Summer Research 2024

filter_data_subway.py & filter_data_bus.py

Main goal: filter out vehicle trips between 0 am and 5 am

- Merge trips.txt with stop_times.txt
- Convert arrival_time and departure_time to 24-hour time scale.
- Filter out trips that have a start time after 0 am and end before 5 am
- Store the processed data in a new text file called filtered_trips.txt

final_subway_network.ipynb & final_bus_network.ipynb

Main goal: construct the one layer transport network for subway and bus

Main methodology:

- **merge_close_nodes(G, threshold)** → Merge stops that are within a certain threshold (200m)
    - clusters → a list containing all individual cluster
    - visited → a set to record nodes that have been added to existing clusters
    - For each node/stop:
        - if it's not been visited:
            - create a new cluster containing it
            - add it to visited
            - instantiate a queue containing the node
            - While the queue is not empty (basically, we want to iteratively find all nodes that are within 200m of one of the nodes in the current cluster) :
                - Pop the first element in the queue
                - Get its position
                - For all other nodes:
                    - If the node has not been visited, Calculate the distance with the popped node
                    - If the distance is within the threshold, add the node to the queue, to visited and to the cluster
        - Add this cluster to clusters
- **build_matrix(stop_details): →** build two adjacency matrix trip_count and travel_time

- Use stop_sequence as an indicator of the next stop. If the current stop_sequence is incremented by 1, we know immediately that the current stop is the next stop of the previous stop.

- **build_network(clusters, trip_count, travel_time, title):** → use networkx to build and visualize the transport network
  - calculate the centers of all clusters and add them to the networkx graph as new nodes.
  - add edges/links between two nodes from the two adjacency matrix built above and store the travel_time and trip_count in each edge.

merge_subway_bus.py

Main goal: construct a two layer transport network by combining subway and bus networks

Main methodology:

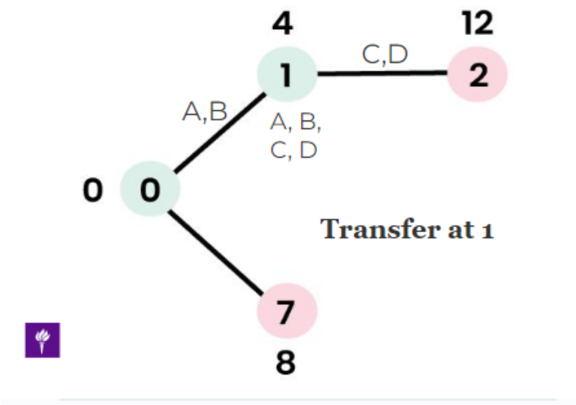
- **merge_subway_bus(subway_network, bus_network, walking_speed=1.47):** → add all subway and bus stations to a new network while keeping their original edges and adding transfer edges between subway and bus stations if appropriate.
  - for every two bus and subway stations, if their distance is walkable(500m), add a transfer edge between them to indicate possible transfers.

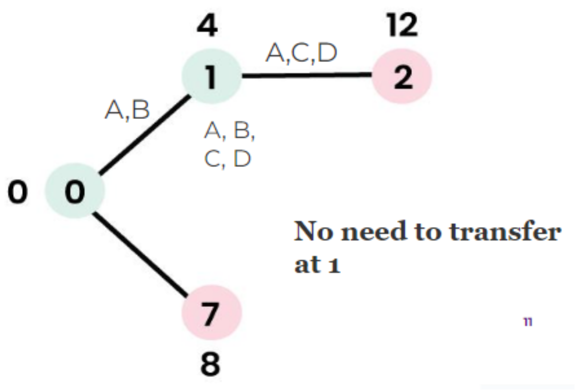
compute_shortest_path.py

Main goal: compute the path with shortest traveling time for two nodes.

Main methodology: standard Dijkstra's Algorithm plus evaluating transfer time when switching routes and stop time at each station.

- Initialize the distance between every node to source node to infinity
- Set the distance to source code to 0
- Initialize a priority queue(min heap) to extract the minimum distance node more efficiently
- Detect transfer:





    - Once two adjacent edges don’t have a common route, there must be a transfer made at the stop.
    - Add 6 mins to the path if a transfer exists.

doctor_network_two_layer.ipynb