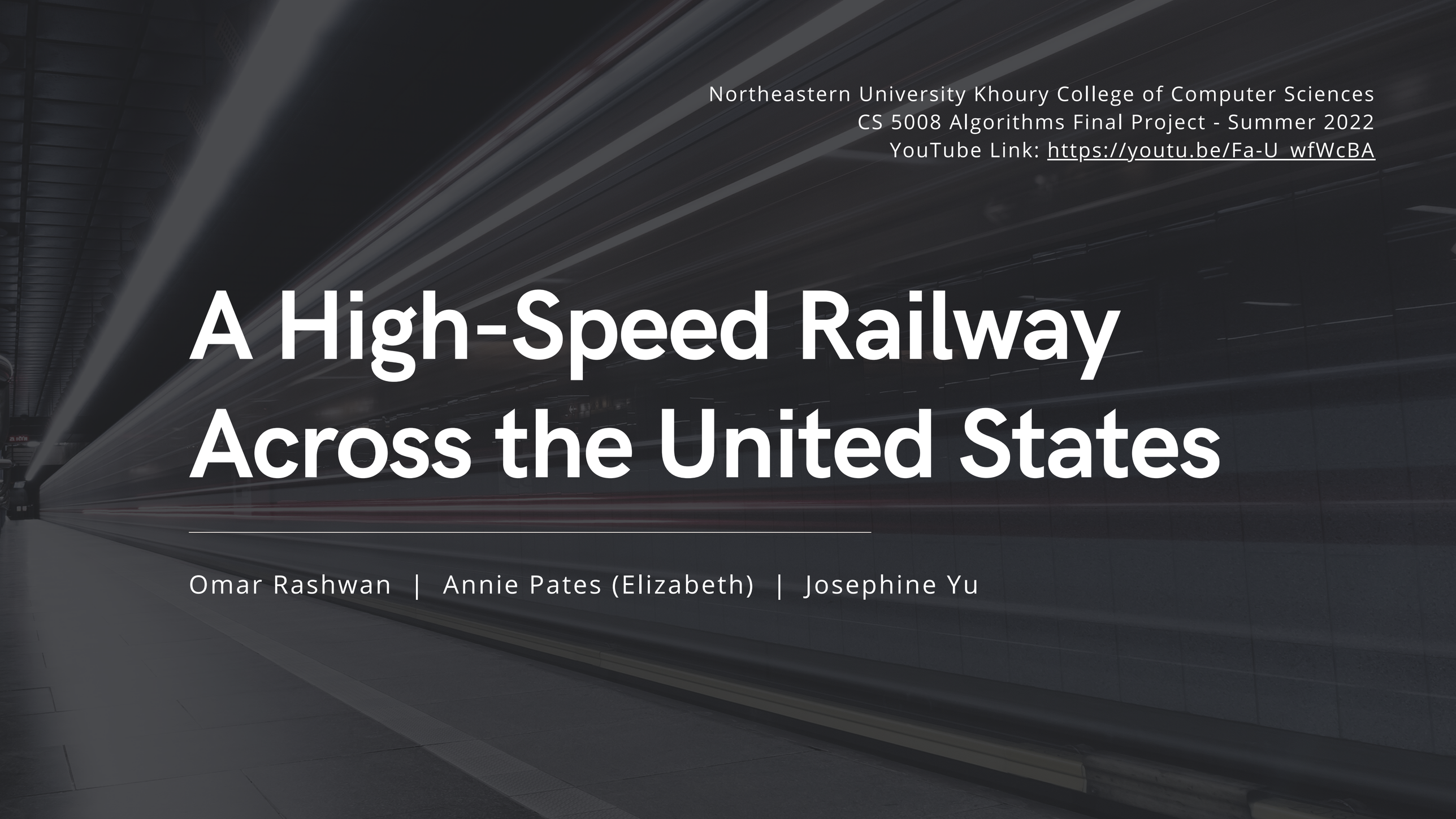


Northeastern University Khoury College of Computer Sciences
CS 5008 Algorithms Final Project - Summer 2022
YouTube Link: https://youtu.be/Fa-U_wfWcBA

A High-Speed Railway Across the United States

Omar Rashwan | Annie Pates (Elizabeth) | Josephine Yu

Table of Contents

00

Project Introduction

Clearly defined question and relevance to our team

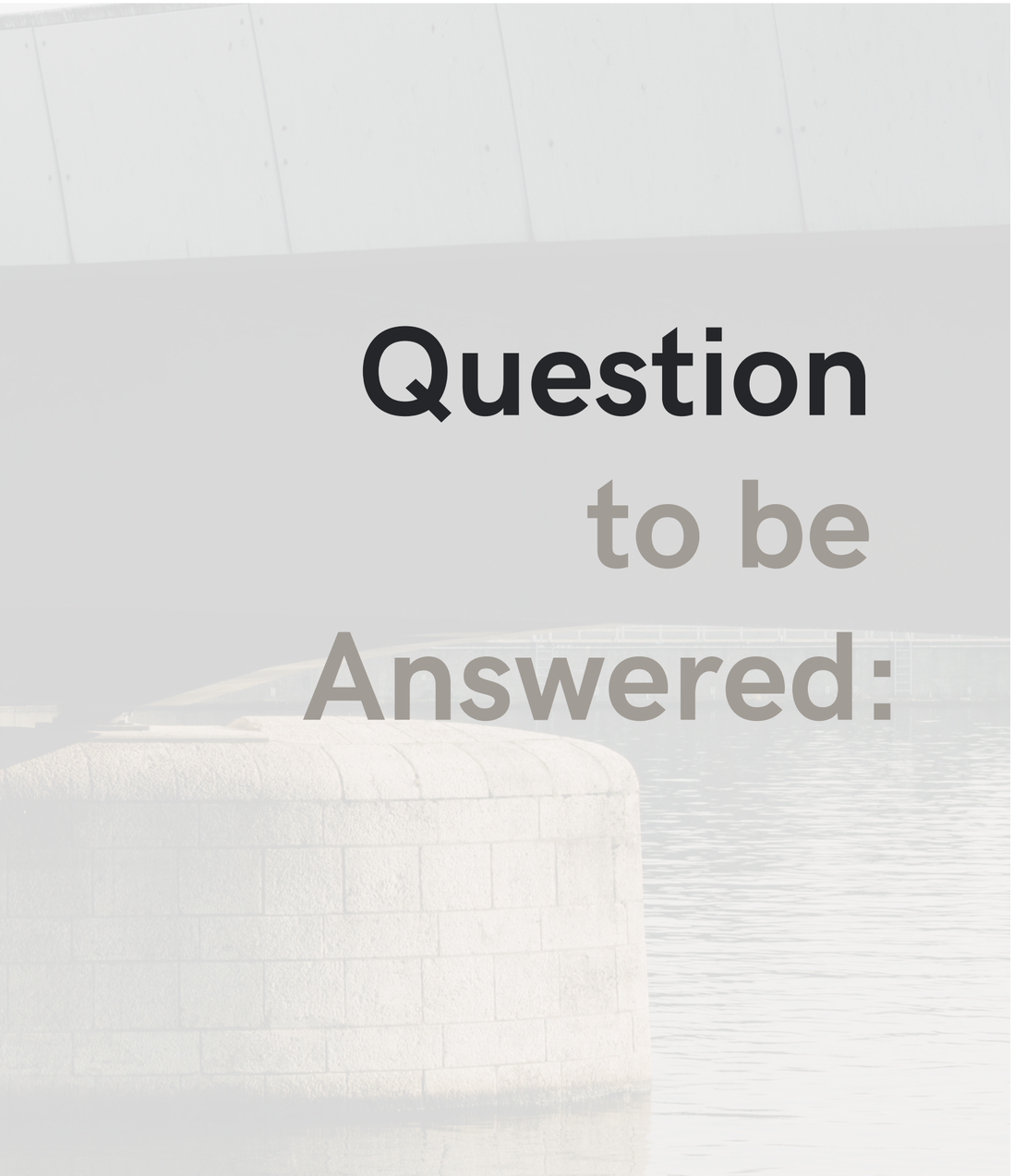
Initial Inspiration

With fuel prices increasing astronomically in the past few months, travel has become more cumbersome.

Each of our group members has been affected by this issue personally lately, which prompted us to brainstorm.

This led to a discussion whether the United States had a transcontinental, high-speed railroad that facilitates an easier and relatively economical travel.

We began brainstorming the best methodology to apply algorithms we have learnt in this class and potential roadblocks that we may face through our theoretical railway building journey.



Question to be Answered:

What is the most efficient way
to build a high-speed railway
to connect the most populous cities
in the contiguous United States?

Relevance to Each Team Member

Omar Rashwan

Ever since watching Thomas the Tank Engine as a young boy, I have been fascinated with rail transit as a means of transportation. This interest in trains has only grown as I learned more about the efficiency of rail versus other means of transport such as road and air travel …

Annie Pates (Elizabeth)

This year, I have weddings in Massachusetts, Rhode Island, Ohio, Minnesota, and Texas as well as trips to Wyoming and Florida. Additionally, my friends who are getting married in the bride's hometown in Texas are committed to having their dog participate in the ceremony …

Josephine Yu

For upcoming travels in Canada, I have been canceling flights and opting for travel via train instead. In Canada, the transcontinental train goes from the west coast in Vancouver to the east coast in Halifax, it is also heavily subsidized by the government…

01

Problem Analysis

How we framed the problem and how we will solve it

Problem Analysis: Assumptions

Parameters given the short time frame

- **Only examining the most populous city in each of the contiguous US states**
  - ie. not considering Hawaii, Alaska at this time
- **Caveat: paths in MST may not be the most effective way to get from one city to another**
  - Used Prim's algorithm to create minimum weight spanning tree
  - Mitigated caveat by applying Floyd-Warshall once we'd created it
- **Ignoring terrain complexities that come with going around/over/through geographical features**
  - ie. Lakes and mountains
- **Not factoring cost in the implementation which could have skewed the paths**

02

Solution Approach

Prim's Algorithm, Minimum-Spanning Tree, Floyd-Warshall Algorithm

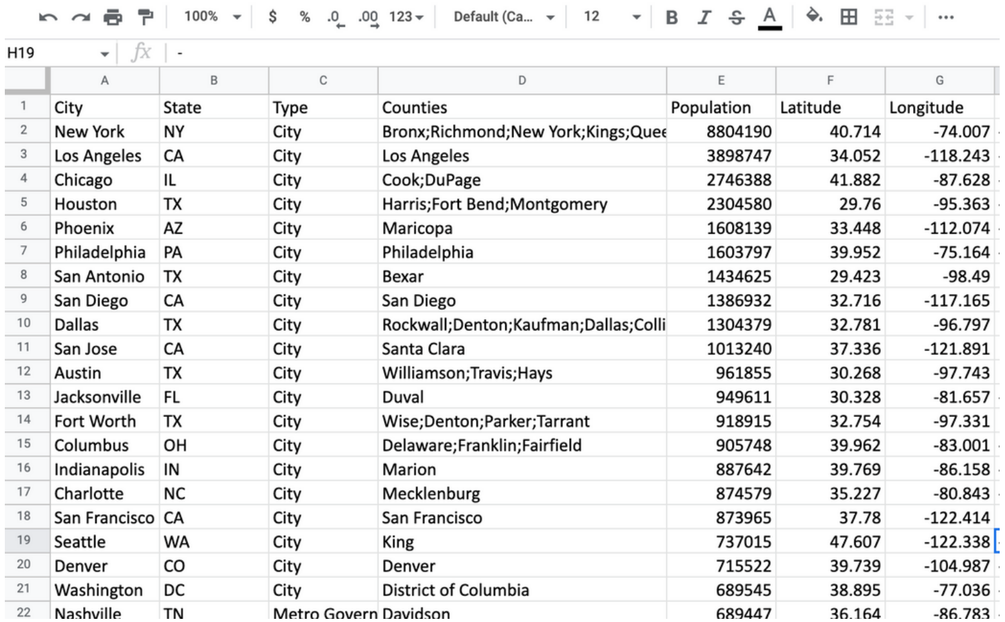
Methods Used to Gather Data

- Wikipedia dataset on the top 330 most populous cities in the United States including:
  - 2021 census information
  - Land area
  - Population density
  - Location longitudes and latitudes

2021 rank	City	State[c]	2021 estimate	2020 census	Change	2020 land area		2020 population density		Location
1	*New York*[d]	New York	8,467,513	8,804,190	−3.82%	300.5 sq mi	778.3 km²	29,298/sq mi	11,312/km²	40.66°N 73.93°W
2	*Los Angeles*	California	3,849,297	3,898,747	−1.27%	469.5 sq mi	1,216.0 km²	8,304/sq mi	3,206/km²	34.01°N 118.41°W
3	*Chicago*	Illinois	2,696,555	2,746,388	−1.81%	227.7 sq mi	589.7 km²	12,061/sq mi	4,657/km²	41.83°N 87.68°W
4	*Houston*	Texas	2,288,250	2,304,580	−0.71%	640.4 sq mi	1,658.6 km²	3,599/sq mi	1,390/km²	29.78°N 95.39°W
5	***Phoenix***	Arizona	1,624,569	1,608,139	+1.02%	518.0 sq mi	1,341.6 km²	3,105/sq mi	1,199/km²	33.57°N 112.09°W
6	*Philadelphia*[e]	Pennsylvania	1,576,251	1,603,797	−1.72%	134.4 sq mi	348.1 km²	11,933/sq mi	4,607/km²	40.00°N 75.13°W
7	San Antonio	Texas	1,451,853	1,434,625	+1.20%	498.8 sq mi	1,291.9 km²	2,876/sq mi	1,110/km²	29.47°N 98.52°W
8	San Diego	California	1,381,611	1,386,932	−0.38%	325.9 sq mi	844.1 km²	4,256/sq mi	1,643/km²	32.81°N 117.13°W
9	Dallas	Texas	1,288,457	1,304,379	−1.22%	339.6 sq mi	879.6 km²	3,841/sq mi	1,483/km²	32.79°N 96.76°W
10	San Jose	California	983,489	1,013,240	−2.94%	178.3 sq mi	461.8 km²	5,683/sq mi	2,194/km²	37.29°N 121.81°W
11	**Austin**	Texas	964,177	961,855	+0.24%	319.9 sq mi	828.5 km²	3,007/sq mi	1,161/km²	30.30°N 97.75°W
12	*Jacksonville*[f]	Florida	954,614	949,611	+0.53%	747.3 sq mi	1,935.5 km²	1,271/sq mi	491/km²	30.33°N 81.66°W
13	Fort Worth	Texas	935,508	918,915	+1.81%	342.9 sq mi	888.1 km²	2,646/sq mi	1,022/km²	32.78°N 97.34°W

https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population

- Kaggle dataset in .csv format



	A	B	C	D	E	F	G
1	City	State	Type	Counties	Population	Latitude	Longitude
2	New York	NY	City	Bronx;Richmond;New York;Kings;Quee	8804190	40.714	-74.007
3	Los Angeles	CA	City	Los Angeles	3898747	34.052	-118.243
4	Chicago	IL	City	Cook;DuPage	2746388	41.882	-87.628
5	Houston	TX	City	Harris;Fort Bend;Montgomery	2304580	29.76	-95.363
6	Phoenix	AZ	City	Maricopa	1608139	33.448	-112.074
7	Philadelphia	PA	City	Philadelphia	1603797	39.952	-75.164
8	San Antonio	TX	City	Bexar	1434625	29.423	-98.49
9	San Diego	CA	City	San Diego	1386932	32.716	-117.165
10	Dallas	TX	City	Rockwall;Denton;Kaufman;Dallas;Colli	1304379	32.781	-96.797
11	San Jose	CA	City	Santa Clara	1013240	37.336	-121.891
12	Austin	TX	City	Williamson;Travis;Hays	961855	30.268	-97.743
13	Jacksonville	FL	City	Duval	949611	30.328	-81.657
14	Fort Worth	TX	City	Wise;Denton;Parker;Tarrant	918915	32.754	-97.331
15	Columbus	OH	City	Delaware;Franklin;Fairfield	905748	39.962	-83.001
16	Indianapolis	IN	City	Marion	887642	39.769	-86.158
17	Charlotte	NC	City	Mecklenburg	874579	35.227	-80.843
18	San Francisco	CA	City	San Francisco	873965	37.78	-122.414
19	Seattle	WA	City	King	737015	47.607	-122.338
20	Denver	CO	City	Denver	715522	39.739	-104.987
21	Washington	DC	City	District of Columbia	689545	38.895	-77.036
22	Nashville	TN	Metro Govern	Davidson	689447	36.164	-86.783

https://www.kaggle.com/datasets/axeltorbenson/us-cities-by-population-top-330

File Reader in Python

- Coded a file reader in Python to iterate through the data of the 330 cities
- Sorted the data by population in decreasing order *(Module 0 - Sorting)*
- Appended data into a nested array initialized as “top48”
  - sorted, most populous 48 cities used to create the graph in an adjacency matrix *(Module 5- Adjacency Matrices and List)*
- Results were outputted to a file, “top48.py”

```
City,State,Type,Counties,Population,Latitude,Longitude
New York,NY,City,Bronx;Richmond;New York;Kings;Queens,8804190,40.714,-74.007
Los Angeles,CA,City,Los Angeles,3898747,34.052,-118.243
Chicago,IL,City,Cook;DuPage,2746388,41.882,-87.628
Houston,TX,City,Harris;Fort Bend;Montgomery,2304580,29.76,-95.363
Phoenix,AZ,City,Maricopa,1608139,33.448,-112.074
Philadelphia,PA,City,Philadelphia,1603797,39.952,-75.164
San Antonio,TX,City,Bexar,1434625,29.423,-98.49
San Diego,CA,City,San Diego,1386932,32.716,-117.165
Dallas,TX,City,Rockwall;Denton;Kaufman;Dallas;Collin,1304379,32.781,-96.797
San Jose,CA,City,Santa Clara,1013240,37.336,-121.891
Austin,TX,City,Williamson;Travis;Hays,961855,30.268,-97.743
Jacksonville,FL,City,Duval,949611,30.328,-81.657
Fort Worth,TX,City,Wise;Denton;Parker;Tarrant,918915,32.754,-97.331
Columbus,OH,City,Delaware;Franklin;Fairfield,905748,39.962,-83.001
Indianapolis,IN,City,Marion,887642,39.769,-86.158
Charlotte,NC,City,Mecklenburg,874579,35.227,-80.843
San Francisco,CA,City,San Francisco,873965,37.78,-122.414
Seattle,WA,City,King,737015,47.607,-122.338
Denver,CO,City,Denver,715522,39.739,-104.987
Washington,DC,City,District of Columbia,689545,38.895,-77.036
Nashville,TN,Metro Government,Davidson,689447,36.164,-86.783
Oklahoma City,OK,City,Canadian;Cleveland;Oklahoma,681054,35.468,-97.516
El Paso,TX,City,El Paso,678815,31.76,-106.488
```

.csv file

>>>

```python
def main():
    filename = "us2021census.csv"
    with open(filename, mode='r') as f:
        header = f.readline()
        data = f.readlines()

    states = {"AK", "HI", "DC"}
    top48 = []
    for i in range(len(data)):
        data[i] = data[i].strip('\n')
        data[i] = data[i].split(',')
        data[i][4] = int(data[i][4])
        data[i][5], data[i][6] = float(data[i][5]), float(data[i][6]
        lst = [data[i][0], data[i][5], data[i][6]]
        if data[i][1] not in states:
            states.add(data[i][1])
            top48.append(lst)

    n = len(top48)

    adj_matrix = [[0 for x in range(n)] for y in range(n)]

    for i in range(n):
        for j in range(i, n):
            city1 = top48[i][1], top48[i][2]
            city2 = top48[j][1], top48[j][2]
            distance = dist(city1, city2)
            adj_matrix[i][j] = distance
            adj_matrix[j][i] = distance

    output = "top48.py"

    with open(output, mode='w') as o:
        o.write("top48 = " + str(top48) + '\n')
        o.write("g = " + str(adj_matrix))
```

csv_reader.py

>>>

```
top48 = [['New York', 40.714, -74.007], ['Los Angeles', 34.052, -118.243], ['Chicago', 41.882, -87.628], ['
g = [[0.0, 3086.703983270828, 943.298051129652, 1656.0989717924465, 2674.0426730785352, 95.59159990815074,
```

top.48.py

Creating the Graph

Began with a graph structure:

- Vertices = cities
- Edges = railway path
- Weighted edge = distance between two vertices

Building our minimum spanning tree:

- Starting vertex = most populous city from our “top48” array
- Applied Prim’s algorithm *(Module 7 Greedy Algorithms 2)*
- Calculated the lowest weighted edge at each step
- Runtime complexity = O((V+E) log(V))

```python
def prim(graph):
    n = len(graph)
    included = {0}
    mst = [[0 for x in range(n)] for y in range(n)]

    while len(included) < n:
        minimum = 999999999
        x, y = 0, 0
        for u in included:
            for v in range(n):
                if v not in included and graph[u][v] < minimum:
                    minimum = graph[u][v]
                    x, y = u, v
        included.add(y)
        mst[x][y] = minimum
        mst[y][x] = minimum

    return mst


def main():

    mst = prim(g)

    data = []

    for i in range(len(top48)):
        data.append((top48[i][0], top48[i][1], top48[i][2]))


    visualize(mst, data)
```

mst_build.py

Graph Visualization

Utilized Python library packages: NetworkX, Matplotlib and Basemap

- Graphed our vertices and edges with relation to the map of the United States
- From the adjacency matrix, [n][0] = city_name, [n][1] = latitude, and [n][2] = longitude.

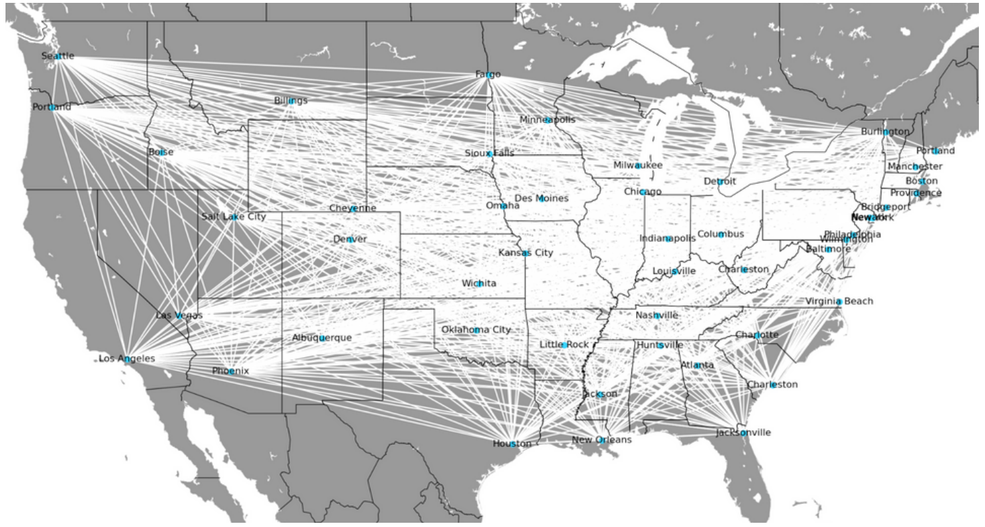
```python
import networkx as nx
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap as Basemap

def visualize(graph, array, x_offset=0, y_offset=0, scale=1):
    '''
    x_offset: shifts all points along the x axis
    y_offset: shifts all points along the y axis
    scale: scales the distance from the origin (0,0) by input
    '''
```

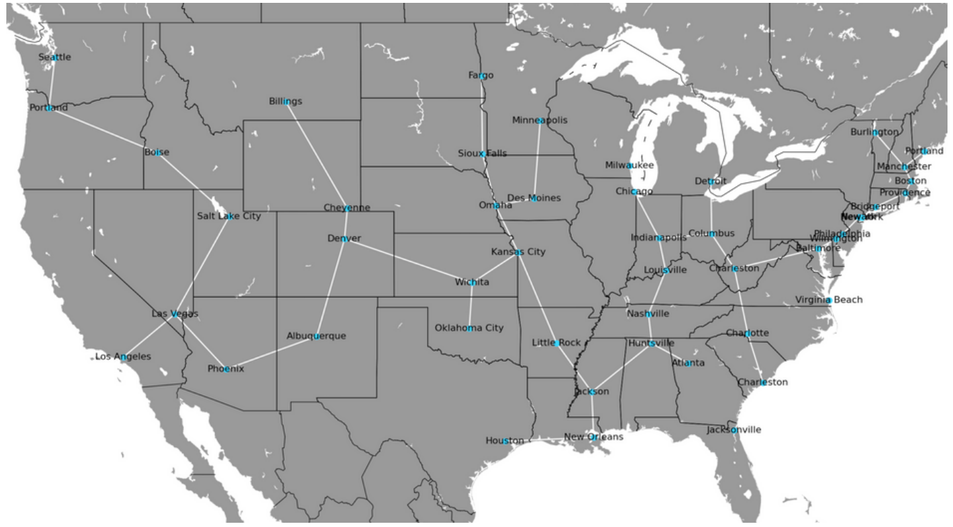
graph_visualizer.py

Total rail track:

Complete graph = 1,445,157 miles



Our railway MST using Prim's algorithm = 9622 miles



Using Floyd-Warshall

Used Floyd-Warshall *(Module 9 Dynamic Programming)* instead of Dijkstra's algorithm *(Module 6 Greedy Algorithms 1)* to determine which additional lines (beyond those on the MST) would improve efficiency of travel

More efficient to find the shortest paths of all pairs of vertices (cities) using matrix multiplication

Our function takes in an adjacency matrix input, calculates the distances between the vertices and returns a matrix of distances

Iteration to find which additional edge reduces the sum of all shortest paths = runtime complexity $O(n^3)$

```python
def floyd_warshall(adj_matrix):
    '''
    Returns a matrix of distances between vertices based on the input adjacency
    matrix. Runs in O(n^3) time.
    '''
    n = len(adj_matrix)
    dist = [[INF for x in range(n)] for y in range(n)]

    for u in range(n):
        dist[u][u] = 0
        for v in range(n):
            if adj_matrix[u][v]:
                dist[u][v] = adj_matrix[u][v]

    for k in range(n):
        for i in range(n):
            for j in range(n):
                dist[i][j] = min(dist[i][j], (dist[i][k] + dist[k][j]))

    return dist
```

analysis_tools.py

Calculating Track Length

```python
INF = 9999999

def track_length(adj_matrix):
    '''
    Returns the total length of track from an input adjacency matrix
    '''
    n = len(adj_matrix)
    track = 0

    for u in range(n):
        for v in range(u + 1, n):
            track += adj_matrix[u][v]

    return track
```

analysis_tools.py

We also utilized a track_length function to:

- Take in the adjacency matrix parameter
- Calculates the track length using two for-loops
- Returns the total length of the track

Runtime complexity is $O(n^2)$

Final Step - Railgorithm

Integrate all functions together:

- input parameters = graph and complete graph
- Stop algorithm once we laid down 20,000 miles of track to be comparable to the current Amtrak rail network of 21,400 miles *(Amtrak Corporate Profile)*.

Runtime complexity = $O(n^5)$

- Two nested loops to call track_length function and applying floyd_warshall to our graph

Total miles of rail track with our High-Speed Rail = 21,106 miles

```python
def railgorithm(graph, complete_graph):
    '''
    O(n^5), lol
    '''
    if len(graph) != len(complete_graph):
        raise ValueError("Graph lengths must be equal")
    n = len(graph)
    curr_sum = track_length(floyd_warshall(graph))



    travel_saved = 0
    x = 0
    y = 0
    for u in range(n):
        for v in range(u + 1, n):
            if graph[u][v] == 0:
                graph[u][v] = complete_graph[u][v]
                delta = curr_sum - track_length(floyd_warshall(graph))
                graph[u][v] = 0
                if delta > travel_saved:
                    travel_saved = delta
                    x,y = u,v

    graph[x][y] = complete_graph[x][y]

    return graph

def main():
    '''
    This took 10 minutes to produce an output with the current parameters
    '''

    mst = prim(g)

    new = mst

    while(track_length(new) < 20000):
        new = railgorithm(mst, g)

    visualize(new, top48)
```

railgorithm.py

03

Implementation Demo

Python Code Demonstration

04

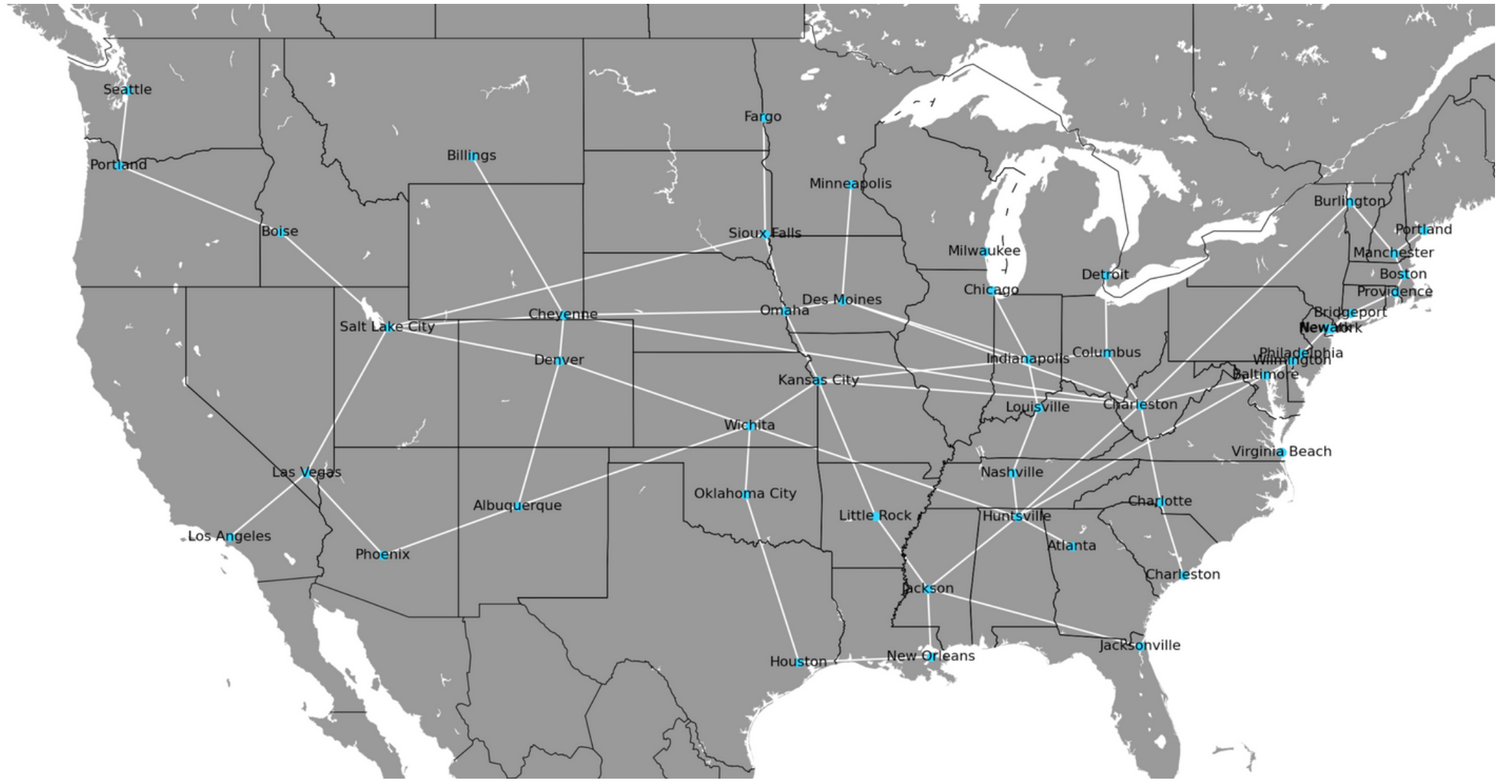
Result & Conclusion

Summary of findings, future considerations, conclusion

Question Being Answered:

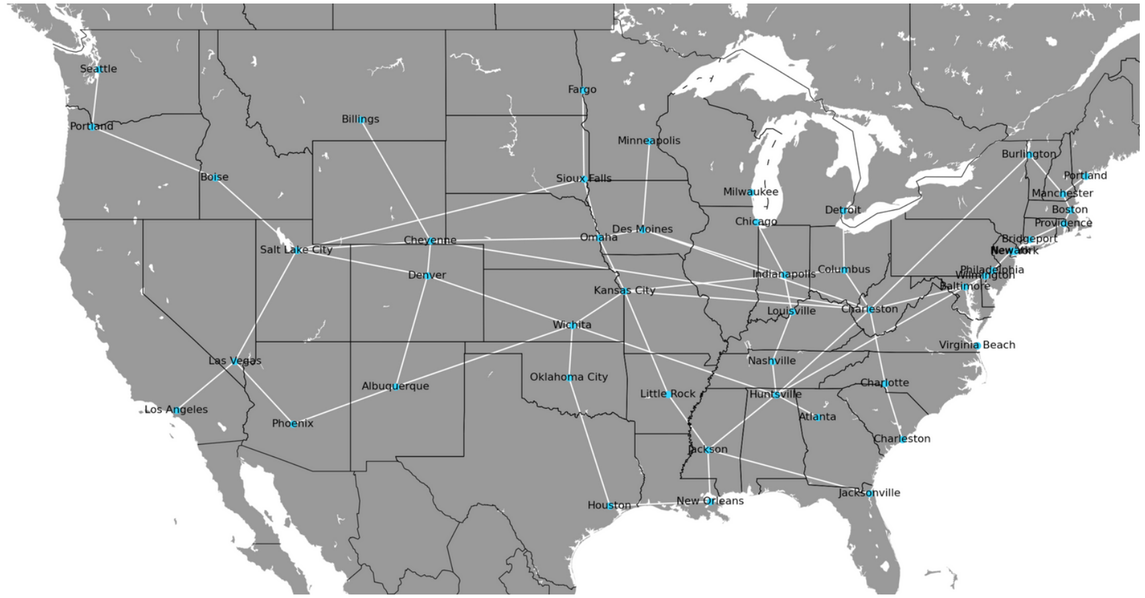
What is the most efficient way to build a high-speed railway to connect the most populous cities in the contiguous United States?

The most efficient way is shown in the graph visualization below with the cities in blue and the rail path in white :



Comparison to Current Amtrak Map

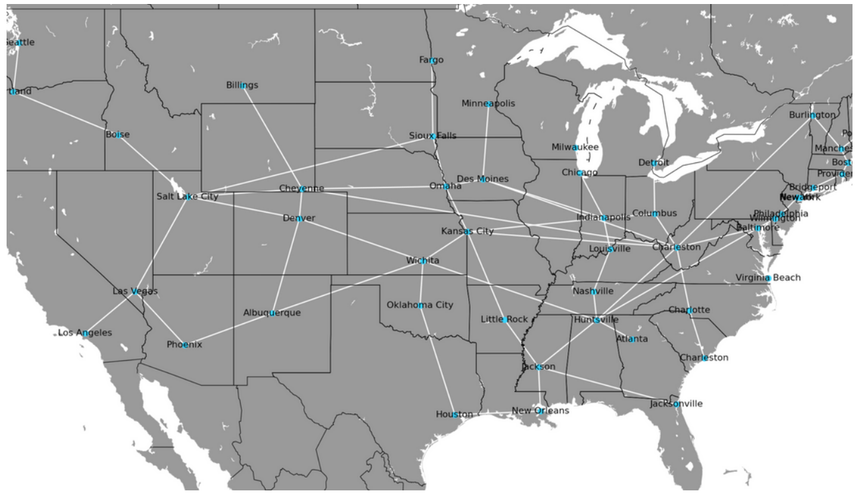
- Laying track like this totaled 21,106 total miles
- This is a dramatic improvement from the 1,445,157 miles in the complete graph, and similar to the 21,400 miles in the current Amtrak map
- We believe that our railway tracks are more efficiently laid out than the current Amtrak



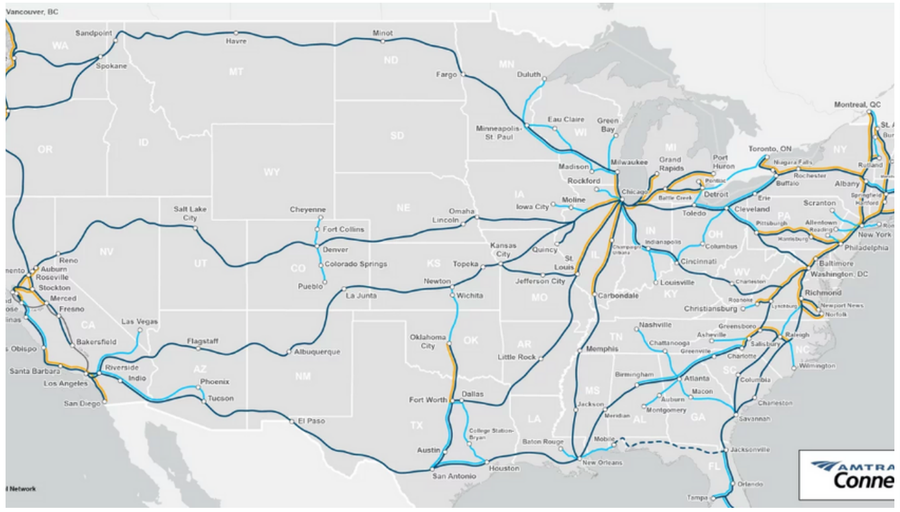
Our High-Speed railgorithm Map

Amtrak Network Railway Map

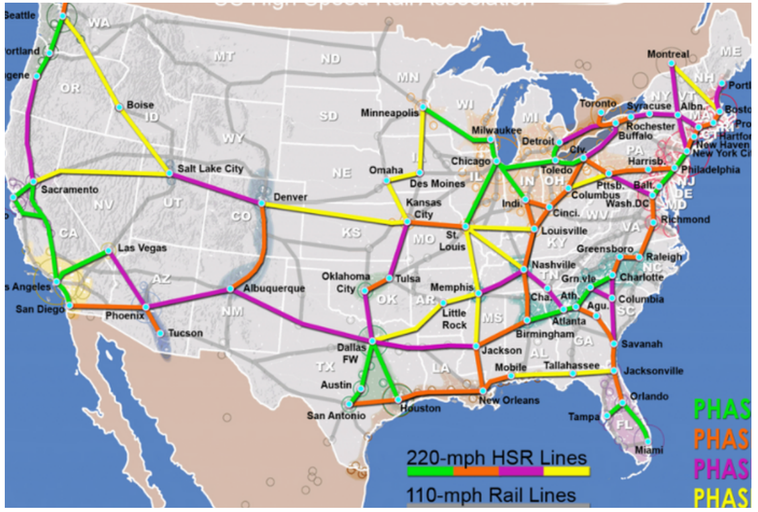
Comparison to Amtrak Proposal



Our High-Speed Railgorithm Map



Amtrak Proposed High Speed Rail Map



US High Speed Rail Association Proposed Map

> Our rail lines are similar to proposed high speed rail maps from Amtrak and the US High Speed Rail Association
>
> We offer more connections between states in the Mountain West, whereas the existing proposals skew more toward connecting a few cities in the West with more populous cities like Seattle and Chicago.

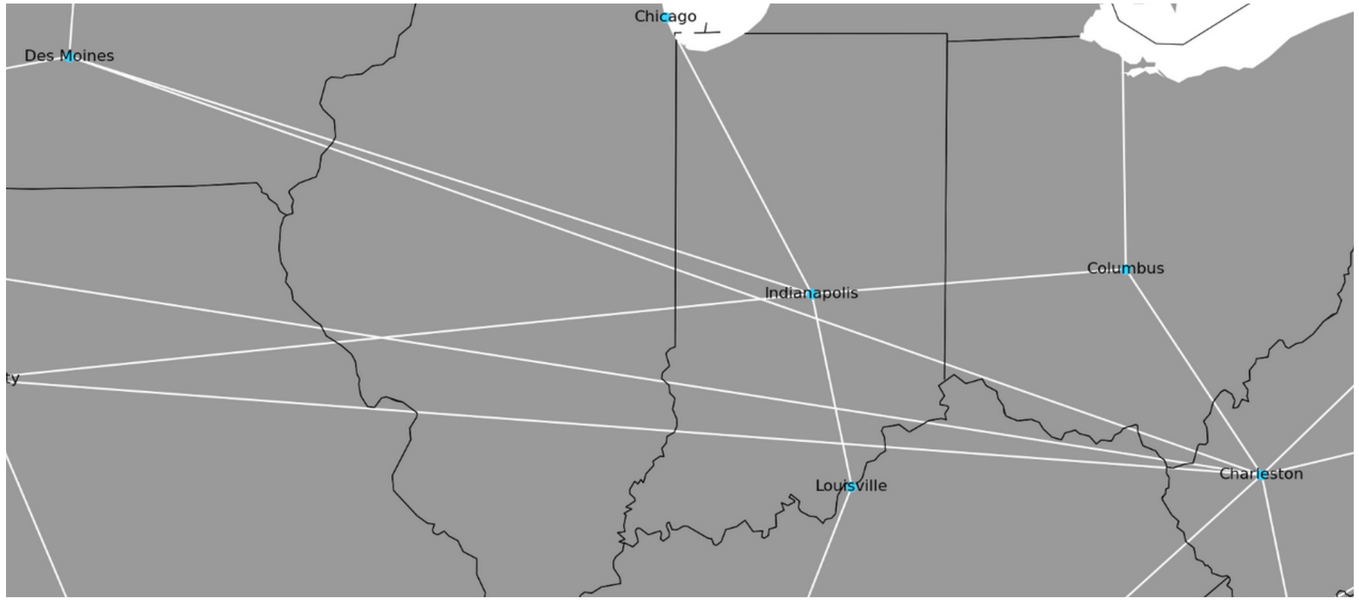
Limitations & Improvements

Floyd-Warshall takes a very long time to run, $O(n^3)$, leading to a total runtime of $O(n^5)$ for our algorithm

- Even though we do not use all the edges in the complete graph, we need to calculate them all in order to decide which ones are the best to add
- In the future, explore other ways of adding edges, perhaps find a clever heuristic we could use with A*

Final graph has busyness around the rust belt

- Several intersecting lines recommended, however, worth doing a cost/benefit analysis before creating all of them.
- Because our algorithm is greedy, there are some inefficiencies such as this edge from Charleston to Des Moines that bypasses Indianapolis, when a more efficient use of track would have been to connect through Indianapolis.



Busyness around the rust belt in the final map

More Considerations

Factor in Terrain

- Potential critical impediment to laying track in certain areas, ie. the Rocky Mountains
- Possible reason why our map differs from the formal proposals

Populous cities not being served by our map

- Create railway connection between Houston - Dallas, Los Angeles - San Jose rather than as many lines in the rust belt
- Advantages to laying track alongside existing railways

Factor in cost

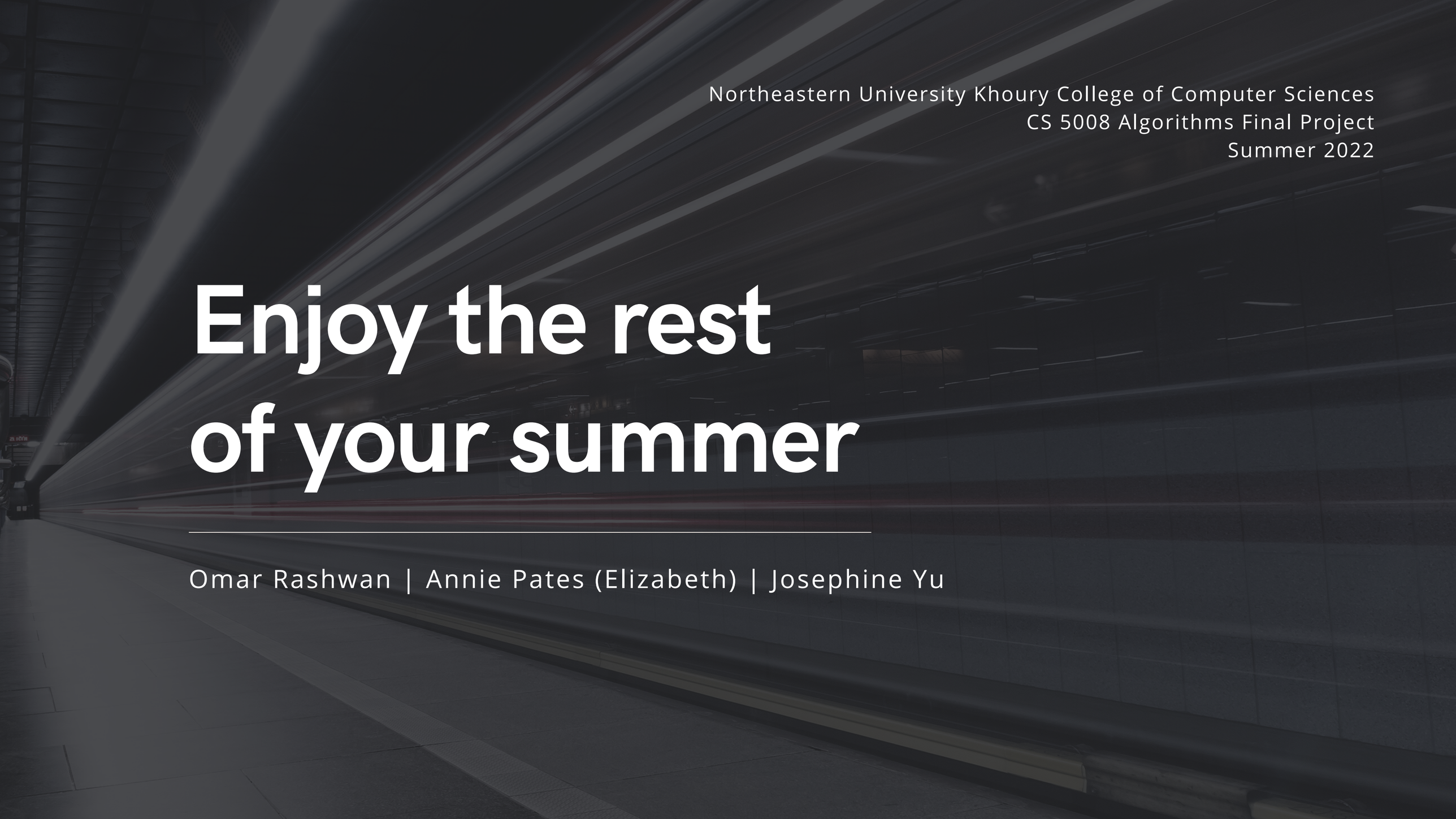
- Consider budget parameters
- Explore cost to connect to Alaska

Ease of construction and ridership demands

- Balancing ridership (connecting most populous cities) with serving those in remote areas (cities in each state)

While our map may not be the final version that should be created,
we believe it could help inform tradeoff decisions in the process of determining
how to build a high speed rail system across the US.

Perhaps our theoretical high-speed railway will come into fruition,
so we can all start taking one soon.



Northeastern University Khoury College of Computer Sciences
CS 5008 Algorithms Final Project
Summer 2022

Enjoy the rest of your summer

Omar Rashwan | Annie Pates (Elizabeth) | Josephine Yu

Final Project Report
"A High Speed Railway Across the United States"

Introduction:

Topic:

With fuel prices increasing astronomically in the past few months, travel by plane, boat and car for example, has become more and more cumbersome. Each of our group members has been affected by this issue in our personal lives lately, which prompted us to start brainstorming around this topic. This led us to discuss whether the United States had a transcontinental, high speed railroad that facilitates an easier and relatively economical travel.

Furthermore, when we began brainstorming the best methodology and potential roadblocks that we may face, we were more and more interested in this particular topic. Perhaps our theoretical railway roadmap may even come to fruition in the future.

Question:

What is the most efficient way to build a high-speed railway to connect the most populous cities in the contiguous United States?

Relevance To Each Team Member:

Omar Rashwan:

Ever since watching Thomas the Tank Engine as a young boy, I have been fascinated with rail transit as a means of transportation. This interest in trains has only grown as I learned more about the efficiency of rail versus other means of transport such as road and air travel. I also grew up in a time where China's rail network grew extensively in both size and speed of transport. Today, China boasts the world's most advanced high-speed rail (HSR) network, accounting for two thirds of the world's high-speed rail. When one considers the similarities between the United States and China in terms of geographic size (3,796,742 sq mi vs. 3,705,407 sq mi) and gross domestic product ($25 trillion vs $20 trillion), it seems strange that China has been able to implement a new rail network so effectively while the United States struggles to maintain the infrastructure it built in the past. This is often attributed to a lack of

political will to build a rail network on a national scale, ostensibly due to the car-centered perspective most Americans have.

Despite this, there are multiple incentives to build a high-speed rail network in the United States. The most talked about reason is also a looming existential threat; global warming. In terms of emissions per passenger, air travel produces the most emissions with 244gCO_2/km. Road travel is just less than half of that with emissions of 102gCO_2/km. However, rail travel is easily the cleanest mode of transportation, with only 28gCO_2 per kilometer traveled. In the United States, there are roughly 3,325 domestic flights each day, which comes out to about 1.2 million domestic flights a year. The emissions saved by switching even half of those domestic flights to rail travel would be substantial to say the least.

High-speed rail could also promote greater interstate commerce and travel. For example, a day trip to New York City from Boston would be infeasible for most people due to it being either a 4 hour drive one-way or a $140 plane ticket. However, if a high-speed rail line were to connect NYC and Boston, the trip would only take about an hour and a half. This would be beneficial for tourism and trade for both cities, and there are many other similar corridors all over the country. Seeing how some of the algorithms we've learned about could be used to develop a high-speed rail network that may hopefully become a reality someday would be very exciting.

Annie Pates:

This year, I have weddings in Massachusetts, Rhode Island, Ohio, Minnesota, and Texas as well as trips to Wyoming and Florida. I love train travel for the easy boarding and uninterrupted work time it offers compared to flying, and would certainly appreciate a cheaper transportation option for some of these destinations, especially one that operates at much higher speeds than current trains. The journey from Wyoming to Minnesota especially has the potential to offer beautiful scenery in addition to the other benefits of train travel. Since the airport in Jackson Hole is quite small, it would also be a useful leg to have connected to a high speed rail system.

Additionally, my friends who are getting married in the bride's hometown in Texas are committed to having their dog participate in the ceremony. They are torn between forcing their precious puppy to fly with the luggage on a plane or beginning their marriage on a 3-day cross-country drive with a carsick dog. While we are unlikely to solve the problem before their October nuptials, it could save future dog-owners from facing a similar quandary!

As high speed rail does seem poised to take off in places like California and Florida, I am curious how the maps that wind up being created will differ from the one we create.

Josephine Yu:

For upcoming travels in Canada, I have been canceling flights and opting for travel via train instead. Wait times at Canadian airports have been making headlines recently. An hour flight from New York was delayed four hours, then two hours sitting on the plane, waiting to take off with no air conditioning. In Canada, the train connects from the west coast in Vancouver to the east coast in Halifax, it is also heavily subsidized by the government making it a viable option to travel. The Canadian government has also been working on laying tracks to operate a high speed frequency rail line train that cuts down a 6 hour car ride between Toronto and Montréal to 40 minutes. Future extensions will include connections to Québec City, Montréal, Ottawa and Toronto. If this form of transportation was available when I was living in Toronto years ago, it would have made traveling to visit family in Montréal much more convenient. Due to the travel time, I was only able to see family by taking advantage of a long weekend. Furthermore, train travel is much more baggage friendly. It requires less personnel to handle bags, minimizing instances of further delays as well as lost baggage.

I am quite interested to see how and whether it is feasible to build a railroad that connects to all the major cities in the contiguous United States. As someone that did not grow up in the US, a nice perk is also learning more of the geography quicker than any class I could have taken. Mapping out the cities will be fun and I am looking forward to discussing the different ways of applying the algorithms with my teammates in a group setting for the first time in this program.

Analysis:

Problem Assumptions:

Given the short time frame, our goal is to apply a shortest-path algorithm we have learned in class. For the sake of simplicity, we will only be examining the most populous city in each of the contiguous US states. We will begin by creating a minimum weight spanning tree of all 48 cities using Prim's algorithm. However, we are keeping in mind that the path between cities on the MST may not (and in many cases

will not) be the most effective way to get from one city to another. We hope to examine and mitigate this by strategically applying the Floyd-Warshall algorithm to add edges. We also chose not to factor in cost budget in the implementation which could have skewed the paths.

Solution Approach:

To ascertain our top 48 most populous cities in the contiguous United States, we used a Kaggle dataset that is based on the Wikipedia dataset with the top 330 most populous cities in the United States[1]. The data included 2021 census information, land area, population density as well as location longitudes and latitudes. The Kaggle dataset included the 2021 population and longitude and latitude coordinates for several thousand of the most populous cities in the US in a .csv file.

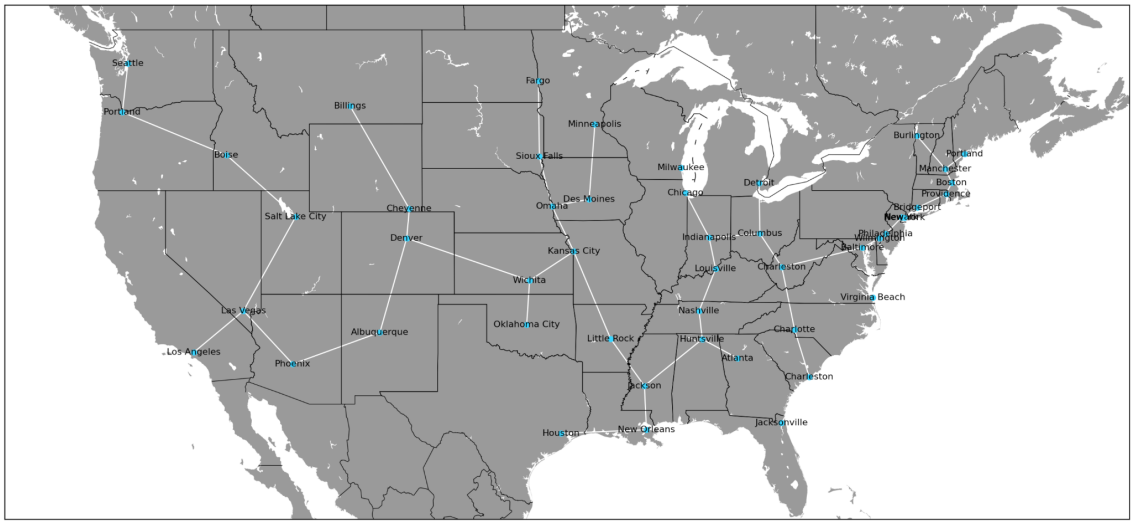
We began coding a file reader function in Python to iterate through the data of the 330 cities to extrapolate the information on cities, states, types, counties, population, latitudes and longitudes. To identify the most populous city, we sorted the data by population in decreasing order *(Module 0 - Sorting)*, starting with the most populated city, New York City. Then, we appended the data into a nested array initialized as “top48” that contains the sorted and most populous city in each state in the contiguous United States. We proceeded to use “top48” to create the graph in an adjacency matrix *(Module 5- Adjacency Matrices and List)* and output the results to a file for ease of use named, “top48.py.” Because of reading the file and creating the adjacency matrix, the time complexity of this program is $O(l + s^2)$ where l is the number of lines in the file and s is the number of states.

We began to map out a graph structure with vertices as the cities, the edges as the railway path and the edge weight as the distance between two cities. We started to build our minimum spanning tree as shown in our “mst_build.py” file. Our starting vertex was the most populous city, as identified before in our “top48” array, followed by using Prim’s algorithm *(Module 7 Greedy Algorithms 2)*, to calculate the lowest weighted edge at each step, which in this case was the distance. Because we used Prim’s algorithm, this took O((V+E)log(V)).

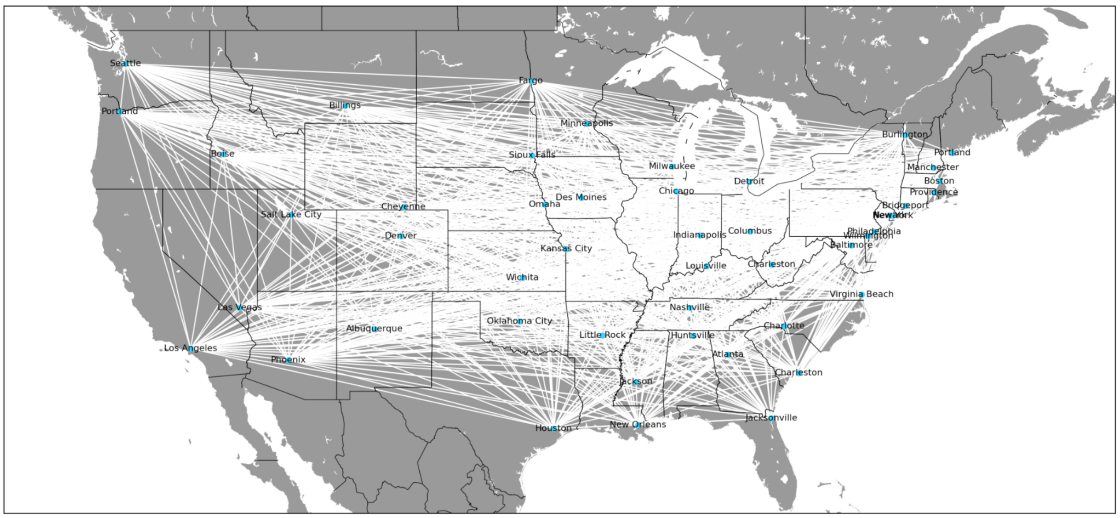
To visualize our graph, we utilized NetworkX and Matplotlib to graph our vertices and edges, as well as Basemap to realistically map out our cities on the map of the United States. The total rail track for the complete graph resulted in 1,445,157 miles.

Our railway after Prim's algorithm generated a minimum spanning tree with a total of 9,622 miles of rail track.

Visualization of the MST using Prim's algorithm:



Visualization of complete graph and MST:



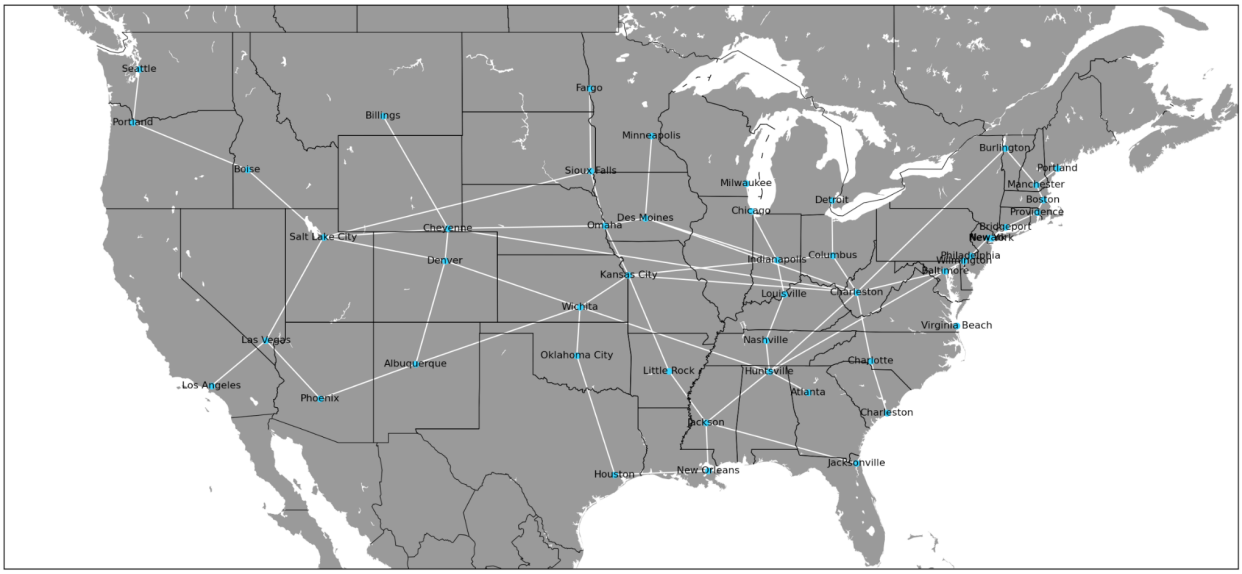
At first we thought to use Dijkstra's algorithm *(Module 6 Greedy Algorithms 1)* to find the shortest paths; however, we ended up using Floyd-Warshall *(Module 9: Dynamic Programming 2)*, since it is more efficient to find the shortest paths of all pairs of vertices (cities) for our use case. Our Python function takes in an adjacency matrix input parameter, calculates the distances between the vertices and returns a matrix of resulting distances. We iterate to find which additional edge reduces the sum of all

shortest paths. With the use of three for loops that iterate through n vertices each, the runtime complexity is $O(n^3)$. We also used a helper function to calculate the total track length that takes in the adjacency matrix input and returns the total length. The runtime complexity is $O(n^2)$.

Lastly, our final step was to take in the graph and complete graph as input parameters, then integrate all the functions altogether. We kept a travel_saved counter and chose to stop our algorithm once we laid down 20,000 miles of track to be comparable to the current Amtrak rail network of 21,400 miles *(Amtrak Corporate Profile)*.[5] The runtime complexity overall resulted in $O(n^5)$, largely due to calling Floyd-Warshall each time an edge is added; if we were to improve the complexity of our algorithm, it would be by implementing another method of recalculating shortest paths when only one edge is added to the graph. Our total miles of rail track with our high-speed rail network resulted in 21,106 miles.

CONCLUSION:

Answering our earlier question, the most efficient way to build a high-speed railway to connect the most populous cities in the contiguous United States is pictured below with cities in blue and the rail path in white.



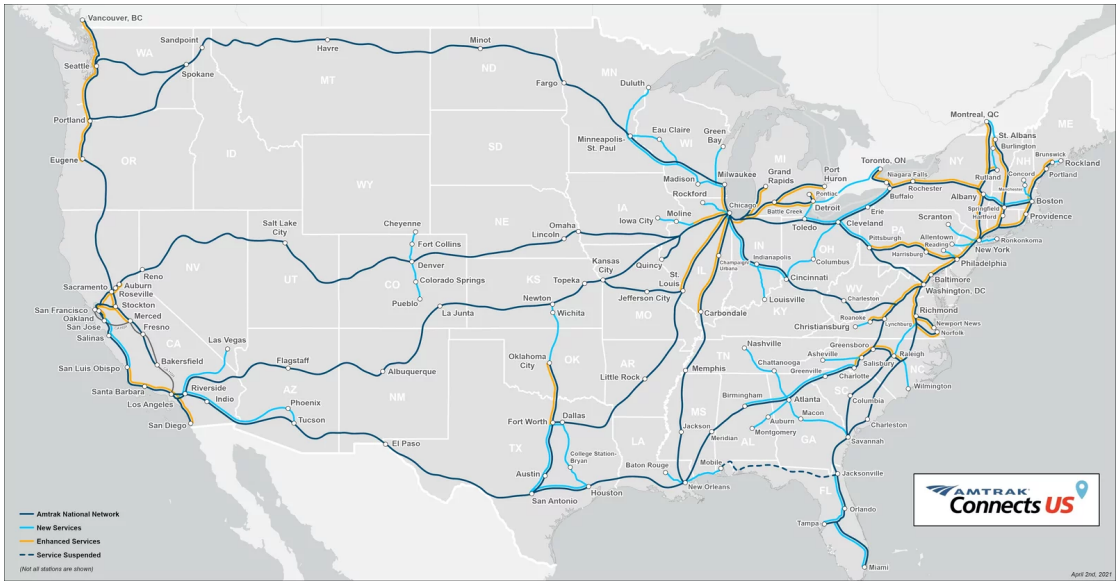
Laying track like this uses 21,106 total miles. This is a dramatic improvement from the 1,445,157 miles in the complete graph, and similar to the 21,400 miles in the

current Amtrak map[5]. We believe that our railway tracks are more efficiently laid out than the current Amtrak lines pictured below.

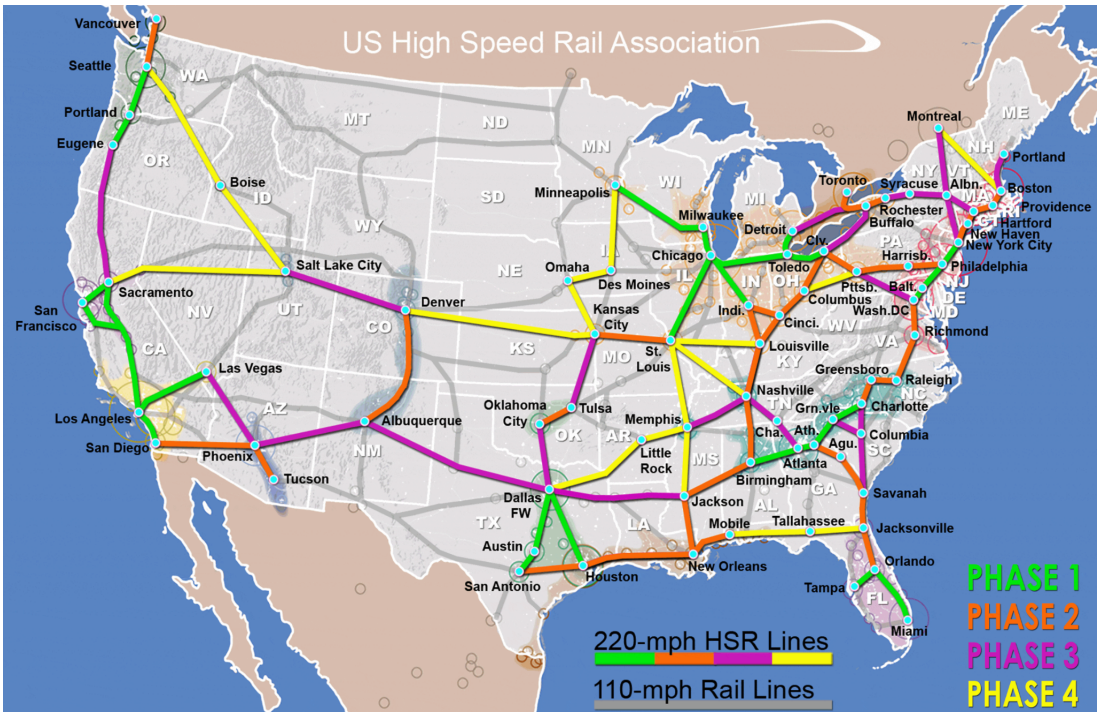
Current Amtrak railway paths[2]:

Our rail lines are similar to proposed high speed rail maps from Amtrak and the US High Speed Rail Association. We offer more connections between states in the Mountain West, whereas the existing proposals skew more toward connecting a few cities in the West with more populous cities like Seattle and Chicago. It would be interesting to know what led them to take those paths instead.

Amtrak Proposed High Speed Rail Map[3]:



US High Speed Rail Association Proposed Map[4]:

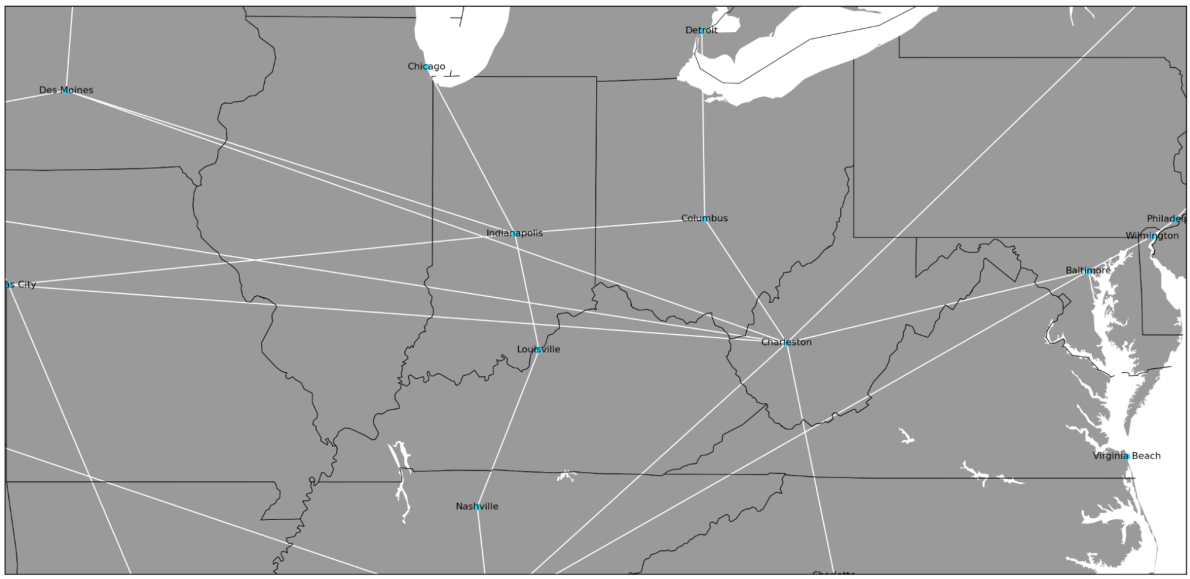


Limitations and Future Prospects:

We ran into a few limitations over the course of this project. The first is that Floyd Warshall takes a very long time to run, $O(n^3)$, leading to a total runtime of $O(n^5)$ for our algorithm. This takes about 10 minutes to run on our machines. Even though we do not use all the edges in the complete graph, we need to calculate them all in order to decide which ones are the best to add to the complete graph. In the future, we would consider other ways of determining which edges to add.

The final graph also has interesting busyness around the rust belt. There are several intersecting lines recommended. While these were the most efficient additional lines at the time they were added, they are not necessarily the most efficient edges for the final graph. This is because our algorithm is greedy, so there are some inefficiencies such as the edge from Charleston to Des Moines that bypasses Indianapolis, when a more efficient use of track would have been to connect through Indianapolis (graph below). Proposed maps from Amtrak and the US High Speed Rail Association (see above) do have several routes in this area, but not quite as many as ours.

Limitation with some inefficient paths:



We have also chosen to ignore certain factors in the interest of time that may end up being important later on. We chose not to consider terrain which may be a critical impediment to laying track in certain areas of the country like the Rocky Mountains. This consideration may be the reason our map looks so different from the formal proposals for that region. Another critical factor we have not considered is that there are more populous cities, especially in large states, that are not being served by our map which only considers the most populous city in each state. It may be more useful to create a line connecting Houston and Dallas or Los Angeles and San Jose than to add some of the lines we have in the Midwest. There may also be some practical advantages to laying track alongside existing railways that we have not taken into account. We would like to explore how costly it would be to also connect Alaska, to see if it may be worth pursuing. Other considerations include factoring in cost and defining the optimal speed in our “high-speed” rail. Currently, Amtrak allows trains to run as fast as 150 mph.[5] High speed bullet trains in Japan reach speeds of up to 200 mph. These are all aspects of future considerations that interest us to further explore and implement.[6]

In the end, the most efficient rail map is not just dependent on distance but requires an understanding of the ease of construction and ridership demands. It would be interesting if we could gather data on these factors so we could come up with our own tradeoff heuristic. We’d like to consider how to balance potential ridership

(connecting the most populous cities) with serving those in remote areas (connecting cities in each state). Notably, the map proposed by Amtrak connects every state but South Dakota, whereas on our map, Sioux Falls is a small hub that connects three other cities. We would like to understand how they decided to make their map and to see if we agree.

While our map may not be the final version that should be created, we believe it could help inform tradeoff decisions in the process of determining how to build a high speed rail system across the US. We hope that this project is under way, so we can all start taking one soon.

Omar Rashwan:

I really enjoyed working on this project as it allowed me to take the algorithms we have been learning about during the semester and apply them to a real-world scenario. Graph problems are especially appealing as there is visual feedback from the graph where you can observe how each algorithm transforms the edges and vertices from one iteration of the graph to another. Another aspect of this project that was enlightening to me was how it drove home the point that a greedy choice is not always the most optimal. While the algorithm we devised made optimal choices in each iteration, the final graph had some inefficiencies that probably could have been avoided if we had designed a more effective algorithm. Perhaps a dynamic programming solution could have led to a more optimal map overall, but since the runtime of our greedy solution was already $O(n^5)$, a dynamic programming algorithm may have had an even worse runtime complexity that would have made it infeasible to use on our hardware. Overall, I found this project valuable for learning how to recognize when to use certain algorithms to solve problems that are not as abstract as those we would find in LeetCode or a homework assignment.

Annie Pates:

This project really drove home for me how different intellectual exercises are from real life problems. It is so clear when we've done the homework that there's a right answer. Even the leetcode exercises that are based on real world applications don't require that much thought beyond problem-solving. This project showed how choosing the right approach and algorithm really affects the outcome. Choosing a greedy algorithm yielded some inefficiencies that I wouldn't have predicted before seeing the actual map. Even beyond choosing our approach, there are so many more

decisions to make around framing the problem to begin with. While we initially considered connecting the 50 most populous cities overall, we eventually chose to connect the most populous cities in each state. So no matter which algorithm we ran, the best way to create a cross country railroad was going to have a human element to it. The decisions get even more complex when we consider which extra lines to add. The tradeoffs between serving lots of people and a range of areas are complex enough without considering budgetary and political factors. It was really interesting to see an example of how the problem won't just be solved by choosing the right algorithm.

Josephine Yu:

It is one thing to learn algorithms in a theoretical setting such as in the modules, and another experience to apply these algorithms in a realistic setting. I enjoyed the challenge working through obstacles with the map of the United States and navigating them with tools we learnt throughout the class. Although the algorithms made sense conceptually, applying the greedy algorithm, Floyd-Warshall, with our track length calculation led us to a surprisingly long, ten minutes to output the final graph. I was not expecting the long running time to process all the edges, but it made sense after seeing the complete graph and all the possible rail paths. For the future, it would be interesting to explore how we could further improve the runtime efficiency. It was a comical process during the analysis portion when we outputted the first complete graph with 1.4 million miles of track. Afterwards, it was neat to note that the MST is only 9622 miles, however, it was not the most efficient way to travel realistically. I also enjoyed thinking of future considerations and other facets we can add onto this project later on. Lastly, I learned a lot regarding graph visualization and utilizing the Python library packages like NetworkX, Matplotlib and Basemap to actualize data into a realistic map of the US—definitely learnt a good amount of US geography.

References

1. Population dataset of top 330 cities: https://www.kaggle.com/datasets/axeltorbenson/us-cities-by-population-top-330

2. Existing cross US train map, Amtrak: https://www.amtrak.com/content/dam/projects/dotcom/english/public/documents/Maps/Amtrak-System-Map-1018.pdf

3. Proposed map based on Pres. Biden's infrastructure plan, NPR: https://www.npr.org/2021/04/06/984464351/as-biden-pushes-major-rail-investments-rail-amtraks-2035-map-has-people-talking

4. Proposed Future Rail Map, US High Speed Rail Association: http://www.ushsr.com/hsrmap/

5. Amtrak current rail track of 21,400 ("Amtrak Corporate Profile"): https://www.amtrak.com/content/dam/projects/dotcom/english/public/documents/corporate/nationalfactsheets/Amtrak-Corporate-Profile-FY2018-0319.pdf

6. Japanese bullet trains: https://www.japanstation.com/shinkansen-high-speed-train-network-in-japan/

‘

Appendix:

Github link with all the files: https://github.com/omrash/CS5800-Project

Appendix A: csv_reader.py

us2021census.csv file here:
https://github.com/omrash/CS5800-Project/blob/main/us2021census.csv

```
def dist(coords1, coords2):

    # one degree in coordinates is approx 69 mi
    MI = 69

    x1, y1 = coords1
    x2, y2 = coords2

    x = (x2 - x1) * MI
    y = (y2 - y1) * MI

    dist = (x ** 2 + y ** 2) ** 0.5

    return dist

def main():
    filename = "us2021census.csv"
    with open(filename, mode='r') as f:
        header = f.readline()
        data = f.readlines()

    states = {"AK", "HI", "DC"}
    top48 = []
    for i in range(len(data)):
        data[i] = data[i].strip('\n')
        data[i] = data[i].split(',')
        data[i][4] = int(data[i][4])
        data[i][5], data[i][6] = float(data[i][5]), float(data[i][6])
        lst = [data[i][0], data[i][5], data[i][6]]
        if data[i][1] not in states:
            states.add(data[i][1])
            top48.append(lst)

    '''
    'data' contains the whole dataset post-processing, 'top48' contains the
    most populous city in each state in CONUS. Currently using 'top48' to
    construct the graph.
    '''

    n = len(top48)
```

```
    adj_matrix = [[0 for x in range(n)] for y in range(n)]

    for i in range(n):
        for j in range(i, n):
            city1 = top48[i][1], top48[i][2]
            city2 = top48[j][1], top48[j][2]
            distance = dist(city1, city2)
            adj_matrix[i][j] = distance
            adj_matrix[j][i] = distance

    output = "top48.py"

    with open(output, mode='w') as o:
        o.write("top48 = " + str(top48) + '\n')
        o.write("g = " + str(adj_matrix))

if __name__ == "__main__":
    main()
```

Appendix B: mst_build.py

```
from top48 import *

from analysis_tools import *

from graph_visualizer import visualize

def prim(graph):
   n = len(graph)
   included = {0}
   mst = [[0 for x in range(n)] for y in range(n)]

   while len(included) < n:
       minimum = 999999999
       x, y = 0, 0
       for u in included:
           for v in range(n):
               if v not in included and graph[u][v] < minimum:
                   minimum = graph[u][v]
                   x, y = u, v
       included.add(y)
       mst[x][y] = minimum
       mst[y][x] = minimum

   return mst

def main():

   mst = prim(g)
```

```
    data = []

    for i in range(len(top48)):
        data.append((top48[i][0], top48[i][1], top48[i][2]))

    visualize(mst, data)

if __name__=="__main__":
    main()
```

Appendix C: graph_visualizer.py

```
'''
Status: WIP, non-functioning

This script uses the NetworkX library to visualize the input graph.
Just import the visualize() function from this file to use.

'graph' should be an adjacency matrix for all n vertices.
'array' should be an array for n vertices with
[n][0] = city_name, [n][1] = latitude, and [n][2] = longitude.
'''

import networkx as nx
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap as Basemap

def visualize(graph, array, x_offset=0, y_offset=0, scale=1):
    '''
    x_offset: shifts all points along the x axis
    y_offset: shifts all points along the y axis
    scale: scales the distance from the origin (0,0) by input
    '''
    n = len(graph)
    G = nx.Graph()
    pos = {}
    name = {}

    # Set up basemap
    m = Basemap(
        projection='merc',
        ellps = 'WGS84',
        llcrnrlon=-130,
        llcrnrlat=25,
        urcrnrlon=-60,
        urcrnrlat=50,
        lat_ts=0,
        resolution='i',
        suppress_ticks=True)

    for u in range(n):
```

```
        # calculate position
        x = (array[u][2] * scale) + x_offset
        y = (array[u][1] * scale) + y_offset

        # Basemap converts lat and long to map coordinates
        x,y = m(x, y)

        # populate dicts
        pos[u] = x,y
        name[u] = array[u][0]

        # adds nodes with array data as attributes
        G.add_node(u, pos=(x,y), name=graph[u][0], lat=graph[u][1], long=graph[u][2])


    for u in range(n):
        for v in range(u+1, n):
            # add edges from matrix if > 0
            if graph[u][v]:
                G.add_edge(u, v, weight=graph[u][v])

    m.drawcountries()
    m.drawstates()
    m.fillcontinents(color='#888888')


    nx.draw_networkx(G,
                     pos=pos,
                     labels=name,
                     font_size=8,
                     node_size=20,
                     node_color='#33ccff',
                     edge_color='white')
    plt.show()
```

Appendix D: analysis_tools.py

```
INF = 9999999

def track_length(adj_matrix):
    '''
    Returns the total length of track from an input adjacency matrix
    '''
    n = len(adj_matrix)
    track = 0

    for u in range(n):
        for v in range(u + 1, n):
            track += adj_matrix[u][v]

    return track
```

```
def floyd_warshall(adj_matrix):
    '''
    Returns a matrix of distances between vertices based on the input adjacency
    matrix. Runs in O(n^3) time.
    '''
    n = len(adj_matrix)
    dist = [[INF for x in range(n)] for y in range(n)]

    for u in range(n):
        dist[u][u] = 0
        for v in range(n):
            if adj_matrix[u][v]:
                dist[u][v] = adj_matrix[u][v]

    for k in range(n):
        for i in range(n):
            for j in range(n):
                dist[i][j] = min(dist[i][j], (dist[i][k] + dist[k][j]))

    return dist
```

Appendix E: railgorithm.py

```
from mst_build import prim
from analysis_tools import *
from graph_visualizer import visualize
from top48 import *

def railgorithm(graph, complete_graph):
    '''
    O(n^5)
    '''
    if len(graph) != len(complete_graph):
        raise ValueError("Graph lengths must be equal")
    n = len(graph)
    curr_sum = track_length(floyd_warshall(graph))

    travel_saved = 0
    x = 0
    y = 0
    for u in range(n):
        for v in range(u + 1, n):
            if graph[u][v] == 0:
                graph[u][v] = complete_graph[u][v]
                delta = curr_sum - track_length(floyd_warshall(graph))
                graph[u][v] = 0
                if delta > travel_saved:
                    travel_saved = delta
                    x,y = u,v
```

```
    graph[x][y] = complete_graph[x][y]

    return graph

def main():
    '''
    This took 10 minutes to produce an output with the current parameters
    '''

    mst = prim(g)

    new = mst

    while(track_length(new) < 20000):
        new = railgorithm(mst, g)

    visualize(new, top48)

if __name__=="__main__":
    main()
```