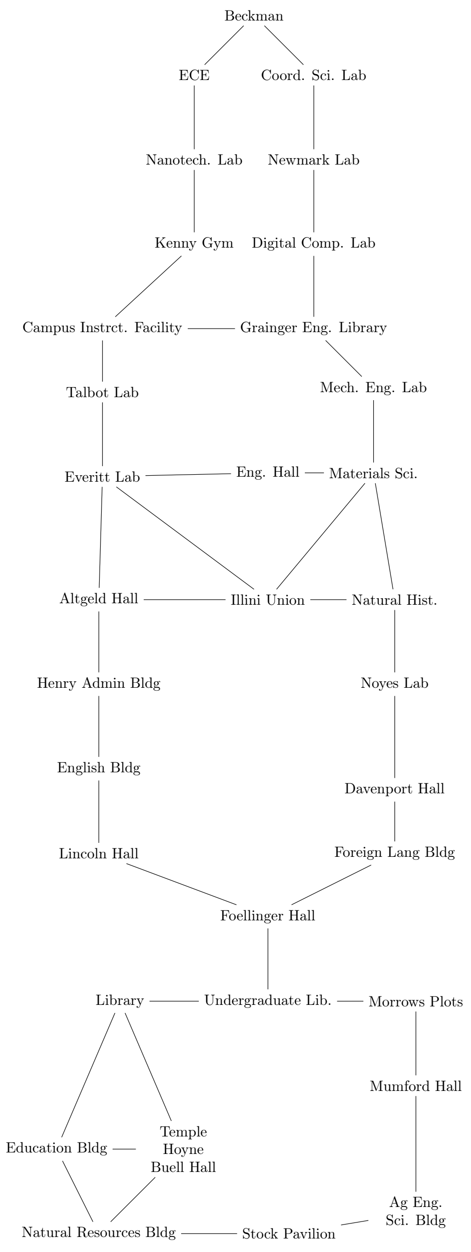
University of Illinois @ Urbana-Champaign

CI 487: Data Structures for CS Teachers

Project 3: Campus Navigator



Objectives and Overview

At a high level this assignment relies on two data structures in the main method of the `Main` class:

1. `Graph<String> map`: This is an undirected, weighted graph which holds information on the locations and distances between locations.
2. `Map<String, Location> locationInfo`: This is a directory which has the code for each building location as the key and the location object containing the actual name and information associated with the building as its value.

The information used to populated each of these data structures is contained in the files `vertexlist.csv` and `edgelist.csv`. The methods responsible for reading this information and populating `map` and `locationInfo` are `readCSV(String fp` and `populateCampusMap`. Both these files and the methods are provided for you but it is *highly* recommended that you read through these files and methods to understand their contents.

Looking at the edge list file you will notice that shorthand codes for the buildings are used in place of their names. This is to reduce entry error since some of the names are long and can become tiresome and error prone to enter. This graphs primary purpose is to construct the map object, where the codes are the vertices and the weights of the edges are those that are stated in the file.

As for the vertex list, this is primary used to construct the information directory `locationInfo`. The building codes present in the file as the first entry in each line will be used as the keys and the two entries following that, which represent the full name and info, will be used construct objects to contain this information for use as the values.

Your task for this assignment, will be to implement the ability for the user to navigate around our little virtual campus map.

1 Structures and Specifications

This assignment is divided into two main parts. Section 1.1 details the construction of a `Location` class which will be used to store information on a given location in each vertex of the graph. Section 1.2 will cover the main portion of the assignment which allows a user to: look around from their immediate location, move to one of the adjacent location, get info on the location they're currently on, and find the shortest path to some other location.

1.1 Step 1 - Create a Location.java File and Class

Each entry in the graph will hold two pieces of information: the name of the location and the info associated with the location. As such, we will need to create a class that holds this information and manages access to it. Complete the following steps to create a `Location` class.

1. Create a file named Location.java and create a `public class` within that file called `Location`.
2. Create two String attributes that are `private final` called `name` and `info`.
3. Create a constructor for that class that allows both of those parameters to be set.
4. Create getters for each of the attributes.
5. Create a toString method that returns a formatted string containing the name and information associated with the location.

You will use this `Location` class to store information relating to each location. After performing this step you should be able to compile and run the program. It should producefollowing output:

```
Locations: [collegeeng, cif, morrows, noyes, ece, davenport, dcl, ...]
Choose a starting location:
```

The first line of the example output has been truncated however the first series of building codes should be the same.

1.2 Step 2 - Implementing Navigation in Main

In the code that is provided, it constructs the map of our campus (`map`) and the information directory for our campus (`locationInfo`). From there, it lists all available starting locations in the form of their building codes and asks the user to select one. If the selection is valid it moves onto the while loop which handles all of the navigation procedures. At each iteration it should ask the user to enter one of the following commands: 1) help, 2) move, 3) info, 4) look, and 5) navigate. Each of these command should have the following behaviors:

- *help:* If the user enters help, it should call the help method which will print the users current location code as well as all available commands.
- *move:* If the user enters move, the move method should be called to handle asking the user which of the adjacent vertices they want to move and validating that enter. **Additionally,** the variable `currentLocationCode` should be update to equal what move returns such that the users current location is update in the event the move they selected was valid.
- *info:* This command will call the info method which takes the information directory and the users current location code, looks up the Location objects associated with that code, and prints out information on where the user currently is.
- *look:* This command takes the map and the users location and prints the location codes of all adjacent vertices.
- *navigate:* Navigate will take the user's current location code, display the location codes of all vertices in the graph, ask the user which they want to traverse to, and use the implementation of Dijkstra's SP to print the list of vertices that make up the shortest path to that destination.

If the user enters a code not in the above list a message indicating that is the case should be printed and the loop should continue to the next iteration. The details on how these methods should be implemented along with some selections of example output are detailed below.

`public static void help(String locationCode)` : This method should print out your current location and the list of available commands which should include all methods below. Feel free to print out any additional information you think might be helpful in navigating your map. Below is an example of the output that should be produced but are welcome to produce and output of your own making:

```
You are currently at <currentLocation>. You can enter the following commands:
    * look
    * move
    * info
    * navigate
```

`public static info(Map<String, Location>, String currentLocationCode)` : This should get the location info associated with the current location from the `locationInfo` Map and display it for the user. For example if the user has the current location code `morrows` the output should be:

```
Morrow Plots:  The Morrow Plots is an experimental agricultural field
at the University of Illinois Urbana-Champaign. Named for Professor
George E. Morrow it is the oldest such field in the United States and
the second oldest in the world.
```

Hint: The parameter `currentLocationCode` is a key to a String-Location pair in the Map. As such, you will want to use the `get` method to get that location and then print the full name of the location and it's info block.

`public static void look(Graph<String> map, String currentLocationCode)` : This should use the graph and the current location code to get the list of adjacent vertices and display them for the user so they can see which locations are immediately around them.
Hint: You should use the map method map.getAdjVertecies(code) to get the list of vertices that are adjacent to the location code you provide as a parameter.

`public static String move(Graph<String> map, String currentLocation)` : This method should print the list of available locations to the user and ask them which they would like to travel to. If they enter a location that is not adjacent you should print a message indicating that is the case and return the original value of `currentLocation`. If the location is valid, it should return that string.

Hint: You will want to get the adjacent vertices in the graph (i.e., map.getAdjVertecies(curr)) and then use the .contains() method to see if the user input is in that list of strings..

`public static void navigate(Graph<String> map, String currentLocationCode)` : This method should initially print all available vertices in the graph (*Hint: use map.getVertecies()*) and ask the user to select one of them. If the location that the user selects is invalid, a message indicating that that is the case should be outputted. However, if the users selection is valid, it should then compute the shortest path from the users current location to that which they selected and print that path (*Hint: use map.dijkstraShortestPath()*). For example, if the user begins at Morrows and wants to navigate to Beckman, they you should get the following output:

```
Enter a command (or help):navigate
Locations: [collegeeng, cif, morrows, noyes, ece, davenport, dcl, thbh, mech, foreign,
    nanotech, library, naturalresc, kenney, english, ugl, stock, mumford, altgeld, ag,
    newmark, union, beckman, henry, talbot, lincoln, materials, edu, naturalhist, csl,
    everitt, foellinger, englibrary]
Select a location to navigate to: beckman
The shortest path is: [morrows, ugl, foellinger, foreign, davenport, noyes, naturalhist,
    materials, mech, englibrary, dcl, newmark, csl, beckman]
Total travel distance: 210
```