Northeastern University Khoury College of Computer Sciences
CS 5008 Algorithms Final Project - Summer 2022
YouTube Link:

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

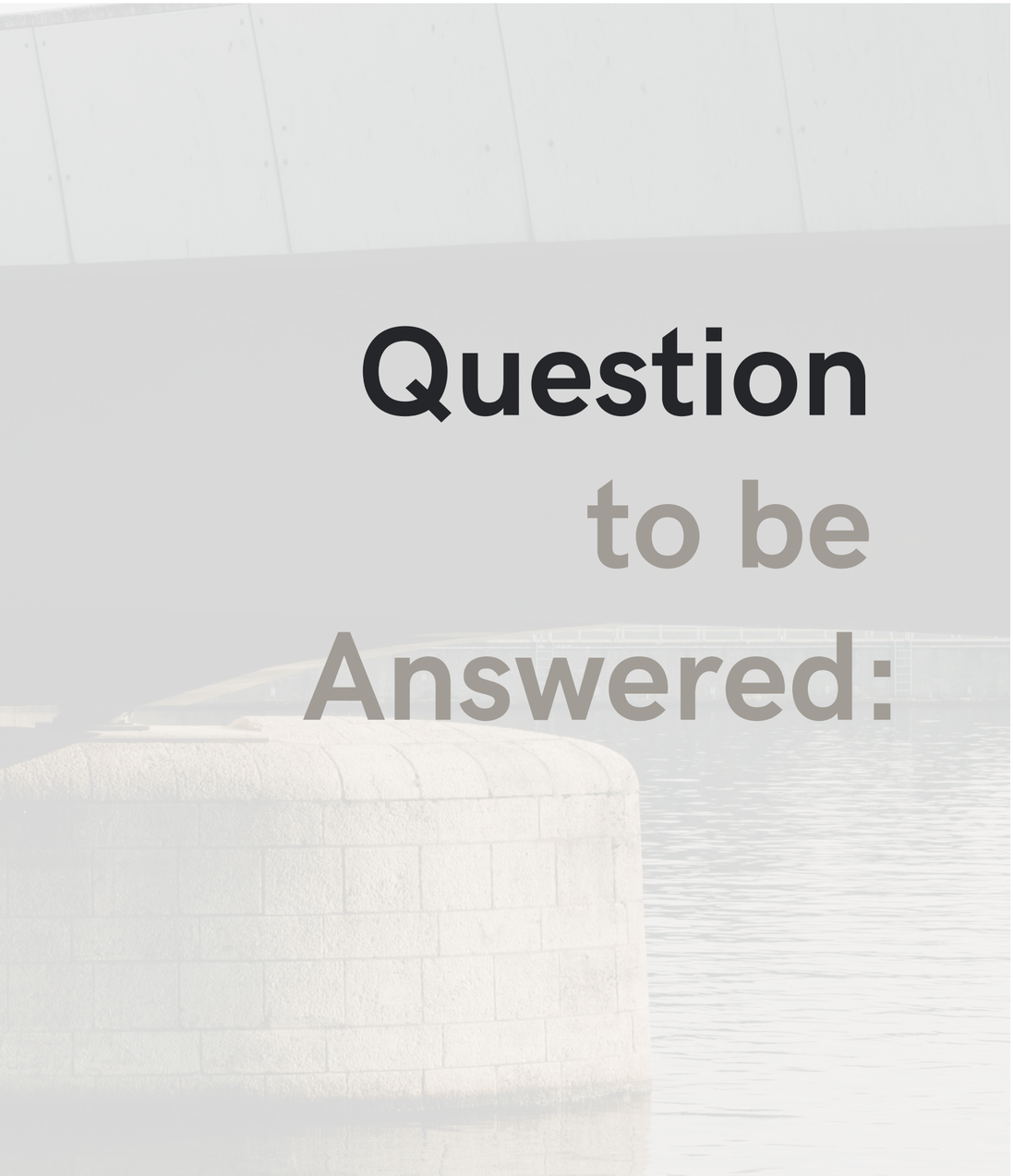
Initial Inspiration

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 discusssion 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 continental 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

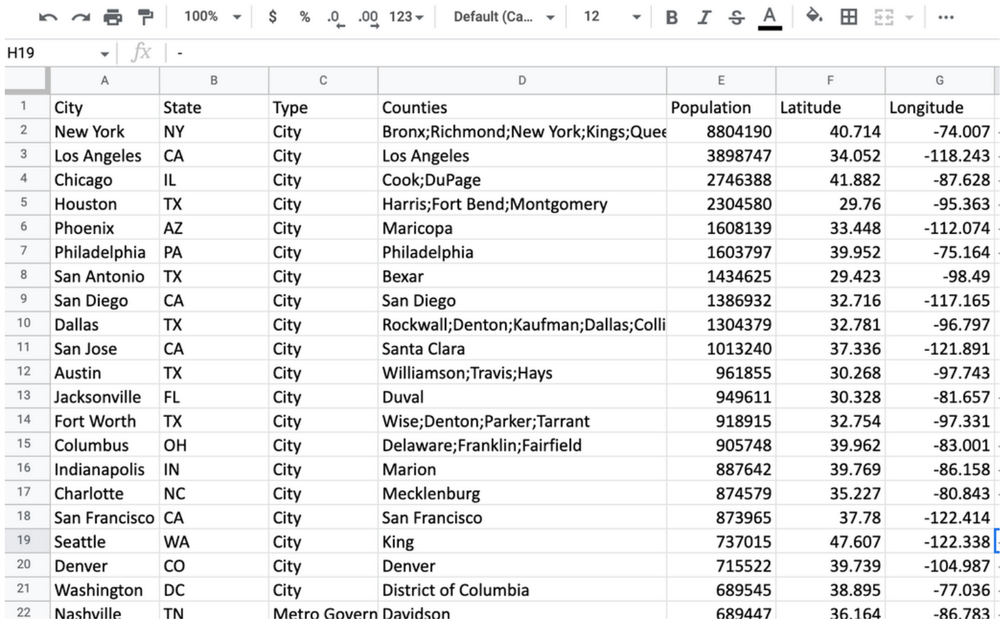
Methods Used to Gather data

- Wikipedia dataset on the top 330 most populous cities in the United States including:
  - 2020 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.

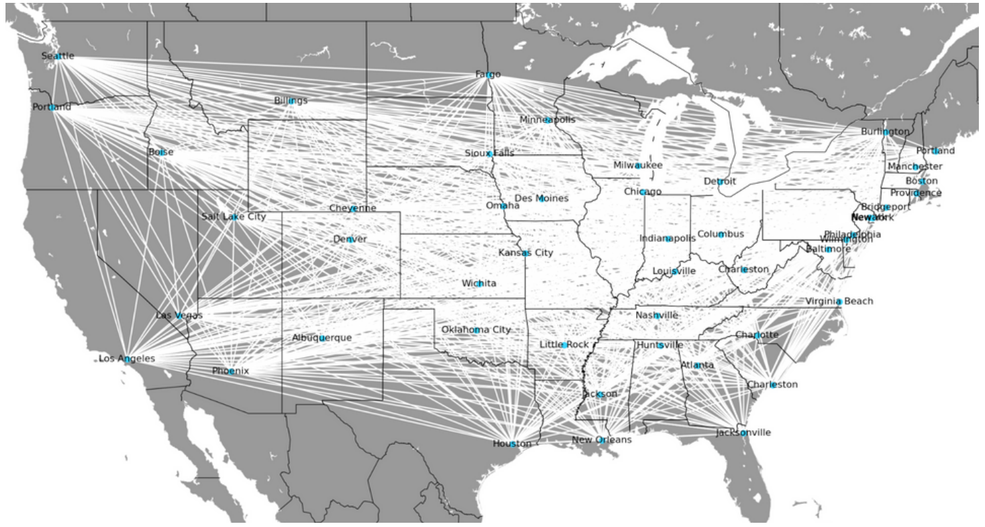
```python
import networkx as nx
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap as Basemap

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
    '''
```

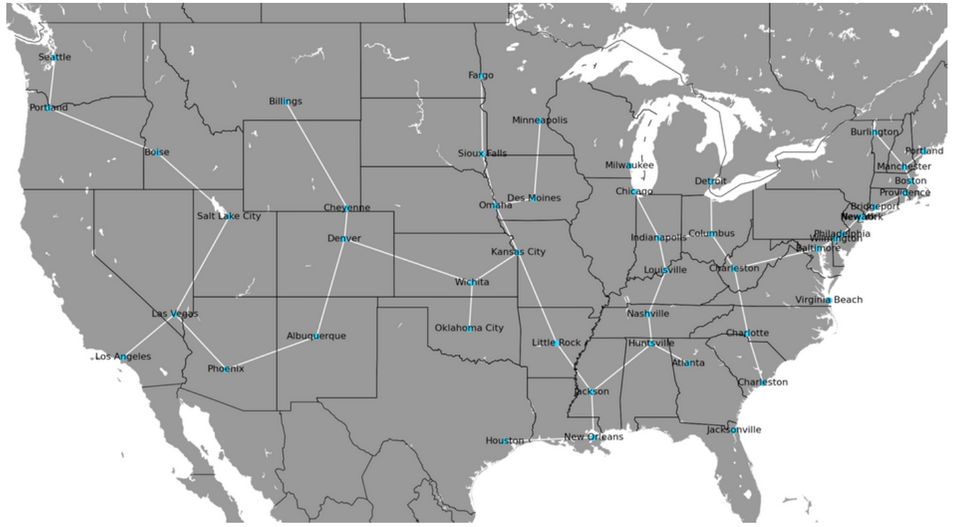
graph_visualizer.py

Total rail track:

Complete graph = 1,445,157 miles



Our railway MST using Prim's algorithm = 9622 miles



Using Floyd-Warshall

From Module 9: Dynamic Programming 2

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

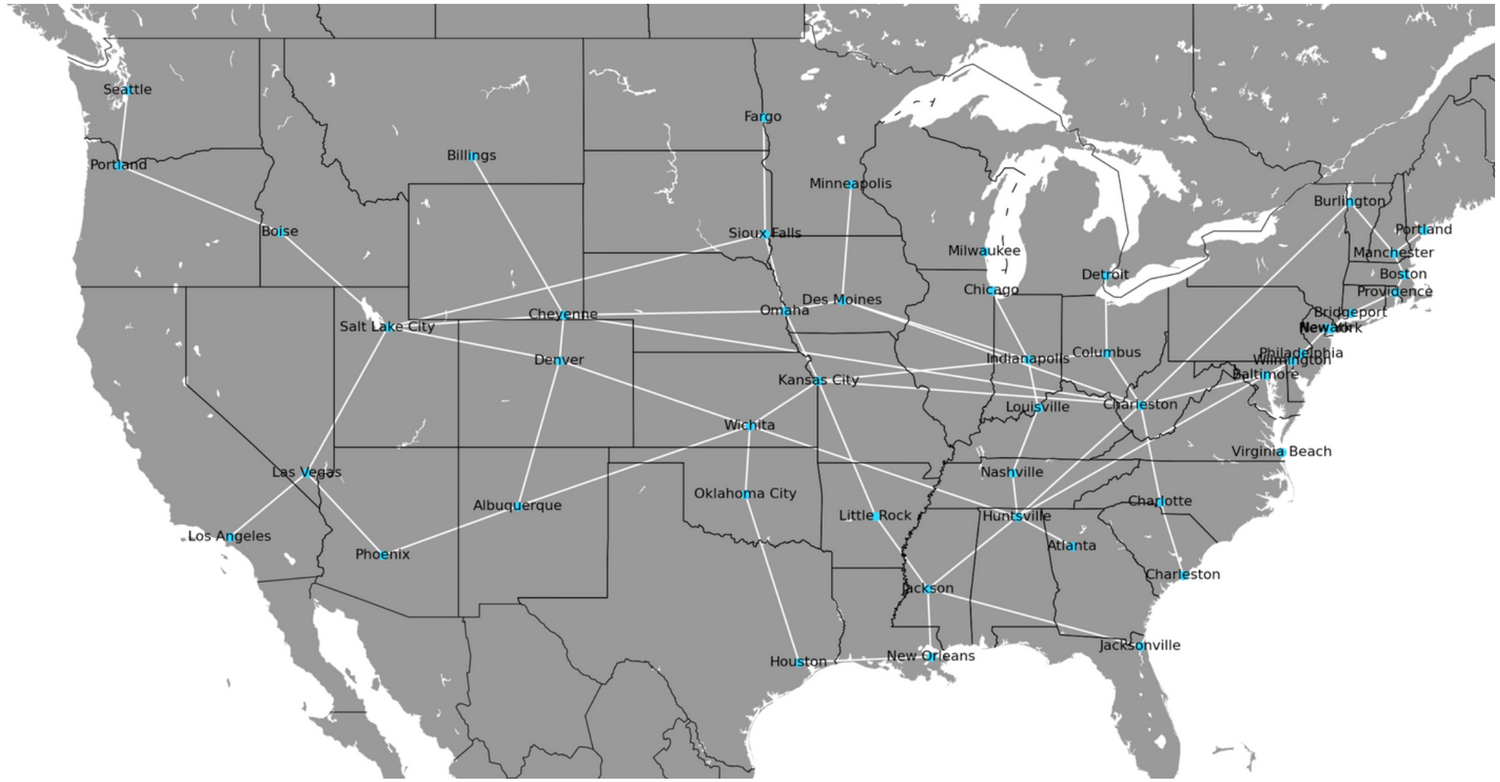
Result & Conclusion

Summary of findings, future considerations, conclusion

Question Being Answered:

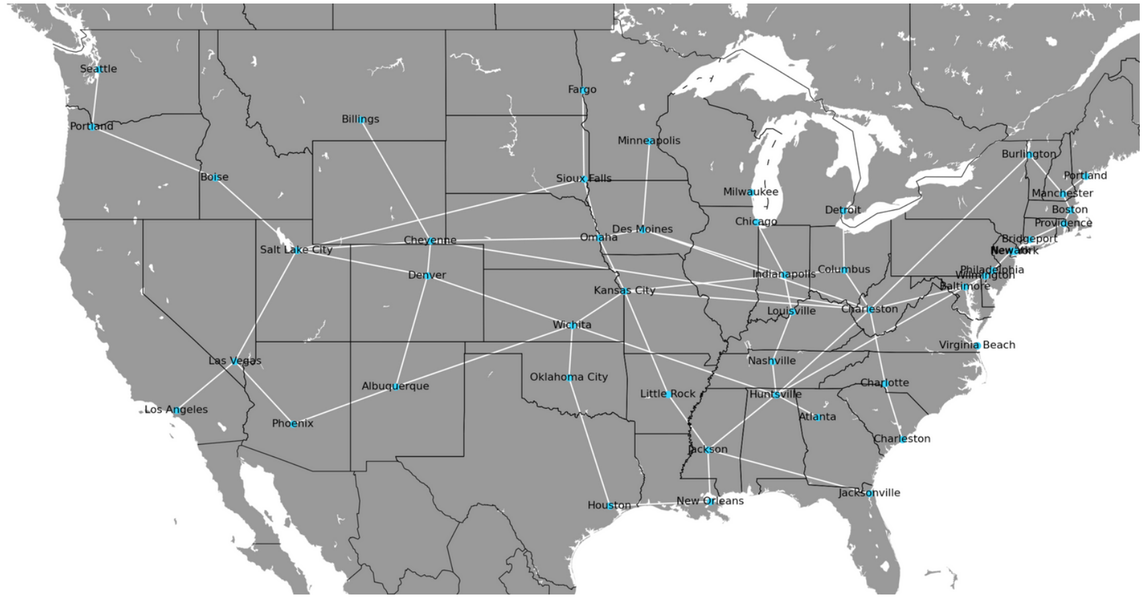
What is the most efficient way to build a high-speed railway to connect the most populous cities in the continental United States?

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

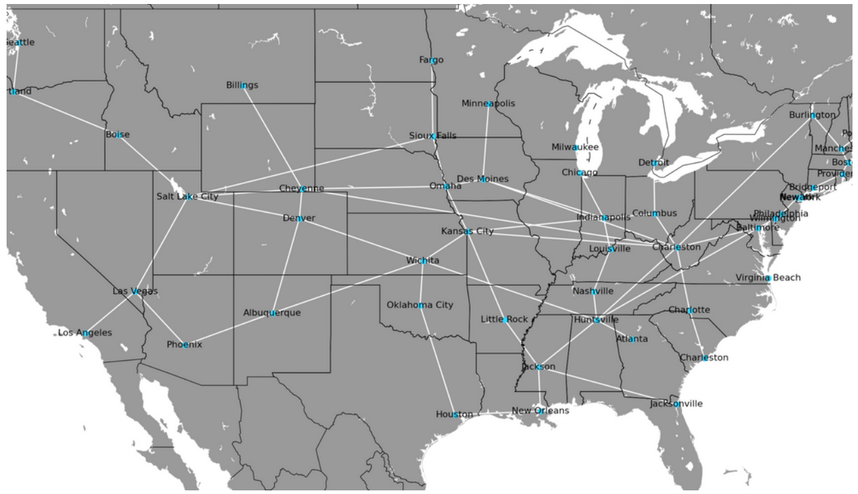
- Laying track like this totaled 21,106 total miles
- This is a dramatic improvement from the 1,445,157 miles in the complete graph, and similar to the 21,400 miles in the current Amtrak map
- We believe that our railway tracks are more efficiently laid out than the current Amtrak



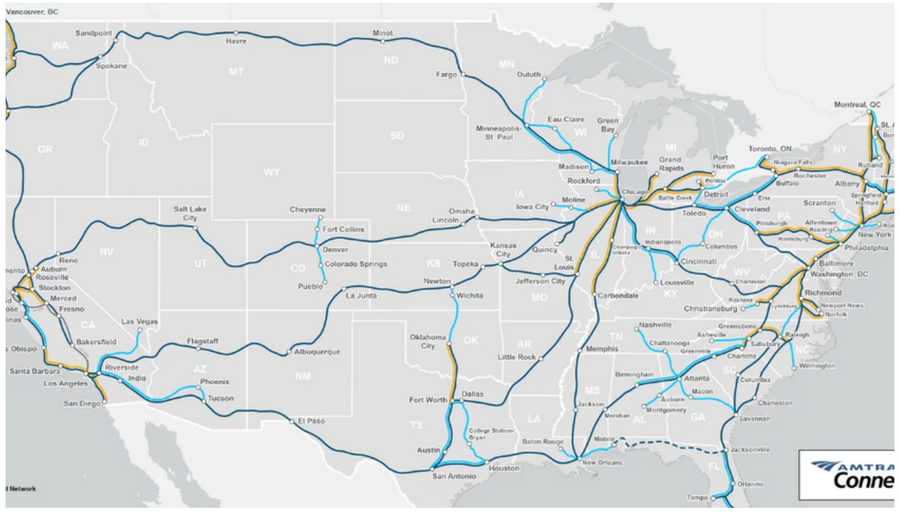
Our High-Speed railgorithm Map

Amtrak Network Railway Map

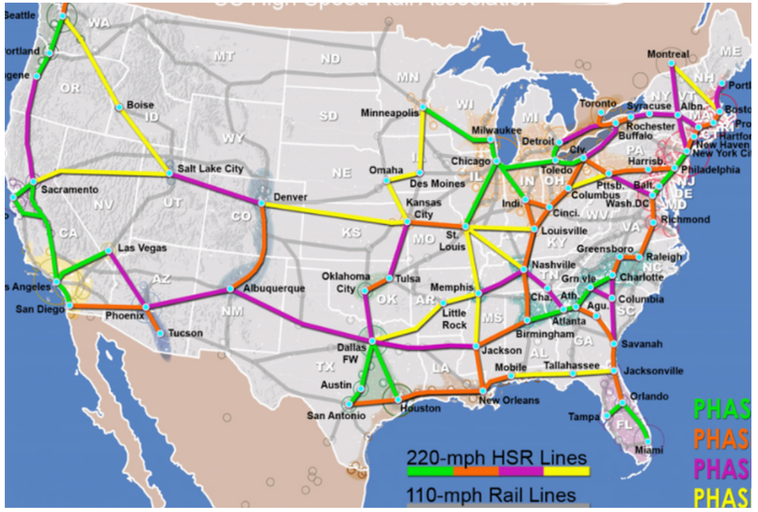
Comparison to Amtrak Proposal



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.

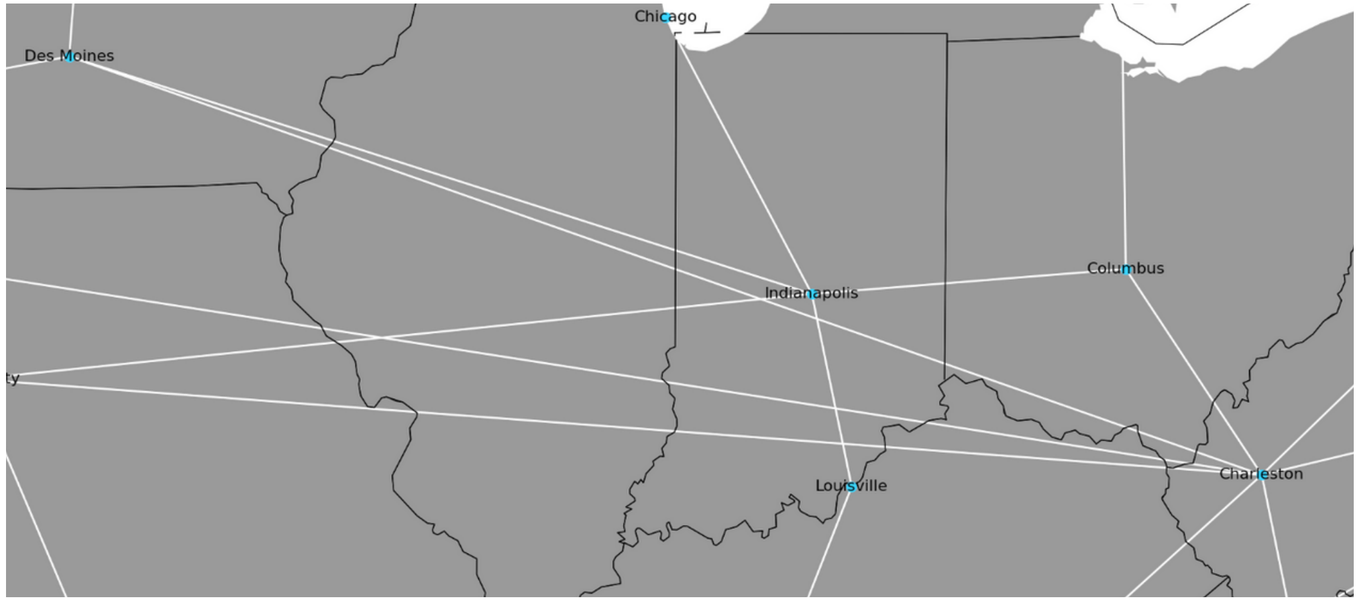
Limitations & Improvements

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

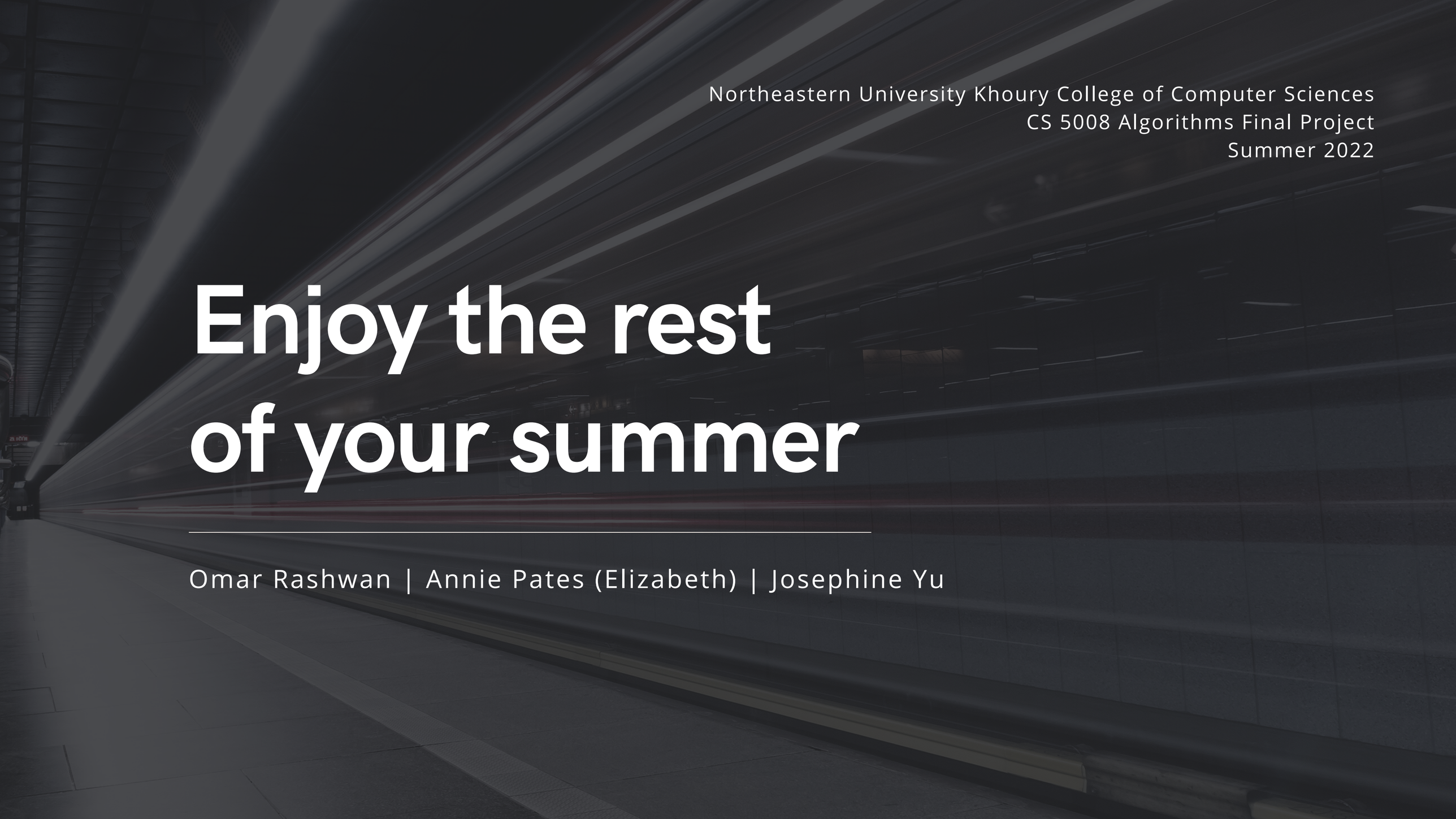
- Consider budget parameters
- Explore cost to connect to Alaska

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