**Assignment 8 - Driving Directions**

*Due Date: 11 p.m., May 6, 2020*

*All public interfaces are minimum requirements. You may add additional public methods if needed.*

For this project you are going to create a library that reads in a file with a list a cities and the roads between the cities. The user will input a starting city and destination city, and you will return the shortest path between the two as a list of cities along the path.

You are required to use at least one of your previous ADT implementations as storage.

Part A: Setting up

For Part A you will read in a text containing cities and their x and y coordinates along a grid.

The file format will be as follows:

paris 6 7

london 3 5

rome 4 7

dublin 3 2

barcelona 4 2

City names will not contain whitespace and will be uniquely named. Input files will be well-formed (no need for error checking). You will need to build an undirected, weighted graph using the x and y coordinates.

The user can only travel in straight lines, so paths are along the x and y grid. In other words, there is a path between two cities if they have equal x coordinates or equal y coordinates and **there is no city between them on the path.**

For example, if you have 3 cities along the same x axis A[5,3], B[5,6], and C[5,4], A is only adjacent C; however, C is adjacent to B in the north direction, and A in the south direction. A city will have, at most, 4 adjacent cities. The adjacent city in any one direction is **only** the closest city in that direction. So the algorithm for building the adjacency list will be to find all cities in one direction, but set only the closest one as adjacent.

You should use an array or vector to store the cities. To hold edges you will need to implement an adjacency list or an adjacency matrix (your choice). Below is a **suggested** structure for your graph. You can make any design changes you like as long as you implement a graph representation (adjacency list or matrix) and utilize an ADT you implemented for one of our previous assignments ) to perform Dijkstra’s Algorithm.

* struct City
  + City attributes
    - int x
    - int y
    - char \* cityname
    - list \* adjacents
  + City related functions
    - City \* createCity(char \* cityName, int xCoor, int yCoor);
    - list \* getAdjacent(City \*);
      * Returns the adjacency list
      * You can modify your Data struct to hold a city pointer
* struct Map
  + Map attributes
    - list \* cities
  + Map related functions
    - Map \* createMap(char \* filename)
      * reads in the file names and creates the list of cities
    - City\* findByName(Map \* map, char \* cityName)

Part B: Finding the Shortest Path

You will need to implement Dijkstra’s shortest path algorithm to determine the shortest path between two cities. If you want to implement a different shortest path algorithm, you must clear it with me first. You must travel in straight lines, and can only change directions at cities. In other words, if cityA has coordinates [5,7] and cityB has coordinates [6,2], there is no path between them. However, if you add cityC with coordinates [5,2], you now have a path between cityA and cityB through cityC.

You should add the following function:

* list \* shortestPath(Map \* map, City \* start, City \* dest)
  + The method should return the shortest path between two cities by returning a list of the cities you will need to travel through.

You will also need a second function that gives the distance between two points on the graph:

* unsigned int pathDistance(Map \* map, City \* start, City \* dest)
  + The function should return the total distance (based on the path you must take) between the two cities.
  + The method should return -1 if there is no path.

Part C: Testing your Code

In your program8.c print out a list of cities, and ask the user what city they would like to start in. Then ask the destination city. You should then print out the shortest path as a list of cities, and the total distance between the two cities.

Part D : Code Organization and Submission

* Required code organization:
  + program8.c
  + Map.c/h
  + City.c/h
  + Data.c/h
  + <One of your Previous Data Structures>
  + makefile
  + readme (if necessary)
* Use below command to tar.gz your assignment folder.
  + tar -czvf <yourid>\_program8.tar.gz <assignment folder>

**Do not include your object files or executables**

# *Tests*

### Test cities files

* + townlist.txt
* Paths I will check (not limited to):
  + Test 1: “attilan” => “smallville”
  + Test 2: "doomstadt" => "gotham"
  + Test 3: "gotham" => "madripoor"
  + Test 4: “attillan”=>”nyc”
* Distances I will check:
  + Test 5: “nyc” => “nyc” = 0
  + Test 6: “attillan” => “madripoor” = 8
  + Test 7: “metropolis”=>”doomstadt” = 8
  + Test 8: “attillan”=>”nyc” = infinite
* Your implementation will also be tested with an unpublished data set.

# *Grading Guidelines (25 pts total)*

## Part A

* + City struct (2 points)
    - uses a constructor and destructor
  + Map struct (2 points)
    - uses a constructor and destructor
  + An Adjacency List or Adjacency Matrix is used to keep track of adjacencies (3 points)
  + A previous ADT is used to implement the Adjacency List or Adjacency Matrix (2 points)
    - should only alter the Data struct to hold a city \*

## Part B

* + Tests 1-8 (2 points each)

## Style Guidelines and Memory Leaks

* + Valgrind Shows Memory Leak or Error (-5 points)