# ECE241 PROJECT 2: Thinking in Graphs

# Due: Nov 11, 2021, 11:50pm on Gradescope

## Introduction

Computer networks have become an integral part of modern infrastructure. Routing, which is a critical component of the networking process decides the paths on which packets traverse from source to the destination. To make the network scalable, paths to a particular destination are determined dynamically by routing protocols. Open Shortest Path First (OSPF) is one of the most widely used intro-domain routing protocols. As the name suggests, OSPF selects the path based on the cost to destinations. Each link in OSPF is assigned a link weight and the cost to a destination following a particular path is defined by the sum of all link weights among the paths. In this project, you'll use a graph to model a network of a set of Internet Service Providers (ISPs) and write algorithms to analyze the forwarding behaviors of routers, i.e., paths to destinations.

In the project, you will build a graph that represents the network topology in ISPs. There is an edge between two routers if they are physically connected. The weight of the edge is the weight for that link, which is used by routing algorithms to determine the best path when forwarding packets. Using this graph, you will be able to identify the forwarding path within the network. You can further analyze properties of the forwarding behavior using the graph.

## Dataset

In this project, we will use network topologies inferred by Rocketfuel which is a research project carried at the University of Washington. You can access their project website at: https://research.cs.washington.edu/networking/rocketfuel/. We will use the dataset: "Backbone topologies annotated with inferred weights and link latencies" from this project. We have provided a parsed version of this dataset, you can find the dataset on moodle. (Please use the dataset we provided, do **NOT** use their original dataset.)

In this dataset, you can see the topology of 6 ISP networks in 6 csv files. Each file stores the ~~inferred~~ link weights for one ISP network.

## Task Overview

In this project, you need to accomplish the following specific tasks.

1. Read the inferred link weight from file and build the ISP network topology graph.

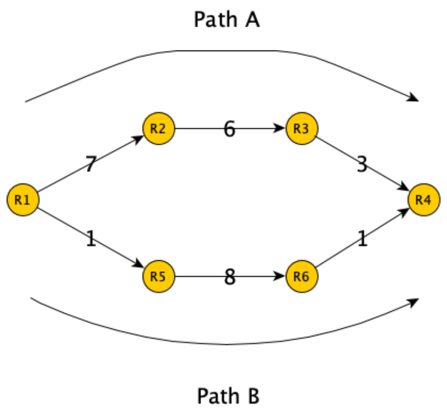
- Create a class named ***ISPNetwork***.
- Create a method ***buildGraph(self, filename: str) -> None*** to load the data from a file (with the input filename as a string). You don't need to return anything but store the graph in a class variable ***self.network***.
- You have to import and use the ***Graph*** and ***Vertex*** classes provided in class. You can add variables or functions to help you finish this project. But you **must NOT** change any existing functions/variables.
  https://gist.github.com/mikezink/09ddf137dd9854c0f3fd01c6212ac4d3
- Each row in the file represents a link between two routers. Each row consists of three elements separated by comma ",". The first two elements are the names of routers and the third element is the corresponding (inferred) link weight.
- You can ignore the meaning of the names of routers, the same name represents the same router.

2. Write a function to determine if a path exists between any two routers.
   - Create a method ***pathExist(self, router1: str, router2: str) -> bool*** to determine whether a path exists between router1 and router2.
   - The path is considered not exist if any of the router doesn't exist.
   - Hint: You can achieve this goal by taking advantage of BFS or DFS algorithm.
3. Build a sub-graph by generating a minimum spanning tree (based on cost) from the destination routers in the network topology.
   - Create a method ***buildMST(self) -> None***. You don't need to return anything but store the sub-graph in a class variable ***self.MST***.
4. Find the shortest path between two routers along the minimum spanning tree.
   - Create a method ***findPath(self, router1:str, router2:str) -> str***
   - The returned path should be a sequence of router names connected by arrows ("->"). For example, the output path two routers R1 and R3 should be a string like "R1 -> R2 -> R3" where R1, R2 and R3 are names of routers.
   - Return "path not exist" if any of the router or the path does not exist.
   - Note: You must use MST you build in Task3, directly find the path from original graph will result in 0 points in this task!
5. Find the forwarding path between any two routers in the original network topology graph. To forward packets between any two routers, it's better to use the path with minimal cost. Therefore, the

forwarding path should **ONLY consider the path with minimal cost,** return the minimal cost as well.

- Create a method ***findForwardingPath(self, router1:str, router2:str) -> str***
- The output should be a path followed by a bracket with cost (sum of link weights) in it. For example, the output for two routers R1 and R3, should be "R1 -> R2 -> R3 (397.00)".
- Return "path not exist" if any of the router or the path does not exist.
- Note: You must use the original graph, find path from the MST will result in 0 points in this task!

6. Suppose another routing algorithm picks the paths based on the maximum link weight, i.e., picks the path with the smallest maximum link weight along the path. Write a function to find the path between two routers.
   - For example, there are two paths from R1 to R4 with link weight labeled on the edge:



The maximum link weight for Path A is 7 and the maximum link weight for Path B is 8, therefore, the Algorithm will pick Path A.

   - Create a method ***findPathMaxWeight(self, router1:str, router2:str) -> str***
   - The returned path should be a sequence of router names connected by arrows (->). For example, the output path two routers R1 and R3 should be a string like "R1 -> R2 -> R3" where R1, R2 and R3 are names of routers.
   - Hint: You can think about modifying Dijkstra's Algorithm.

## Hints and suggestions

1. Try to debug your program using a small subset of links first. Only use the big, complete file once you feel confident your program works.

2. You can write your code based on any ISP topology file downloaded from the dataset. However, you have to make sure your code works for any topology in the dataset. The autograder might use any topology in the dataset to evaluate your code.
3. Try to get portions of the program to work step-by-step. See if you get the shortest path function to work first and then build the MST. You can perform the final two tasks at the very end.
4. Start early. Even though we provide a lot of the basic code from the lectures, it might take a while to write and debug these methods.

**What to submit**:

Please submit all .py files for your project to Gradescope. All code should be well commented. Note: Don't forget to upload the **Graph.py** file even though you don't make any change to it. Do **NOT** upload the dataset!

*Reminder:* The course honesty policy requires you to write all code yourself, except for the code that we give to you. Your submitted code will be compared with all other submitted code for the course to identify similarities. Note that our checking program is not confused by changed variable or method names.

**Grading:**

- Code works (90%)
- Comments**,** Program structure, Readability (10%)