Data Structures

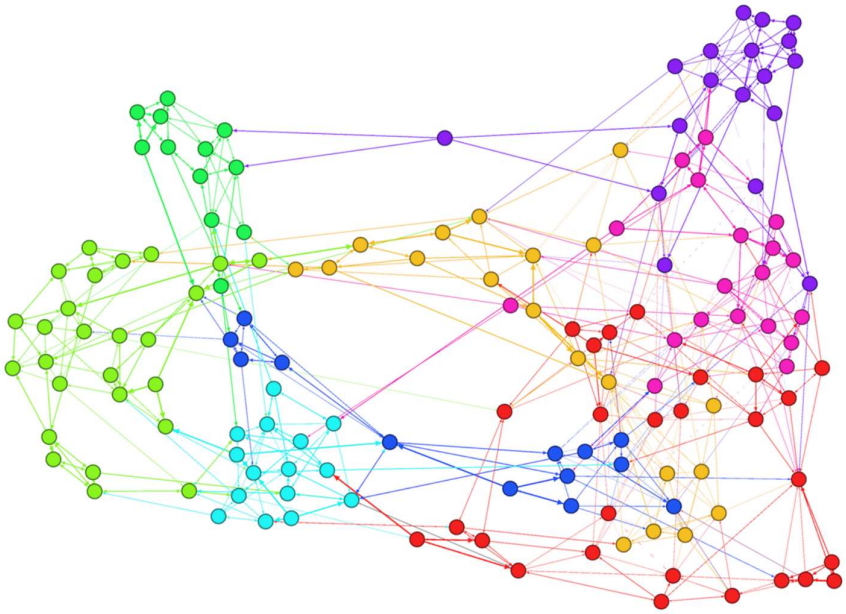
CS202

Assignment 5

Deadline: 7th December 2019

11:55 pm

In case of ambiguity, contact Anusheh Zohair Mustafeez at 21100072@lums.edu.pk



Ali and Mannan are procrastinating instead of working on their project. Mannan has come across a play titled “Six Degrees of Separation” written by American playwriter John Guare that says that every person in world is connected to everyone else in the world by a chain of no more than six connections. Mannan asserts that Ali knows Donald Trump by way less than 6 connections, whereas Ali says even if that is true Mannan will be related to Donald Trump by lesser connections than him. Things escalate and now they desperately want to prove their points. Hadi, who is sitting nearby interjects that this problem can be easily solved using graph theory. Ali and Mannan quickly bring their Googling skills in use and find a subset of dataset of connections made public by a popular social media company. But, since they did not pay attention during their Data Structures classes, they do not know how to make useful deductions from the dataset. They need your help to settle the feud between them.

Part 1: Parse the dataset:

You are provided with the mentioned dataset in the form a text file, ‘friends.txt’. You can explore the dataset provided to you, lines following the first line contains names of persons. You’ll use node class to represent a person. Lines following the line Connections is the connection between persons. You will use edge class to represent a connection between two people.

You must parse the dataset in two forms:

1) If Boolean variable isUnitLength is on, all edges must have weight equal to 1.
2) If it is off, you must pick the value provided to you with each connection. This value is called index of friendship. It is measured by how much two people interact with each other. **Lesser the value is for two people, closer those people are to each other.**

Part 2: Test Reachability:

Before finding the route between two people, you must check if these two people are even connected to each other according to our dataset. You have studied two Reachability algorithms namely BFS (breadth first search) and DFS (depth first search). You can implement either of these, but you will have to provide reason why you went with your chosen algorithm. Your algorithm will take name of first person (i.e. your starting node) and toFind person (person you are testing reachability for) as input and must return true if these two people are connected and false otherwise.

Part 3: Shortest Path Algorithm:

You have studied Dijkstra's well renowned algorithm for shortest path in class. You will implement Dijkstra's algorithm in this part to find shortest path between two people. It will take name of starting person, last person as input and will return a vector containing names of all people along the path. The additional variable d is for path length.

Dijkstra Primer:

To find shortest path, one approach can be to find all the possible paths and then look for shortest among them. But, this algorithm is not scalable and have an exponential execution time. So, we move towards a smarter algorithm called Dijkstra's Algorithm named after its creator Edsgar Dijkstra.

The idea is to keep a queue of all paths constructed so far, prioritized in terms of distance (A perfect use for the priority queue). At each stage, you dequeue the shortest path; if it leads to the destination, your work is done. Otherwise, you use that path as the basis for constructing new extended paths that add an edge leading away from the node at the end of the path. You discard the extended path if you already know a better way to get there (i.e. you have already found a shorter path to that node), otherwise you enqueue the extended path to be considered with the others. After updating the queue, you dequeue the next shortest path and explore from there. At each step, the search expands outward until you find a path that ends at the destination. The graphs in the data files are fully connected, meaning, every pair of nodes is connected by some path, and thus you will eventually find a path. The order of the paths considered by the algorithm guarantees that you will find the shortest such path first.

Part 4: Minimum Spanning Tree:

MST finds the edges in graph with minimum cumulative weight that connects the whole graph together. MSTs have a variety of applications such as minimizing cost of some operation, finding clusters in graphs, finding bottlenecks in graphs etc. You will use Kruskal's algorithm to find minimum spanning tree in this part. This function takes no input and returns a vector of nodes, where each node will have only the edges corresponding to MST.

Kruskal Primer:

Joseph Kruskal devised one of the simplest MST algorithms in 1956. It is based on the idea that you consider the edges in the graph in order of increasing cost. If the nodes at the endpoints of the edge are unconnected, then you include this edge as part of the spanning tree. If, however, some path already connects the nodes, you can discard this edge.

Given below is a primer that will help you get started:

- The tricky part of this algorithm is determining whether a given edge should be included. The strategy you will use is based on tracking connected sets.
- Initially, each node is in its own set all by itself. Which means there will be "n" sets comprising of individual nodes. Each set contains the nodes, which are connected by some edge.
- The algorithm then considers each edge, e, in turn, ordered by increasing weight. (Use getMin() of the priority queue for this)
- If an edge e connects two different sets, then e is added to the set of edges of the minimum spanning tree, and the two sets of nodes that are now connected by e are merged into a single set.
- If, on the other hand, e connects two vertices/nodes that are already a path between them using some earlier edge, then e is discarded. (Check to see if the two end points of the edge e belong in the same set)
- Once the algorithm has added enough edges to form a spanning tree (meaning that only one set of nodes remain and that set contains all the nodes), it terminates and outputs this tree as the minimum spanning tree.

Part 5: Deductions:

Run all the above algorithms in context of following questions in main of Graph.cpp and answer following questions. You should answer these questions by printing them out in the terminal.

1) Check reachability for following nodes and deduce nature of the dataset.
   i- Mannan, Ali
   ii- Mannan, Trump
   iii- Ali, Trump
2) For a graph with unit length weights, how many hops is Ali away from Trump?
3) What about Mannan?
4) Who has smaller value of path in terms of index of friendship?
5) Run the MST on both unit weight graph and weighted graph. Could there exist more than one MST for one of the graphs? What implications can you draw from result of both runs? What benefit can a social media website have from the MSTs you have produced? Can you think of other applications of MST in terms of social connection graphs?

Some useful libraries:

Priority Queues in queue: https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/

Sets: https://www.geeksforgeeks.org/insertion-deletion-stl-set-c/

Vectors: http://www.cplusplus.com/reference/vector/vector/

You are welcome to use templated version of heap you implemented in PA4 as a replacement for priority queue and your own implementation/replacement of set.

Best of luck! 😉

Note for compilation:

- Your code **must** compile properly on a Linux environment.

Submission Guidelines:

Submit a **zip** folder containing the following files on your lms Assignment 5 tab:

- Graph.cpp
- Graph.h

Naming Convention: A5_*your roll number*.zip