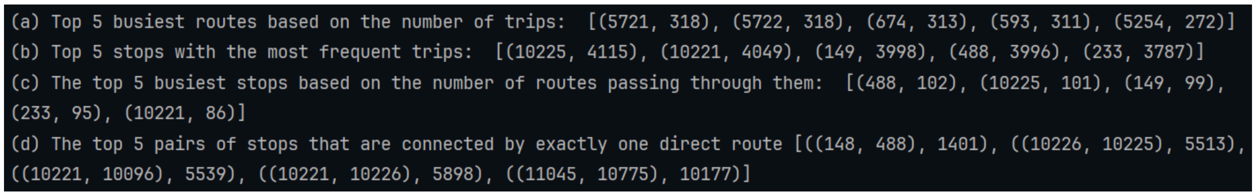
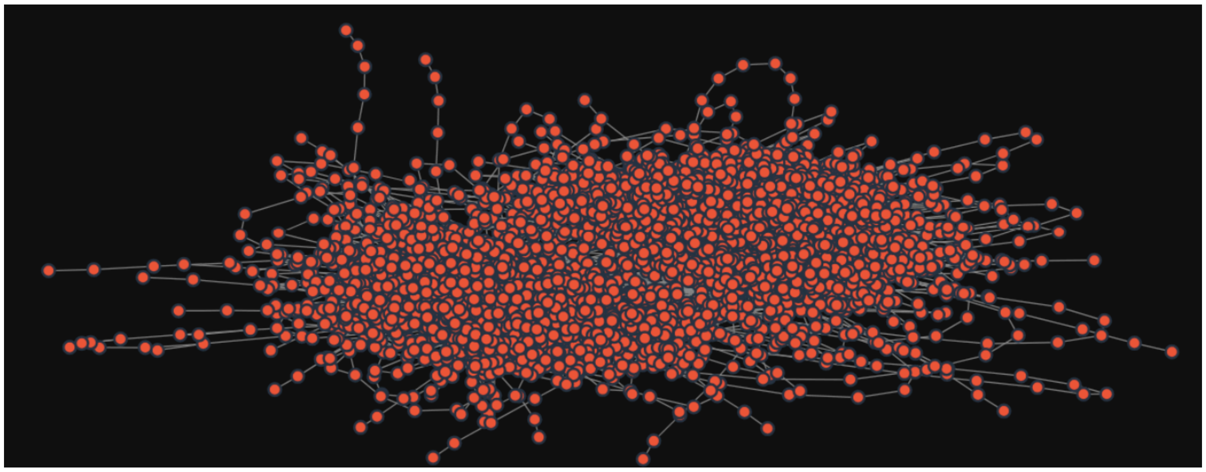
AI - ASSIGNMENT REPORT [COMPUTATIONAL]

Q1 Creating Knowledge Base
Output of the (a), (b), (c), and (d)



```
(a) Top 5 busiest routes based on the number of trips:  [(5721, 318), (5722, 318), (674, 313), (593, 311), (5254, 272)]
(b) Top 5 stops with the most frequent trips:  [(10225, 4115), (10221, 4049), (149, 3998), (488, 3996), (233, 3787)]
(c) The top 5 busiest stops based on the number of routes passing through them:  [(488, 102), (10225, 101), (149, 99),
(233, 95), (10221, 86)]
(d) The top 5 pairs of stops that are connected by exactly one direct route [((148, 488), 1401), ((10226, 10225), 5513),
((10221, 10096), 5539), ((10221, 10226), 5898), ((11045, 10775), 10177)]
```

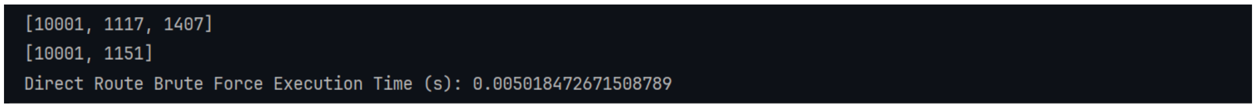
The plot for the knowledge base was created using the route to stops:



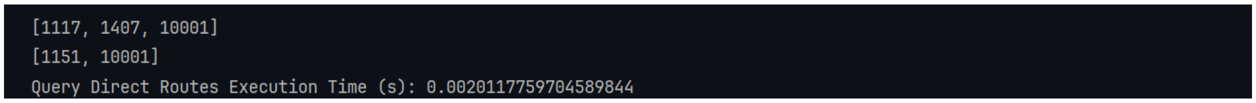
Q2 Reasoning [(i): Direct Route Brute Force, (ii): Query Direct Routes]

Used Time Libaray available in python for determining the Time of Execution.
Used TraceMalloc library available in python for determining the Memory Usage.

(a): The Execution time of (i) Direst Route Brute Force was 0.005s approx. for the given two public test cases and (ii) Query Direct Routes was taking ½ Execution time taken by (i) that comes out to be 0.002s. This proves that that the (ii) was comparatively faster the (i), (ii) will be preferred for complex queries to be answered. As size of input data increases we must tend toward the use of (ii)more and more as pyDatalog helps in reducing the redundant checks and has better time complexity then (i). (ii) is twice as fast as (i) when comparing the results we got.

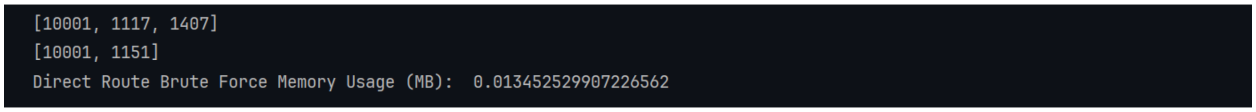


```
[10001, 1117, 1407]
[10001, 1151]
Direct Route Brute Force Execution Time (s): 0.005018472671508789
```

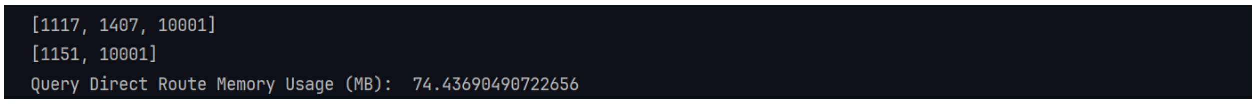


```
[1117, 1407, 10001]
[1151, 10001]
Query Direct Routes Execution Time (s): 0.0020117759704589844
```

To Compare memory usage by both the algorithms, the results I got show that (i) takes a lot more memory then (ii) because pyDatalog requires additional memory to store rules, facts and load data. While it takes a lot of memory results in faster queries for complex queries and large input sizes. Its pyDatalog's framework takes a significant amount of memory which involves in storing terms and caching results on the other hand (i) did not use any extensive framework like this it straightforward brute force algorithm therefore (ii) uses more amount of memory then (i)



```
[10001, 1117, 1407]
[10001, 1151]
Direct Route Brute Force Memory Usage (MB):  0.013452529907226562
```



```
[1117, 1407, 10001]
[1151, 10001]
Query Direct Route Memory Usage (MB):  74.43690490722656
```

(b): Intermediate Steps involved in Direct Routes Brute force were:

- Iterate over all routes and stops.
- Check for conditions involved like check for both stops present in an route and second stop comes after first.
- Store and return the result
- Its a simple, straightforward brute force algorithm, with minimal memory usage and slower query results.

Similarly the intermediate steps involved in Query Direct Routes were:

- Defining of logical rules and terms with respect to routes data.
- Involves indexing and caching the results which uses significant amount of memory as compared to
- Retrieve results from inferred solution. The memory usage was high but faster results for large input data size as compared to brute force approach.

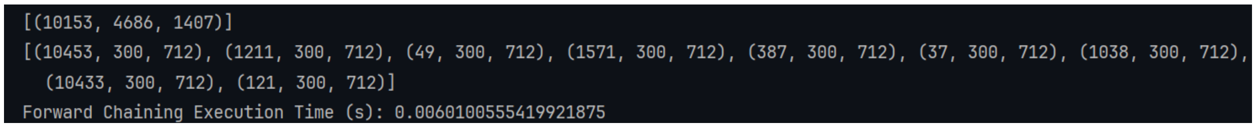
These were intermediate steps involved in both the algorithms.

(c): For Direct Route brute force the steps involves are (1) iterate over all routes and in all routes iterate over all stops included in it. Which comes out to be roughly around O(Routes X Stops) complexty. These many steps are involved in Direct routes brute force.

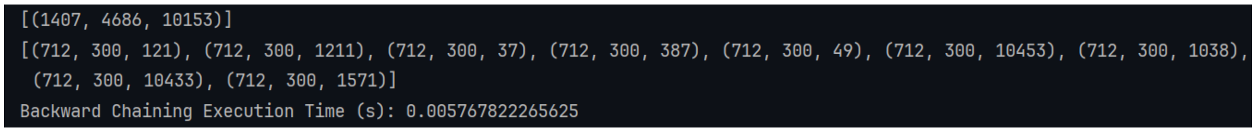
For Query Direct Routes the main number of steps were included in intializing datalog and in indexing and caching of routes and stops in rules and relations invloved in datalog framework which will be around O(RxS) and for each query result it may take only almost constant time to compute results for that which reduces steps if inputs data is huge and we need to query out many queries then it wil be significantly faster and using less number of steps then Direct Route Brute Force.

Q3: Planning [(i): Forward Chaining, (ii) Backward Chaining]

(a): The Execution time of (i) Forward Chaining was 0.006s approx. for the given two public test cases and (ii) Backward Chaining was taking slightly less Execution time taken by (i) comes out to be 0.005s. This proves that that the (ii) was faster then (i), (ii) will be preferred for complex queries to be answered. As the size of input data increases, we must tend toward the use of anyone as pyDatalog helps in reducing the redundant checks and has better time complexity for both the algorithms. (ii) is slightly faster then (i) when comparing the results we got.

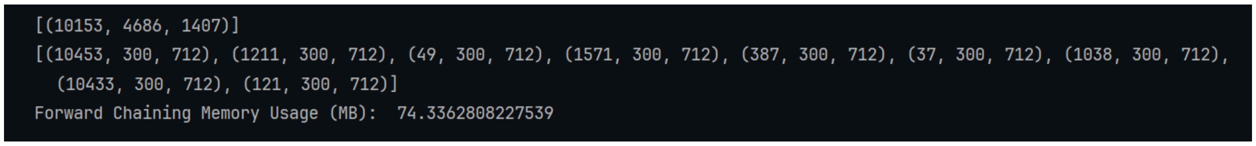


```
[(10153, 4686, 1407)]
[(10453, 300, 712), (1211, 300, 712), (49, 300, 712), (1571, 300, 712), (387, 300, 712), (37, 300, 712), (1038, 300, 712),
  (10433, 300, 712), (121, 300, 712)]
Forward Chaining Execution Time (s): 0.0060100555419921875
```

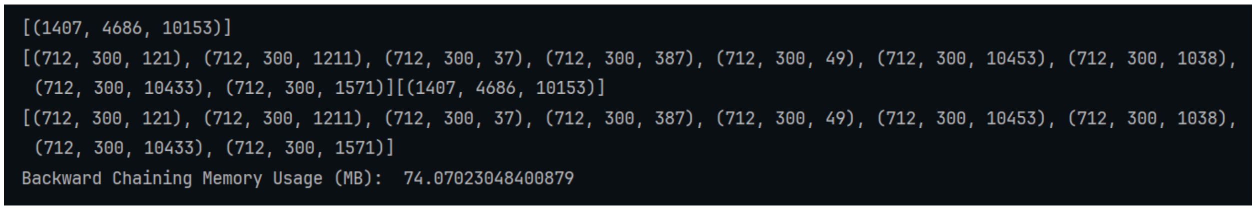


```
[(1407, 4686, 10153)]
[(712, 300, 121), (712, 300, 1211), (712, 300, 37), (712, 300, 387), (712, 300, 49), (712, 300, 10453), (712, 300, 1038),
 (712, 300, 10433), (712, 300, 1571)]
Backward Chaining Execution Time (s): 0.005767822265625
```

To Compare memory usage by both the algorithms the results I got for the public testcases shows that both are taking almost the same amount of Memory as both algorithms required pre-data initialization in the Pydatalog framework. That proves that both algorithms are taking similar time almost same.



```
 [(10153, 4686, 1407)]
 [(10453, 300, 712), (1211, 300, 712), (49, 300, 712), (1571, 300, 712), (387, 300, 712), (37, 300, 712), (1038, 300, 712),
   (10433, 300, 712), (121, 300, 712)]
 Forward Chaining Memory Usage (MB):  74.3362808227539
```



```
[(1407, 4686, 10153)]
[(712, 300, 121), (712, 300, 1211), (712, 300, 37), (712, 300, 387), (712, 300, 49), (712, 300, 10453), (712, 300, 1038),
 (712, 300, 10433), (712, 300, 1571)][(1407, 4686, 10153)]
[(712, 300, 121), (712, 300, 1211), (712, 300, 37), (712, 300, 387), (712, 300, 49), (712, 300, 10453), (712, 300, 1038),
 (712, 300, 10433), (712, 300, 1571)]
Backward Chaining Memory Usage (MB):  74.07023048400879
```

(b): Intermediate Steps involved in Forward Chaining were:

- Starts with Data and moves toward the target stop
- Uses data initialized in the datalog
- Used rules and facts to move towards the target stop
- Query from predicates to get resultant paths
- Append in final solution array

Similarly the intermediate steps involved in Backward Chaining were:

- Starts with Target stop and uses data to support
- The supporting facts and rules helps to run algorithm
- The facts and rules were already been intialized in datalog
- Uses predicates to query out results paths

- Appends in final solution array

These were intermediate steps involved in both algorithms.

(c): Overall Number of steps involved in both the algorithms is almost similar in terms of number steps just change in order of steps involved in both. The steps involved in forward chaining in beginning are in Reverse order in Backward Chaining. Like Forward Chianing first check if rule matches, query the results and check if goal reached but backward chaining first check if goal reached if not does it exists in known facts uses rules to search then move toward its goal.

Therefore The number of Steps involved in both is same just change in ordering of steps Both algorithms query results from initialized pyDatalog Framework.