CS 305: Finding Shortest Flight Paths
Fall 2019

During the semester free tutoring is available for CS 305. In addition to tutoring you can email: stepup@up.edu to schedule an appointment for more in-depth tutoring.

Due Date: Sunday November 24th NLT Midnight. All the code files (.h and .c files), makefile, and short answer questions in a pdf or docx file should be submitted as a single zip file to Moodle.

This assignment should be completed **individually**. Advice: start as soon as the homework assignment is assigned, so you have plenty of time to get the code to compile, have time to test, have time to debug, have time to re-test, and have time to complete the summary report.

Specification

You will complete the code to implement Dijkstra's algorithm (shortest paths from single source node) to find the shortest flight plan (in total mileage) between a source city and destination city.

For example, if the user types:
```
./routeFinder SampleRoutes.txt PDX IAD
```

the program reports:
```
The path is 3 flight(s) long:

PDX -> SEA -> ORD -> IAD.
2433 miles.
```

The small graph sample in SampleRoutes.txt consists of a set of 18 (line 1) airports with the airport names listed alphabetically first (lines 2-19) followed by the flights between the cities (lines 20-end). Graph is directed, so if the flight connection is from A-B but not from B-A, the connection B-A is not listed in the flight connection list. If there is a flight from PDX to SEA, and a flight from SEA to PDX then both flight connections are listed but don't assume they have the same distance. The graph is weighted by the mileage between the two airports. The mileages and connections are from www.world-airport-codes.com/distance/ or https://www.transtats.bts.gov.

Open `graph.h`. The graph is implemented as an adjacency list.

A node in the adjacency list contains:
- the destination
- cost
- next node in the list.

An adjacency list object represents a single vertex in the graph and contains:
- airport name (3-letter code)
- predecessor (for dijkstra's algorithm)
- color (for dijkstra's algorithm)
- dValue (for dijkstra's algorithm)

- head (pointer to head of linked list of adjacent nodes)

The entire graph is represented as:
- V (number of vertices in the graph)
- array (the array of adjacency list objects)
- jump (mapping from node numbers, 0 through |V|-1, to airport codes)

It might be helpful at this point for you to draw a simple graph of a few nodes and convert the graph to the code representation. Before asking for homework help, be sure that you do this step to ensure you understand the graph data structures.

Open `graph.c` to see how the functions are implemented. Note that this assignment does not ask that you alter the graph representation, but reviewing these functions will help you understand how the adjacency list representation is implemented.

Now open `main.h`. This file defines a few constants (for command line arguments) and the prototypes for the functions defined in `main.c` and the new file `dijkstra.c` that you will create. You will implement the following three functions: dijkstra, isEmpty, and getMin.

Open `main.c`. This file creates the set of vertices (airports) and checks that the program is being run correctly. It checks that the airport names given at the command line are in the graph. If not, it reports an error and exits. Then, the airports are sorted and a graph with these airports is constructed. In the `buildGraph` function, you will see the flights (edges) that exist between airports. Then, `dijkstra` is called from the source. The path is found, starting at the destination and working through the predecessors back to the source. The total number of flights along the shortest path, the path, and the total mileage is printed to the screen. Finally, the graph object is freed.

At this point, you should have a good idea what the starter code does. If you have questions, please ask.

GET STARTED

0. Implement the freeGraph function at the bottom of the graph.c file. The function should free all structures allocated on the heap. This should force you to understand how the graph and application logic is implemented. Run your code through valgrind to make sure you implemented the free function correctly
`valgrind ./routeFinder AllUSRoutes.txt PDX SFO`

Next:
* Remove the block comment by deleting the lines 56 and 131 in main.c
* Alter the makefile as needed

1. Create a new file called `dijkstra.c` in the same directory as the other files. At the top, include <stdio.h>, <stdlib.h>, <string.h>, “main.h”, and “graph.h”.

Implement the following functions (can copy and paste function prototypes from `main.h`):

- `isEmpty`
  - Note: This function should return 1 if there are no white nodes in the graph g (all have been colored black by dijkstra). It should return 1 if the graph is null. It should return 0 if there is at least one white node in the graph g.
  - Hint: g->array[i].color is the color of the ith vertex in g.
- `getMin`
  - Note: This function should return the index of the smallest white node in the graph g.
  - If the graph is null, return -1.
  - Hint: iterate through the graph; if the node is white, check to see if its dValue is smaller than the smallest found so far; if so, update the smallest value and index to this node.
  - If the graph has no white nodes, return -1.
- `dijkstra`
  - Note: This function should implement dijkstra’s algorithm from the src node to find all the shortest paths from src to other nodes in the graph g. When the function is finished, all dValues for all nodes should be set, all nodes’ colors should be black, and all nodes’ predecessors should be set.
  - Note: At the beginning (graph is initialized), the nodes’ colors are set to white. If this function was to be called more than once in a program, we would need to set all nodes’ colors to white. So, you may assume g->array[i].color is white for all values of i when g is passed in to dijkstra.
  - You should implement the algorithm as follows:
    - Find node number for `src` (go through the labels of the vertices and return the index of the node when the `src` string is equal to the label; you can use `strcmp` or `strncmp` to compare strings).
    - Set the source node’s dValue to 0.
    - While the graph is not empty (use `isEmpty` function):
      - Let U = getMin(g) //use `getMin` function
      - For each neighbor N of U:
        - If N’s dValue > [U’s dValue + cost(U, N)], update N’s dValue to [U’s dvalue + cost(U, N)]. Also, update the predecessor for N to be U.
      - Set U’s color to black

At this point, you should be able to compile and run the program:

```
make clean; make
routeFinder SampleRoutes.txt PDX IAD
```

(or)

```
./routeFinder SampleRoutes.txt PDX IAD
```

The implementation prints out a bunch of information about the graph and then the shortest path information as at the end.

Documentation

The code should be well-commented, well indented, use consistent bracing, and include enough whitespace to be readable. The `dijkstra.c` file should have a block comment at the top that includes information about the author and included code. Each function should have a header comment. See the file posted to moodle about good style that was generated by the CS 305 students.

Test cases

- You should try running the program of several pairs of cities (some that are not in the graph and some that are).
- Test your code on both test files provided. The AllUSRoutes.txt uses what US Government's FAA has in its database as far as the airports go and I pulled all flights for all carriers for 2017 to establish the connectivity between individual airports. Your code has to work on files SampleRoutes.txt and AllUSRoutes.txt.
- Graph is not connected, so some airports serve only local connections (or the FAA's list keep the airports without service, or the airports are military without commercial carrier service), but not to other US cities.
- Try to find the longest air route
- Try to find the route with most connecting flights (will still be shortest route)

Additional Enrichment

If you have time, you may try writing the graph data structures from scratch instead of using the starter code.

You could try modifying the graph so it is undirected instead of directed. But, be sure to save your original code in a directory and put the new code in a separate directory.

You could try adding flight times and check that the flights depart after the earlier flights land in the path. (This will require some substantial changes to the existing code.) Be sure to save your original code in a directory and put the new code in a separate directory.

Please document the additional features in your code and your written summary. This is not extra credit per se, but a chance for you to do some exploration.

Logistics

1. Download the starter code located on Moodle. If using Linux, you can unzip it with the `unzip` <name of zip file> command. You may use any IDE that you can find for C or just a text editor (emacs) to write your code. However, your code *must compile* with gcc on the EGR Linux machines. It is to your advantage to make sure your code compiles with gcc on the Linux computers, since that is used for grading.
2. When you are finished and ready to submit:
    1. Create a new folder called `username_HW5`.
    2. Put all starter code, your `dijkstra.c` file, and makefile in this folder.
    3. Copy your HW5Summary.docx file into this folder.
    4. Zip the folder by right-clicking and Send To->Compressed Folder (on Windows). There should also be a zip utility on Mac.
3. **What to turn in:** Submit your **username_HW5.zip** file to Moodle before the due date and time. After logging into learning.up.edu, navigate to CS 305. You will find a link to submit this file. You do not need to print anything for this homework submission.

Grading Guidelines (total of 20 points)

Your files will be graded in three separate categories:

- (5 points) Code Quality: Design, commenting, whitespace, readability, proper indentation, consistent naming conventions, good variable names
- (10 points) Code Operation: Does code do what is listed in the specification? Note: code that does NOT compile will receive 0 points for code operation.
    - 2 points freeGraph
    - 1 points isEmpty
    - 1 points getMin
    - 6 points dijkstra
- (5 points) Summary report: Completeness, correctness, and clarity

HW 5 Report Guidelines and Format – use the template provided below. Include the questions in your write-up.

Name: **CS 305 HW 5 Report**

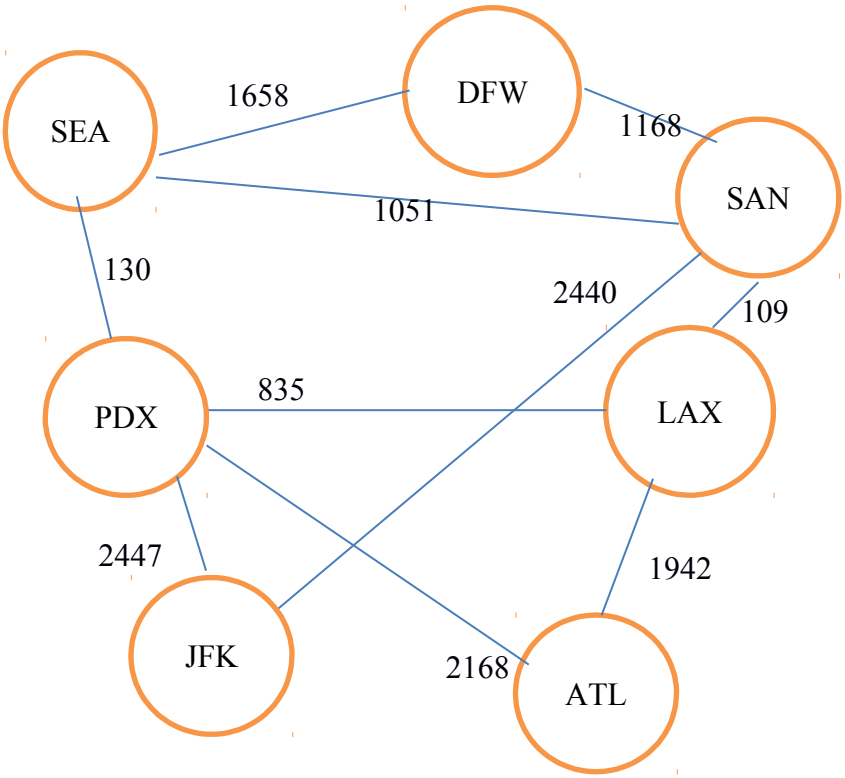
1. Questions (include these in your write-up):

1a. (.5 pt) If your program does not meet the specifications, please note the differences and anything that is not working correctly. If it works, write "Meets all specifications."

1b. (1.5 pts, 0.25 pts each) What is the output from your program for the following pairs of cities? (The top row has been done for you.)

Source	Destination	Path	Miles
PDX	MCO	PDX -> ATL -> MCO	2572
LAX	PVD		
ATL	JFK		
SEA	LAX		
PHX	DEN		
PVD	JFK		
DFW	ATL		
MTM	MRI		
HIB	HIO		
INL	ZZV		
WWD	YUM		
YAK	BAF		
AVL	BIF		

2. (1 pt) Use **Prim**'s algorithm to find the minimum spanning tree for the following subset of the airport graph. Start the algorithm using vertex LAX.



In your answer, show the final minimum spanning tree edges and all the vertices.

3. (1.5 pts) Show each step of **Kruskal**'s algorithm for finding the minimum spanning tree for the graph in #2. Draw the minimum spanning tree at each step of the algorithm. For example, the first step would be to put edge (PDX, SEA) into the spanning tree. Be sure to enumerate all the steps and show the final minimum spanning tree. (For the graph above)

4 a. (.25 pt) How much time did you spend in total on this homework assignment (including the report)?

b. (.25 pt) What was the most challenging part for you when completing this assignment?

Appendix A: (copy this statement if it applies to you) I verify that the code and this write-up were authored by me. I have documented the help I have received in comments in the code files. I have not distributed my code to anyone else except via this homework submission.

Appendix B: Copy and paste your `dijkstra.c` here (use Courier New 8pt font so the characters line up correctly)