Computer Science 211

Data Structures and Algorithms

Spring, 2017

Lab Report - *Implementing Dijkstra's Algorithm*

Matthew Jordan

CSCI 211 Spring 2017

Assignment Analysis and Design

Overview

This program was probably the funnest one we've done so far in the class. It really felt like a big-ish scale project due to the number classes and the complexity of the algorithm that we used. This program takes what the professor (Charles Herbert) has already written and augments it to include an algorithm called Dijkstra's algorithm and uses that algorithm to compile a stack of vertexes to represent the 'lightest' path between two points specified by the user.

Before any work was done, the code for the generation of the map and the graph of cities drawn on top of the map was already provided by the professor. The City Project, CityMap, City, Vertex, Edge, and AdjacencyNode classes were already present and largely functional. The classes Interface and CityStack were added and implemented by me.

Designing the GUI

Before the start of the project, I knew I wanted to implement an intuitive way for the user to select which cities they wanted to perform the algorithm on that were foolproof, as the need to type in the name of a vertex EXACTLY as it is known as in the class would make testing arduous. So I started with a new class titled 'Interface' and build a simple GUI that the user could choose the Cities they wanted from a drop down that way they didn't have to type it in every time. This proved difficult as I had forgotten how to use swing and awt. So I went and read the previous chapters of the book to relearn.

What I basically did was set up an interface class that implemented JFrame and implemented ActionListener so it would be able to register the inputs from buttons and drop down menus. The properties of this GUI would have the two JCombo boxes, one for the start city and one for the end city, the button to run the code to calculate shortest distance, all of the data for the graph (i.e. the array of Cities and it's count, and the array of Edges and it's count), two City variables to hold the user selected startCity and endCity variables, and a variable to hold a Stack of Cities once the calculation was finished. In addition to all of this there is a JFrame variable that is used to manipulate the canvas for CityMap so data can be passed to the other window. In retrospect I wish I had found a way to implement the GUI that I made into the CityMap class because figuring out how to update the canvas from another class was very difficult.

The constructor for the interface class takes all of the data for the graph, as well as the JFrame that has the canvas as a component. It instantiates the drop down menus by passing the cities arrays through their constructors as well as the button that calls the methods for Dijkstra's algorithm. The actionPerformed method is added to all three components of the JFrame to instantiate the values for the startCity and endCity, as well as call Dijkstras algorithm for the calculate button, which is the part of the assignment that is most important.

Dijkstra's Algorithm and related code

After the user choses two Cities and presses the calculate button the first the ActionListener does is run a loop that goes through the whole array of Cities and calls a method that I wrote for the City class that sets all of their relevant information. It sets each City's visited variable to false, it set's it's bestDistance variable back to MAX_VALUE, and sets it's immediatePredecessor to null. Then it set's the cityStack variable to null as well, this variable keeps track of the Path from the starting city to the ending city. Then it calls the printNewLines method which takes all of the data in the CityStack and draws the lines from the starting city to the ending city. The first time that the calculate button is pushed, none of this is necessary because none of that information has been calculated yet, but it is necessary for any subsequent calculations.

*Next DijkstrasAlg (**Dijkstra's Algorithm)** is called and which returns a Stack that is passed to cityStack. The cityList array is also passed through this method. DijkstrasAlg starts by instantiating a variable for the current City being looked at, which starts off as the startCity variable. That city's best distance is set to zero and then a while loop is started (***END STEP 1***). The exit condition for this while loop is a call to a method called isUListEmpty. What isUListEmpty does is to iterate through the entire cityList array and check for the first City that is unvisited, if there is a City that is unvisited, the method returns false at the first instance. While that method returns false the while loop will iterate. I realize this could possibly be simple by returning true from the method but for ease of understanding I decided that it was better to set the method's return to check if it's equal to false.*

*Inside the first while loop and AdjacencyNode is instantiated from the adjacencyListHead variable in the currentCity's linked-list. This list is the first of many that we will preform calculations with. That node's distance is added to currentCity's getBestDistance to create a new distance to compare the adjacent city's bestDistance's variable. If the total of **currentCity's** bestDistance and **adjacent City's** distance <— (**this distance is the distance from currentCity, not startCity**) is less than the current value for adjacentCity's getBestDistance then getBestDistance is set to the value of the previously calculated addition. This process is repeated for every City in currentCity's adjacency list. Once the whole list has been looked at, the inside while loop is exited (***END STEP 2a***) and currentCity's visited value is set to 'true' (***END STEP 2b***).*

*Next the algorithm call a method to look for the closest bestDistance value for every City in the cityList array that has a 'false' value for the visited variable and returns that City, called closestUnvisited. Back in the main while loop, the currentCity variable is now the City that was returned by the closest Unvisited method (**END STEP 2c**). This process is repeated until every city in the cityList array has a 'true' value for the visited variable. Once that is completed the method takes the City variable endCity and creates a Stack from the linked list that was generated by the algorithm by calling the getImmediatePredecessor method and pushing the return of that method to the top of the stack until the startCity is reached (**END STEP 3**).*

Drawing the Map

This stack is returned from the DijkstrasAlg method back into the actionPerformed method which then re-draws the entire map, but this time when it is finished drawing the map it draws the whole path from the starting city to the destination. This I did by passing the generated Stack through the constructor method of cityMap. I managed this by making a second constructor method for cityMap that took the CityStack as the last variable to pass into it and adding a CityStack to the data contained in the cityMap class.

Instantiating a new cityMap was almost exactly the same as the other constructor. In fact other than setting the CityStack variable the altered Constructor is identical. The difference this time is in the paint method. In between the part of the method that draws the ovals to represent the Cities on the map and the drawString method which paints the names of the Cities onto the map there is a branch that checks to see if cityStack is null. If it is not null then the method draws the edge of each vertex in the stack, as well as large colored ovals for the starting and ending points in the stack. As well as displaying each element in the stack on the left hand side of the map with the total distance at the bottom.

Assignment Code

Included in submission

Assignment Testing

I tested this program many time along the way, adding necessary methods, most of which printed out to the console as this was before I had fully implemented the GUI and map-updating. I regrettably deleted those methods, thinking that they were just adding to the clutter. I checked the algorithm a few times by hand and it was correct each time. I compared my stack to Phat Phan's stack for the same start and end points and we had the same answer.