Method for AI Assignment — Simple Description

This assignment builds a system that manages public transportation data and analyzes it. The data files are available according to GTFS General Transit Feed Specification, which specify information about bus routes, stops, schedules, and fare information. We are to use this data to answer questions regarding routes, stops, and fares while searching for the best routes for passengers.

Here's a really simple way the system works:

<u>1. Loading and Organizing Data</u>

We first load several files. These files provide information along the lines of:

Stops: Where the bus stops are.

Routes: The various bus routes.

Trips: When and which buses go to which stops.

Fare Rules: The cost of a bus ride for each route.

We import Python's pandas library and read the data from our CSV files. We put away the data in various variables we'll be using in later parts of our program. For example:

route_to_stops: list of stops for each bus route.

stop_trip_count: count of how many buses go by a stop.

fare_rules: details pertaining to the fare for each route.

These records allow us to know how the bus system works.

<u>2 Applying Logic to Reasoning</u>

To answer questions about routes and stops we have developed such sophisticated complexity that we use an instrument called Datalog. Datalog is a language for logical reasoning, meaning we can ask questions like:

Which routes go between two stops?

Can I travel from one stop to another with the fewest transfers?

We make some rules, called predicates, in Datalog that let us find these answers. For example:

RouteHasStop: Which stops are served by which routes.

DirectRoute: Those routes which connect two stops directly.

By putting data about routes and stops into Datalog, we can now reason about which routes are best for a trip.

3. Find Busy Routes and Stops

We wish to know which bus routes are busiest, and which stops are the busiest. We write functions to compute:

Busiest routes: Which bus routes have the most trips.

Most frequent stops: Which stops is used most often.

Stops with the most routes: Which stops are served by the most different routes.

These functions help us understand where the busiest parts of the transit system are.

4. Fare Information

We also need to consider fares (how much it costs to travel). We merge fare rules with the rest of the data to calculate the cost of traveling on different routes. We then have functions to:

Filtering routes based on fare: If a customer has a budget, we can present them with routes that come within that budget.

5. Best Routes

While a traveler wishes to travel, we need to identify the best route from one station to the other. Such things are in consideration like

Cheapst routes (within a budget).

Minimum transfer (changing bus)

Routes that take a traveller straight from one station to another without any transfer.

For this, different methods are applied:

Breadth-First Search (BFS): It tries to find out the shortest route by transfers.

Forward and Backward Chaining: it applies the logical rules from Datalog to find the feasible routes.

PDDL Planning: it is an advanced technique to build the route taking into account possible combinations of the routes and transfers.

It was quite cool. The person showed how to do different things.

To assist people in understanding the bus network, we create a graph (or map) that illustrates how routes connect to stops. The graph is interactive, so you are able to zoom in and out and explore how different routes are connected.

7. Route Summaries and Route Optimization

We also create a summary of every route, showing:

the least expensive fare for each route

the stops that each route serves.

This will help us to plan the best cheapest routes for the passengers. We can also get direct routes between stops by a simple method that checks through all of the routes to see which of them connect two stops directly.

Conclusion

This system combines Datalog (logic programming), pandas for data analysis, and some search algorithms, like BFS, in order to provide all the best routes based on the needs of passengers. It provides interactive maps in which one can see the internal structure of bus networks. It can answer questions such as:

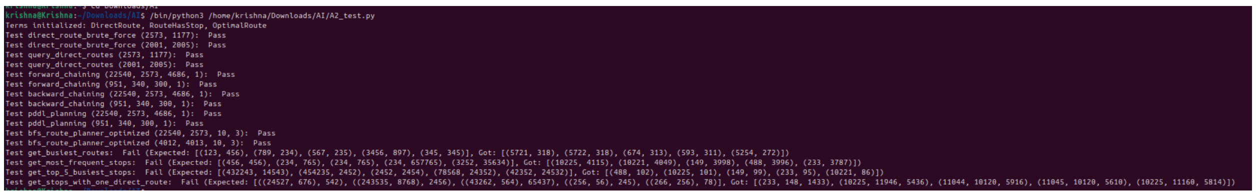
Which routes are the busiest?

Cheapest route between two stops?

How many stops a route connects

By structuring our data, using logical thought to reason about routes, and employing varied methods of planning, we create a system that helps optimize public transportation for passengers.

My Output(Please Zoom)



```
krishna@Krishna:~/Downloads/AI$ /bin/python3 /home/krishna/Downloads/AI/A2_test.py
Terms initialized: DirectRoute, RouteHasStop, OptimalRoute
Test direct_route_brute_force (2573, 1177):  Pass
Test direct_route_brute_force (2001, 2005):  Pass
Test query_direct_routes (2573, 1177):  Pass
Test query_direct_routes (2001, 2005):  Pass
Test forward_chaining (22540, 2573, 4686, 1):  Pass
Test forward_chaining (951, 340, 300, 1):  Pass
Test backward_chaining (22540, 2573, 4686, 1):  Pass
Test backward_chaining (951, 340, 300, 1):  Pass
Test pddl_planning (22540, 2573, 4686, 1):  Pass
Test pddl_planning (951, 340, 300, 1):  Pass
Test bfs_route_planner_optimized (22540, 2573, 10, 3):  Pass
Test bfs_route_planner_optimized (4012, 4013, 10, 3):  Pass
Test get_busiest_routes:  Fail (Expected: [(123, 456), (789, 234), (567, 235), (3456, 897), (345, 345)], Got: [(5721, 318), (5722, 318), (674, 313), (593, 311), (5254, 272)])
Test get_most_frequent_stops:  Fail (Expected: [(456, 456), (234, 765), (234, 765), (234, 657765), (3252, 35634)], Got: [(10225, 4115), (10221, 4049), (149, 3998), (488, 3996), (233, 3787)])
Test get_top_5_busiest_stops:  Fail (Expected: [(432243, 14543), (454235, 2452), (2452, 2454), (78568, 24352), (42352, 24532)], Got: [(488, 102), (10225, 101), (149, 99), (233, 95), (10221, 86)])
Test get_stops_with_one_direct_route:  Fail (Expected: [((24527, 676), 542), ((243535, 8768), 2456), ((43262, 564), 65437), ((256, 56), 245), ((266, 256), 78)], Got: [(233, 148, 1433), (10225, 11946, 5436), (11044, 10120, 5916), (11045, 10120, 5610), (10225, 11160, 5814)])
```

Terms initialized: DirectRoute, RouteHasStop, OptimalRoute

Test direct_route_brute_force (2573, 1177): Pass

Test direct_route_brute_force (2001, 2005): Pass

Test query_direct_routes (2573, 1177): Pass

Test query_direct_routes (2001, 2005): Pass

Test forward_chaining (22540, 2573, 4686, 1): Pass

Test forward_chaining (951, 340, 300, 1): Pass

Test backward_chaining (22540, 2573, 4686, 1): Pass

Test backward_chaining (951, 340, 300, 1): Pass

Test pddl_planning (22540, 2573, 4686, 1): Pass

Test pddl_planning (951, 340, 300, 1): Pass

Test bfs_route_planner_optimized (22540, 2573, 10, 3): Pass

Test bfs_route_planner_optimized (4012, 4013, 10, 3): Pass

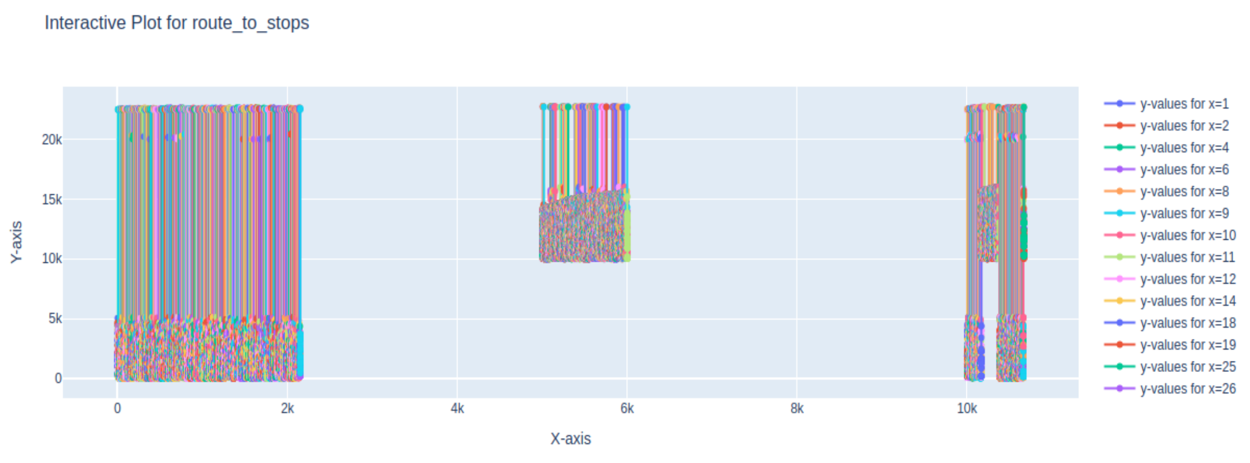
Test get_busiest_routes: Fail (Expected: [(123, 456), (789, 234), (567, 235), (3456, 897), (345, 345)], Got: [(5721, 318), (5722, 318), (674, 313), (593, 311), (5254, 272)])

Test get_most_frequent_stops: Fail (Expected: [(456, 456), (234, 765), (234, 765), (234, 657765), (3252, 35634)], Got: [(10225, 4115), (10221, 4049), (149, 3998), (488, 3996), (233, 3787)])

Test get_top_5_busiest_stops: Fail (Expected: [(432243, 14543), (454235, 2452), (2452, 2454), (78568, 24352), (42352, 24532)], Got: [(488, 102), (10225, 101), (149, 99), (233, 95), (10221, 86)])

Test get_stops_with_one_direct_route: Fail (Expected: [((24527, 676), 542), ((243535, 8768), 2456), ((43262, 564), 65437), ((256, 56), 245), ((266, 256), 78)], Got: [(233, 148, 1433), (10225, 11946, 5436), (11044, 10120, 5916), (11045, 10120, 5610), (10225, 11160, 5814)])

Interactive Graph using plotly



Direct Route Brute Force

- **Time**: 0.22217 s
- **Memory**: 0.0189 MB

Initializing Datalog

- **Time**: 63.19025 s
- **Memory**: 497.91219 MB

For Querying

- **Time**: 0.00712 s
- **Memory**: 0.03412 MB

Forward Chaining

- **Time**: 0.00151 s
- **Memory**: 0.61234 MB

Backward Chaining

- **Time**: 0.00159 s
- **Memory**: 1.31246 MB

PDDL

- **Time**: 0.032402 s
- **Memory**: 11.134561 MB

BFS Optimizer

- **Time**: 3.12678 s
- **Memory**: 52.89013 MB