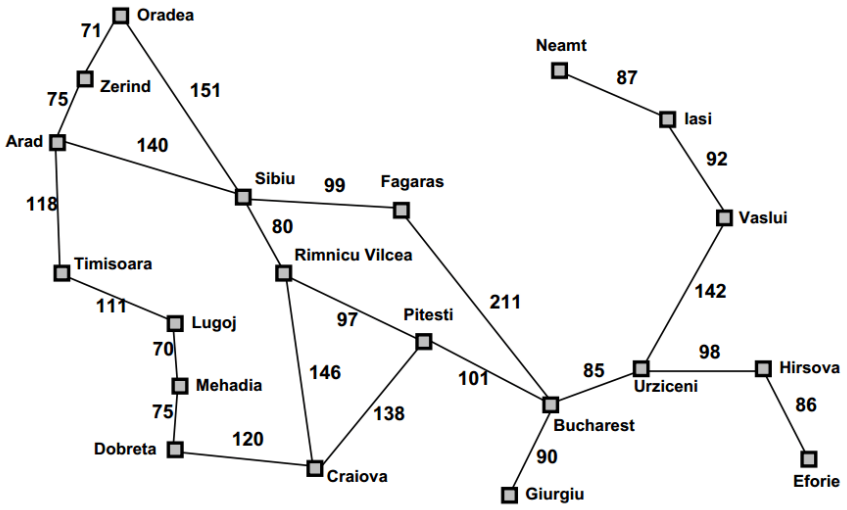
Introduction to Algorithms **F2023**

Problem Set 3

Handed Out: November 2nd, 2023 *Due: November 16th, 2023*



In this assignment will be focused on graphs! Well, one graph in particular: a map of cities in Romania! This is a relatively well known map in Computer Science circles because it is often used to introduce graph algorithms, and now it's our turn to play with it. In this assignment, you will be implementing 2 graph algorithms: Breadth First Search and Dijkstra's Algorithm. In addition, you will be implementing the data structures necessary to support these algorithms. I have provided you with two text files that contain the information required to construct this graph: RomaniaVertices.txt and RomaniaEdges.txt. As implied by the name, RomaniaVertices.txt contains the name of every vertex in the graph, and RomaniaEdges.txt contains information about every edge in the graph. RomaniaVertices.txt is pretty self explanatory in terms of file structure (each vertex name is on its own line); however, RomaniaEdges.txt is a bit more complex. RomaniaEdges.txt is a comma separated text file in which each row represents a different edge. The first two entries on a row represent the two vertices connected by the edge, and the last element in the row is the weight associated with that edge. You will only use this weight for some parts of the assignment. I will indicate when this is necessary. This graph is UNDIRECTED. To represent this in an adjacency list, you can simply add duplicate edges that indicate that directionality goes two ways.

As always, you will submit a writeup in Word or PDF that summarizes your results and all code as a zip file. Submit the writeup (with attached source code) to the Canvas submission locker before 11:59pm on the due date.

Implement a Priority Queue using a Binary Heap (20 Points)

For Dijkstra's Algorithm to work correctly, we must use a priority queue so that we can efficiently retrieve the node that has the smallest distance from the source node at any given time. The first part of this assignment simply involves implementing a priority queue data structure utilizing binary heaps, specifically a **MIN heap**. The heap itself should be created using something like an array or a vector. There are three functions that must be implemented for the priority queue to work properly: Insert, Extract Min, and Decrease Key. You should also implement any additional methods required for these three methods to work correctly.

The correctness of this data structure will be determined by its performance on Dijkstra's algorithm.

- As mentioned at the start of class, all code should be written in C++.
- Please include instructions for how to compile and run your code in your writeup.
- Explain any implementation choices you had to make in the final report (such as underlying data structure), and also include the text representation of your graph (described above)!

Breadth First Search (20 points)

In this part of the assignment, we're going to implement a class graph search algorithm: Breadth First Search! To support this, you must first represent the graph using an **adjacency list**. You can use any basic data structure to implement the adjacency list, as long as it is not a specialized one designed for graph computation (such as you might get from the network library). For this part of the assignment, you WILL NOT be using edge weights. The breadth first search algorithm that we discuss in class makes use of several auxiliary data structures, including a queue to keep track of the search process. You are allowed to use any data structure you'd like for these auxiliary structures, within reason (nothing that can just solve the problem with a single method call, but I don't think that will be an issue). This means that you can use pre-existing queue implementations if you'd like!

Along with the BFS algorithm, you will need to implement the PRINT-PATH method outlined in the book. This will allow you to print out the shortest path from some starting vertex, s, to any other vertex, v. Using this method combined with your BFS implementation, print out the shortest path from **Arad to Sibiu**, from **Arad to Craiova**, and finally from **Arad to Bucharest**.

Details:

- Feel free to use pre-existing data structures for your auxiliary data structures. As always, don't use methods that can, essentially, answer the question in one method call.

- As mentioned at the start of class, all code should be written in C++.
- Please include instructions for how to compile and run your code in your writeup.
- Explain any implementation choices you had to make in the final report (such as how ties were broken).
- Be sure to include your shortest paths in the writeup!

Dijkstra's Algorithm! (40 points)

Dijkstra's Algorithm! This is probably one of the most iconic graph algorithms out there. It calculates the single-source shortest path on a weighted graph. As you can probably tell, we're going to be using edge weights for this part of the assignment. In this part of the assignment, implement Dijkstra's algorithm. Here is where we will make use of the priority queue data structure that was implemented in Part 1! Once you've implemented Dijkstra's, use it to find the shortest path from **Arad to Bucharest**. You should be able to print out this path using a method similar to (possibly identical to, depending on your implementation) the PRINT-PATH method used in part 2. Is this path different from the path from Arad to Bucharest you found in part 2? Why or why not?

Details:

- As always, don't use methods that can, essentially, answer the question in one method call.
- As mentioned at the start of class, all code should be written in C++.
- Please include instructions for how to compile and run your code in your writeup.
- Explain any implementation choices you had to make in the final report (such as how ties were broken).
- Remember to include the shortest path and answers to questions in the writeup!

Writeup (20 points)

You will include a written report with your submission detailing important details about your implementation, as well as the results of any analyses requested in the assignment. The report must be complete and clear. A few key points to remember:

- Complete: the report does not need to be long, but should include everything that was requested.
- Clear: your grammar should be correct, your graphics should be clearly labeled and easy to read.

- Concise: I sometimes print out reports to ease grading, don't make figures larger than they need to be. Graphics and text should be large enough to get the point across, but not much larger.
- Credit (partial): if you are not able to get something working, or unable to generate a particular figure, explain why in your report. If you don't explain, I can't give partial credit.

Bonus: Prim's Algorithm (10 points)

Prim's algorithm is commonly used to construct minimum spanning trees. For extra credit, implement Prim's algorithm on the Romania graph. All rules from previous parts apply. Once you've implemented this algorithm, print the spanning tree by printing the full predecessor structure.