# CSCI 311 - Fall 2022:
# Programming Project: Connecting a City

Brian King Edward Talmage

Due: **On gradescope** by Friday, December 2, 2022
Based in part on problems in CLRS and other sources

## Instructions

**This is a group project. You (will) have received an Overleaf share invitation to your group's report, and can find group members there. You will collectively submit one version of your solution. Be certain that the student who uploads your solution then adds all group members to the submission on gradescope.**

## 1 Overview

While they may seem abstract and meaningless when presented in class, graphs are fundamental to many, if not most, of the real-world problems we solve with computing. As an example of using a graph to solve a physical problem, and an application of a few different graph algorithms we've seen, we want you to consider the following scenario. Suppose you have been hired by a city that wants to update Internet service connectivity for its residents. To fully connect the city, they want to provide Internet service along every road, meaning, the cable must follow every existing road. However, the city has a limited budget. They have hired you to come up with a plan that fully connects the city along its roads, while minimizing the amount of cable required to achieve this feat.

The city will provide you with a file that describes the network of roads. This is easily represented as a graph, where nodes are intersections connecting all of their roads , and edges represents the roads between intersections. Each edge will have a weight representing the distance between the vertices. For example, consider the City of San Joaquin, California, USA (courtesy of `https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm` below. This city has $18,263$ intersections (nodes) connecting 23,874 roads (edges). The resulting network is visualized below:

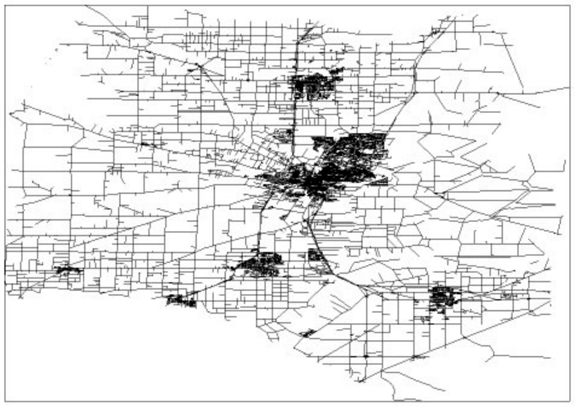

Figure 1: City of San Joaquin County Road Network

At this stage of the course, you have seen a plethora of different algorithms and structures for solving a wide range of problems. Moreover, you have observed how not all algorithms (and their underlying data structures) are to be treated equally!

For this problem, on the surface it would appear that your primary task, from the client's perspective, is to come up with a solution that minimizes the amount of cable you need to use to fully connect the city. However, from the student perspective, that is not enough. As upper-level computer scientists, we also care about understanding the strengths and weaknesses of different algorithms that could be used to solve a given problem. Not all solutions are created equal, especially when it comes to making judicious use of computational resources at hand (i.e. time and space.) Therefore, you will implement more than one option and analyze the (real) time each of your implementations takes to arrive at a solution.

## 2 Your Task

**Objective 1:** Given a file storing a graph representation you are to find the cheapest way you can to connect every node. That is, given graph $G = (V, E)$, you should return a subset of edges $E'$ such that $(V, E')$ is a connected graph. You are to optimize in two dimensions: The sum of the weights of all edges in $E'$ should be as small as possible, but your code should also be as fast as possible.

**Objective 2:** You are to compare the time of different implementations, both to each other and to their asymptotic complexities. To this end, you need to have a second implementation in your code. This second implementation must make a different choice for at least one of

- graph representation,
- algorithm used for finding minimum length of cable to connect the graph, or
- data structure used for above algorithm.

You may also change other implementation decisions. The more you change, the more points you will receive. Run the two versions of your solution on several datasets and plot their times. In your report, show the graph of these data points and discuss the asymptotic and measured complexities of your algorithms.

### 2.1 File Format

The format we will use for storing graphs in this project stores edges explicitly and vertices implicitly. This does not allow any vertices which have no incident edges, but such a vertex would be unconnectable, so we will assume there are none in our graph.

Each line in the data file will represent one edge in the graph (except for comments, which are lines starting with `#`). It stores four pieces of information about each edge, separated by spaces:

1. Edge ID: A unique integer for each edge.
2. Start Node ID: The integer ID of the first vertex the edge touches.
3. End Node ID: The integer ID of the second vertex the edge touches.
4. Length: The floating-point length of the edge.

You should read in an input file in this format, build a representation of the graph, compute your shortest connecting set of edges, and write it out as a file in the same format.

Several sample datasets are available at `https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm` [1]. Use files named "XXX Network's Edges (Edge ID, Start Node ID, End Node ID, L2 Distance)". The one pictured above is map 4, City of San Joaquin Valley, and should be a good starting test.

## 3 Deliverables

You may use any language you want, as long as I can run it from a bash terminal on a Bucknell lab machine. I plan to do this in class, so make sure your code works. Java, python, and the C family would all be fine, among others.

- You will submit all source code for both of your implementations, along with a plain-text README describing how I can compile, run, and interact with your code. The README should be very concise and specific.
- You will submit a short (try to keep it no more than 1-2 pages) description of the algorithmic structure of your project, including what data structures and implementations you use. Include asymptotic runtimes, a plot of your code's real-world runtimes on inputs of different sizes, and a discussion of the differences between your algorithms and their runtimes, expected and measured. This should be a PDF typeset in LaTeX. Do not submit images separately.
- We will have a competition between different groups' submissions in class on Monday, December 5. For this to work, and for me to test the correctness of your solutions, it is **imperative** that your solutions have a standard interface. I should be able to call your code from a bash prompt (or script!) as `languageCommand programName inputfile outputfile` or `./programName inputfile outputfile`, where
  - `languageCommand` is "python", "java", or similar,
  - `programName` is your executable (If you're using python, this is just your code file. If you're using Java, C/C++, etc., this will be the output of the compiler.), and

  – `inputfile` contains the graph you need to connect, and your code should write its set of output edges to `outputfile`.

Your code should not print huge amounts of data to the terminal. Instead, your code should write just the edges along which your code chooses to lay cable to `outputfile`. I will run a script to calculate the weight of your solution. I will also use a script to time your solution from call to return, so you do not need to worry about instrumenting your code.

### 3.1 A Note on Performance

As you may know, not all programming languages are equally efficient. python is notoriously slower than C, for example. We typically ignore these differences in our class, as they are (usually) just constant-factor differences and we are more interested in the inherent complexity of the algorithms we design to solve problems. In fact, even for this project, using the same language for your two implementations should give you a decent performance comparison. For that reason **use the same language for both of your implementations**. Further, to ensure you're practicing the topics from the course, **do not use libraries for your data structures**. Basic types (lists, etc.) are fine, but if you need a structure which is not built-in, write it yourself.

The only issue will be when we put different groups' projects head-to-head. Here, it is very possible that there will be an inherent advantage for some languages over others. First, don't worry about this too much–the competition is a small number of bonus points and you can also get points elsewhere. Second, it may be worth considering using a faster language for your entire project. But make sure the **entire group** is comfortable in the language you choose. If someone doesn't have experience in a particular language, **do not use it**.

## 4 Workflow

I suggest you start with coding your file handling (input and output) and data structures. Be sure to start the writeup early, as you need accurate descriptions of your algorithmic runtimes. Once you have file handling and data structures (within the first week!), you can assemble them and have a working prototype which returns the entire graph. Then you can get started on interesting features and a second implementation, particularly as we cover relevant material in class.

## 5 Grading

You will be graded on the following:

- README present and clear (1pt)
- Code runs (1pt)
- Data Structures (1pt per implementation)
- Functionality: Code generates a subset of edges (1pt per implementation)
- Correctness: Code generates a connected and lightweight subset of edges (3pts per implementation (1 for connectedness, up to 2 for weight))
- Runtime: Both asymptotic and real-world improvements (2pts)
- Report (4pts: 2 for overall structure, 2 for algorithmic comparison)
- Bonus: Cool features (up to 3pts)
- Bonus: Competition (up to 2pts)

## References

[1] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In Claudia Bauzer Medeiros, Max J. Egenhofer, and Elisa Bertino, editors, *Advances in Spatial and Temporal Databases, 9th International Symposium, SSTD 2005, Angra dos Reis, Brazil, August 22-24, 2005, Proceedings*, volume 3633 of *Lecture Notes in Computer Science*, pages 273–290. Springer, 2005.