AI Assignment 2

Ritika Thakur | 2022408

Theory

AI ASSIGNMENT — Knowledge, Representation, Reasoning and Planning

RITIKA THAKUR | 2022408

THEORY

Q1. G: Light is green — G_t: Light is green at time t
R: Light is red — R_t: Light is red at time t
Y: Light is yellow — Y_t: Light is yellow at time t

(a) $(G \wedge \sim R \wedge \sim Y) \vee (\sim G \wedge R \wedge \sim Y) \vee (\sim G \wedge \sim R \wedge Y)$

(b) $G_t \longrightarrow Y_{t+1}$
$Y_t \longrightarrow R_{t+1}$
$R_t \longrightarrow G_{t+1}$

(c) $(G_t \wedge G_{t+1} \wedge G_{t+2}) \longrightarrow \sim G_{t+3}$
$(Y_t \wedge Y_{t+1} \wedge Y_{t+2}) \longrightarrow \sim Y_{t+3}$
$(R_t \wedge R_{t+1} \wedge R_{t+2}) \longrightarrow \sim R_{t+3}$

Q2. Colour (x, c) : x node has colour c ; $x \in N, c \in C$
Edge (x, y) : there exists an edge $e \in R$ between x and y.
Yellow (x) : node x is coloured yellow ; $x \in N$
Red (x) : node x is coloured red ; $x \in N$
Green (x) : node x is coloured green ; $x \in N$
Clique (x, c) : node x belongs to clique with colour c; $x \in N, c \in C$

1. $\forall x, y\ (Edge(x,y) \longrightarrow \forall c\ (Colour(x,c) \longrightarrow \sim Colour(y,c)))$

2. $\exists x, y\ (x \neq y \wedge Yellow(x) \wedge Yellow(y) \wedge \forall z\ (Yellow(z) \longrightarrow (z=x \vee z=y)))$

3. $\forall x\ (Red(x) \longrightarrow \exists y\ (Green(y) \wedge Edge(x,y)) \vee \exists z\ (Edge(x,z) \wedge Edge(z,y)) \vee \exists p\ (Edge(x,z) \wedge Edge(z,p) \wedge Edge(p,y)) \vee \exists q\ (Edge(x,z) \wedge Edge(z,p) \wedge Edge(p,q) \wedge Edge(q,y))))$

4. $\forall c \in C\ \exists x\ (Colour(x,c))$

5. $\forall c \in C\ \exists x\ (Colour(x,c)) \wedge \forall x \forall y\ ((Clique(x,c) \wedge Clique(y,c)) \longrightarrow Edge(x,y) \vee (x=y)) \wedge \forall z\ \exists! c\ (Clique(z,c))$

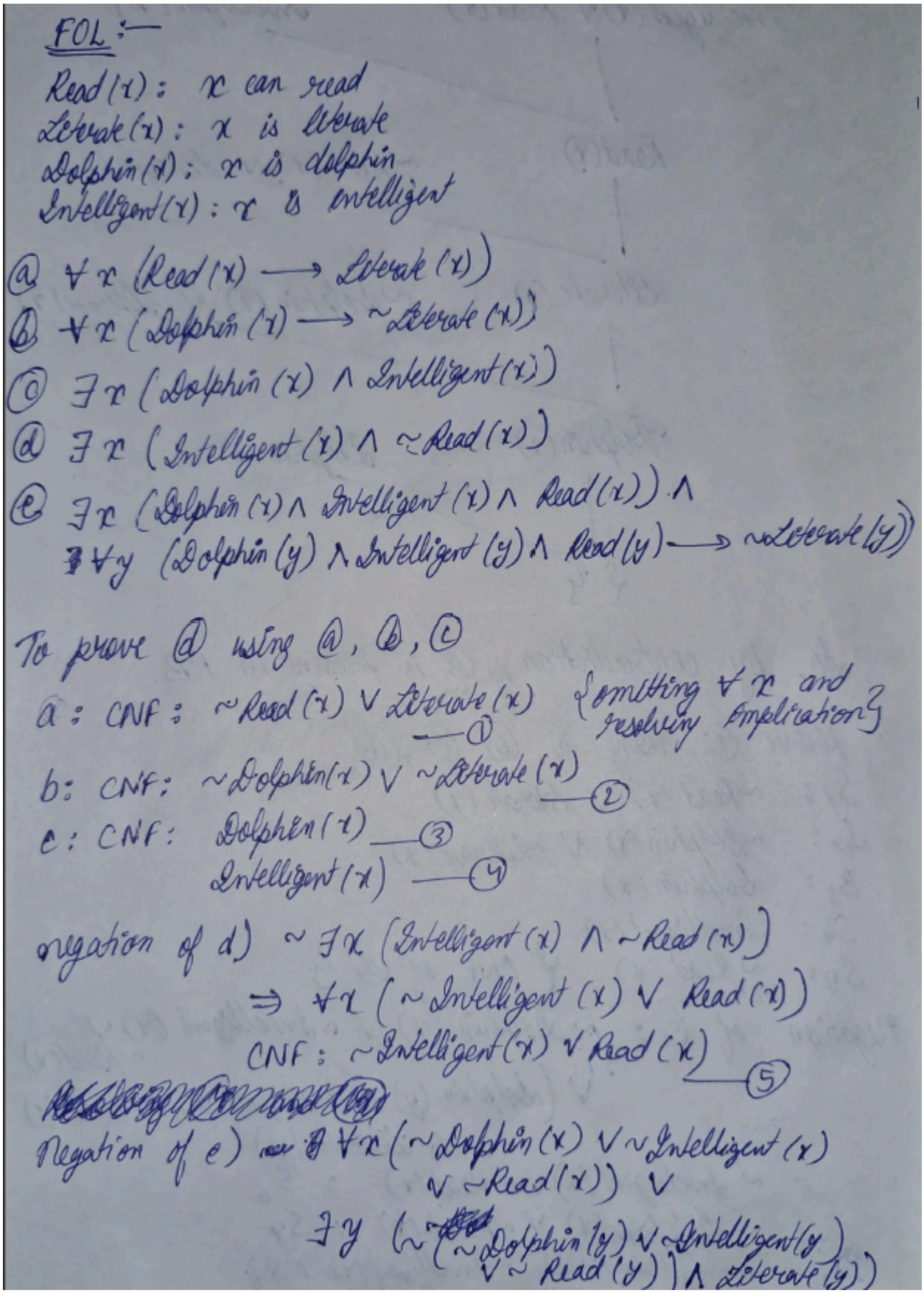
Q3. PL:-

R : one can read
L : one is literate
dolphin : constant

(a) $R \longrightarrow L$

(b) dolphin $\longrightarrow \sim L$

(c) (d) (e) cannot be represented in PL since we cannot represent "some", "all" etc quantifiers using PL.



FOL :—

Read(x) : x can read
Liberate(x) : x is liberate
Dolphin(x) : x is dolphin
Intelligent(x) : x is intelligent

(a) $\forall x$ (Read(x) $\rightarrow$ Liberate(x))

(b) $\forall x$ (Dolphin(x) $\rightarrow$ $\sim$Liberate(x))

(c) $\exists x$ (Dolphin(x) $\wedge$ Intelligent(x))

(d) $\exists x$ (Intelligent(x) $\wedge$ $\sim$Read(x))

(e) $\exists x$ (Dolphin(x) $\wedge$ Intelligent(x) $\wedge$ Read(x)) $\wedge$
$\exists \forall y$ (Dolphin(y) $\wedge$ Intelligent(y) $\wedge$ Read(y) $\rightarrow$ $\sim$Liberate(y))

To prove (d) using (a), (b), (c)

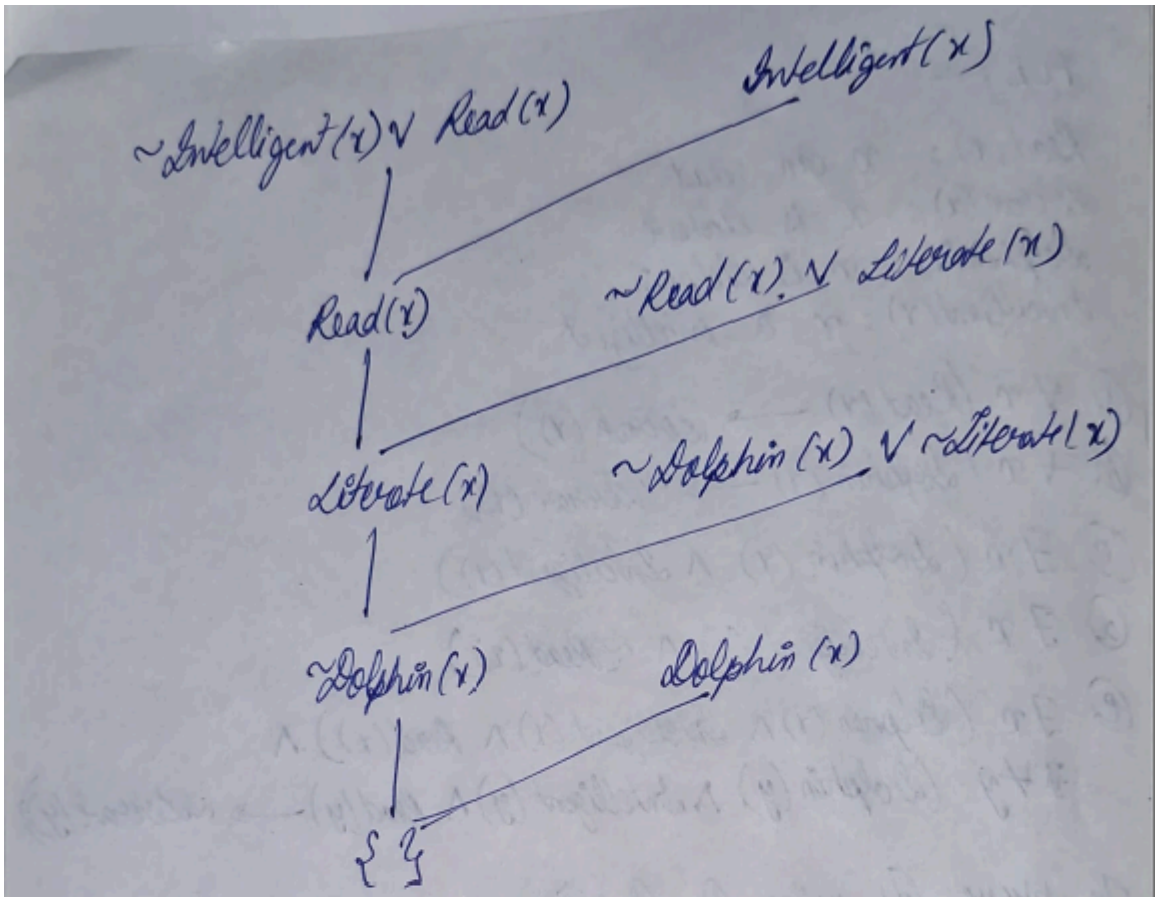
a : CNF : $\sim$Read(x) $\vee$ Liberate(x) —— (1) {omitting $\forall x$ and resolving implication}

b : CNF : $\sim$Dolphin(x) $\vee$ $\sim$Liberate(x) —— (2)

c : CNF : Dolphin(x) —— (3)
Intelligent(x) —— (4)

negation of d) $\sim \exists x$ (Intelligent(x) $\wedge$ $\sim$Read(x))
$\Rightarrow \forall x$ ($\sim$Intelligent(x) $\vee$ Read(x))
CNF : $\sim$Intelligent(x) $\vee$ Read(x) —— (5)

Negation of e) $\sim \exists \forall x$ ($\sim$Dolphin(x) $\vee$ $\sim$Intelligent(x)
$\vee$ $\sim$Read(x)) $\vee$
$\exists y$ ($\sim$($\sim$Dolphin(y) $\vee$ $\sim$Intelligent(y)
$\vee$ $\sim$Read(y)) $\wedge$ Liberate(y))



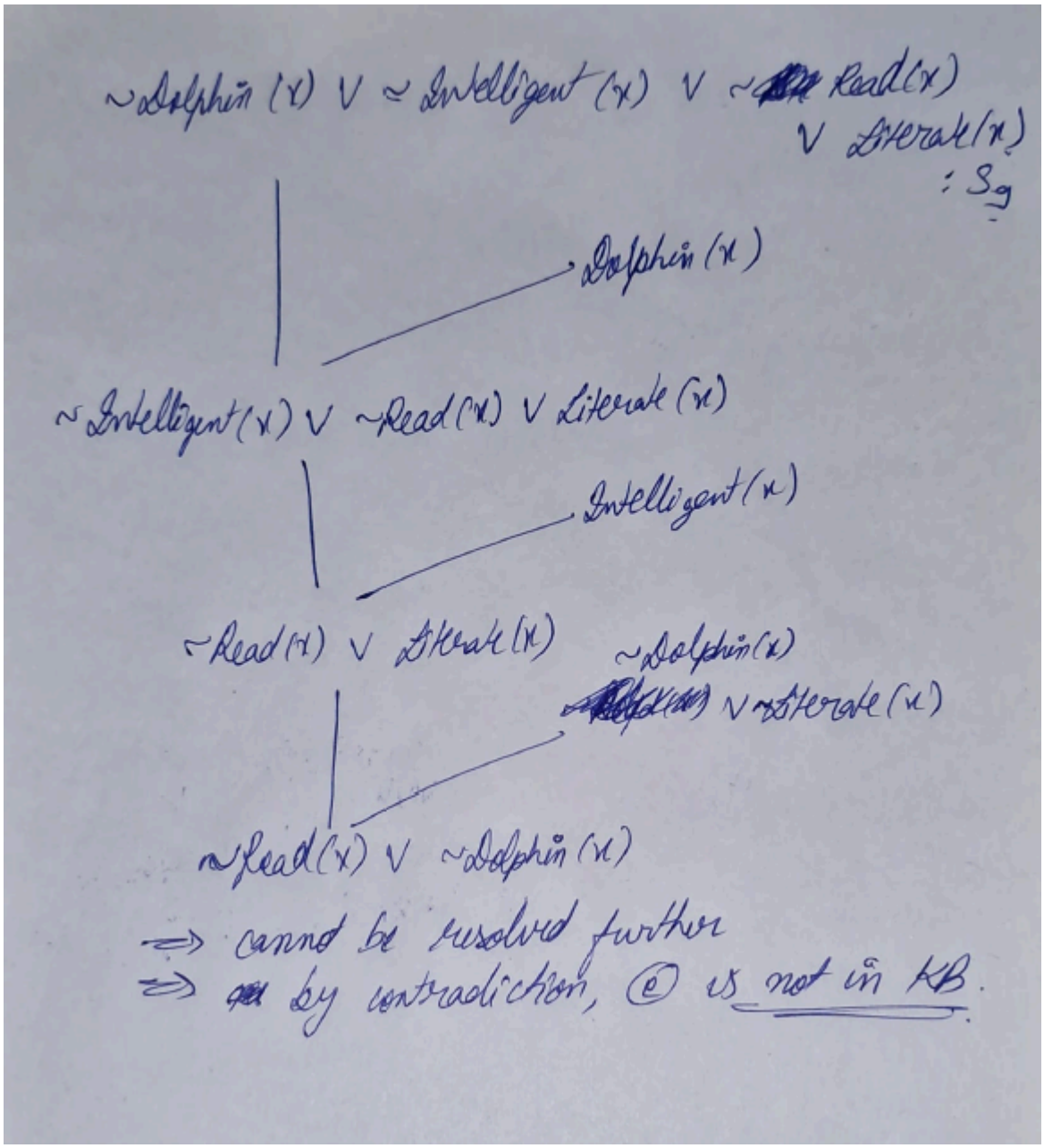
⇒ So, by contradiction, Ⓓ is present in KB.

To prove Ⓒ using Ⓐ, Ⓑ, Ⓒ, Ⓓ.

S_1: ~Read(x) ∨ Literate(x)
S_2: ~Dolphin(x) ∨ ~Literate(x)
S_3: Dolphin(x)
S_4: Intelligent(x)
S_5: ~Read(x) { CNF of Ⓓ }

Negation of Ⓒ: ~Dolphin(x) ∨ ~Intelligent(x) ∨ ~Read(x)
∨ (Dolphin(y) ∧ Intelligent(y) ∧ Read(y) ∧ Literate(y))

= ~Intelligent(x) ∨ ~Read(x) : S_6
~Dolphin(x) ∨ ~Read(x) : S_7
~Dolphin(x) ∨ ~Intelligent(x) : S_8



Computational

Question 1

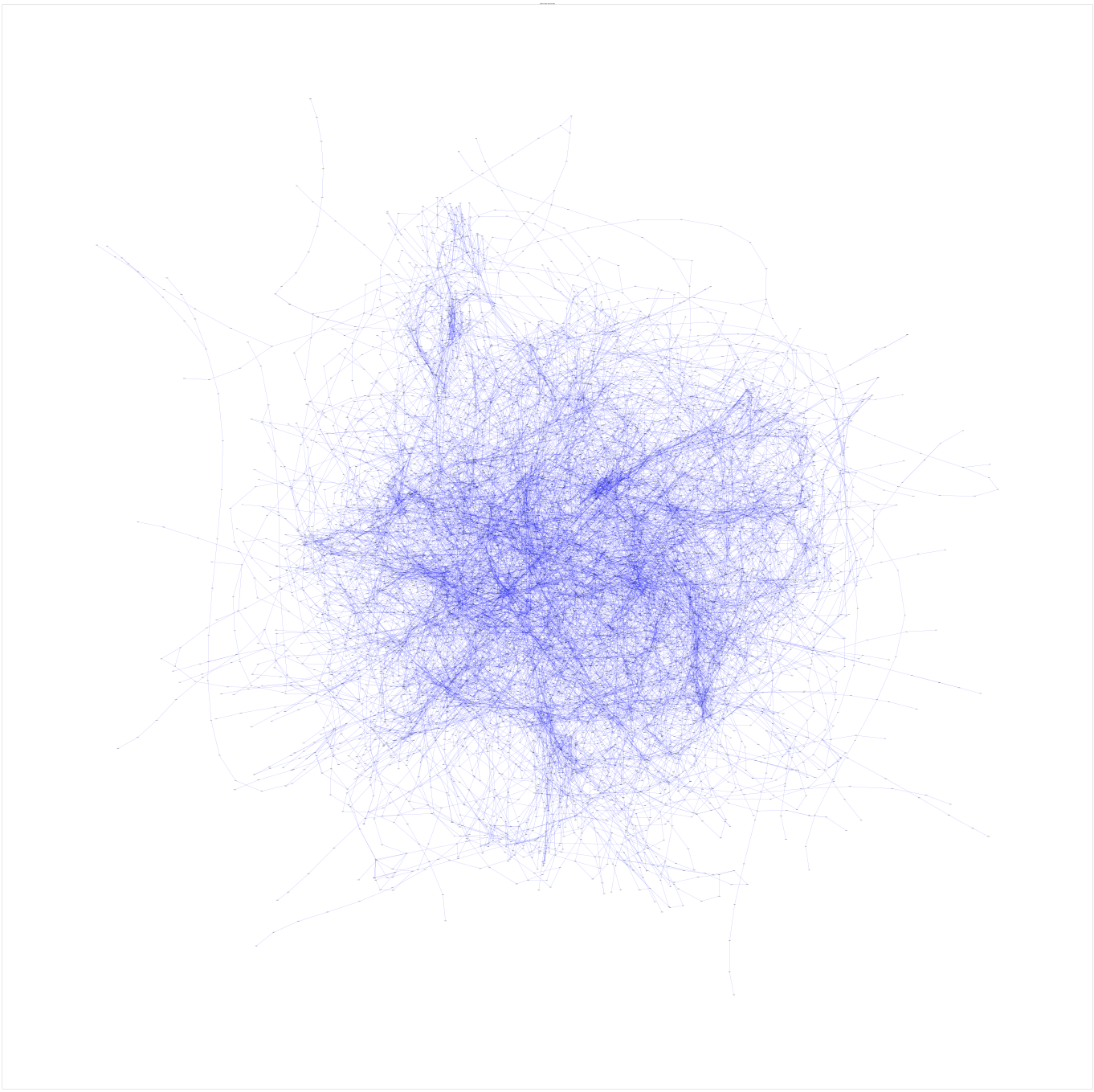
-> Loading static data: in code file
-> Setting up KB: in code file
-> (a) Top 5 busiest routes based on the number of trips:
[(5721, 318), (5722, 318), (674, 313), (593, 311), (5254, 272)]

(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) Pairs of stops (start and end) that have only one direct route between them:
[((233, 148), 1433), ((11476, 10060), 5867), ((10225, 11946), 5436), ((11044, 10120), 5916), ((11045, 10120), 5610)]

-> Graph representation for KB:



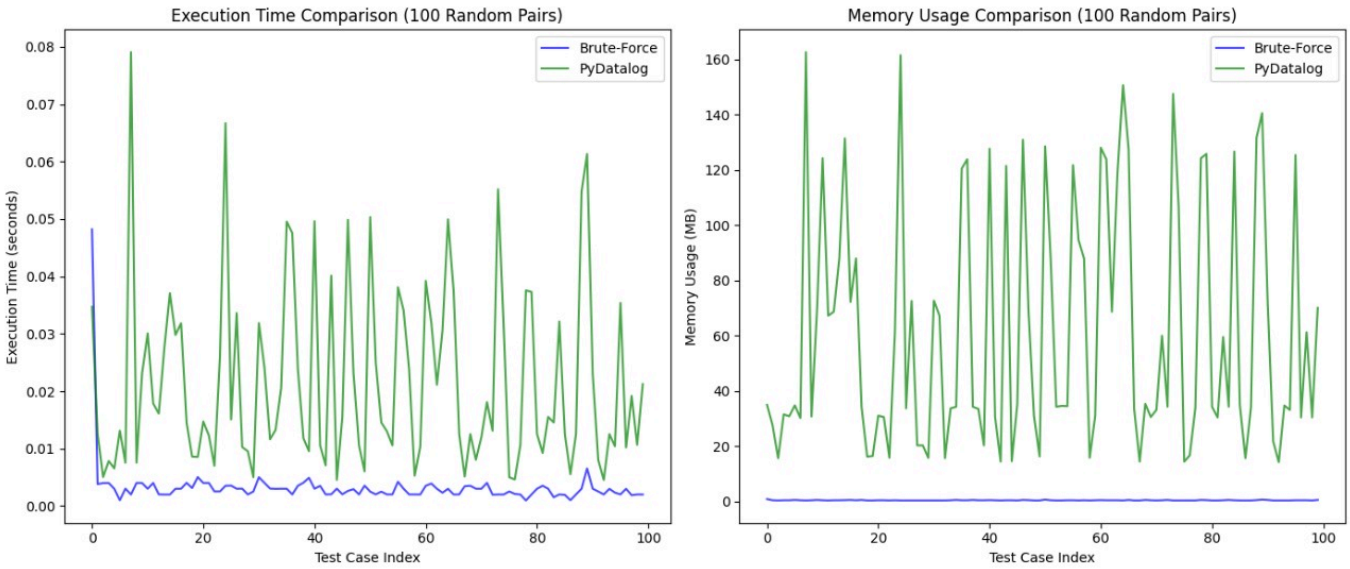
Question 2

-> Brute force for direct route: in code file

-> FOL based reasoning for direct route: in code file

-> (a) Time complexity and Space complexity comparison:

```
Brute-Force Average Execution Time: 0.00329029560089111133 seconds
Brute-Force Average Memory Usage: 0.421279296875 MB
PyDatalog Average Execution Time: 0.02164255142211914 seconds
PyDatalog Average Memory Usage: 61.23416015625 MB
```



As we can notice, average execution time and average memory usage for FOL based reasoning is higher than brute force. However, there is a significant difference in the number of intermediate steps required for both the methods.

(b) Number of intermediate steps: Due to the iterative nature of the brute force method, the number of steps is very high (over 2500). On the other hand, the FOL based reasoning method requires only 21 intermediate steps.

```
Checking route: 142 with stops [146, 148, 149, 488, 233, 915, 916, 2161, 2162, 3569, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177]

Checking route: 10001 with stops [3928, 3929, 3930, 20001, 20002, 20003, 20004, 20005, 545, 1998, 1999, 2000, 2001, 998, 999, 2149, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 14
  Found start stop 2573 and end stop 1177 in route 10001
  Start stop index: 50, End stop index: 56
  Valid route 10001 (start stop comes before end stop)

Checking route: 362 with stops [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 756, 19, 20, 21, 1145, 1147, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 132

Checking route: 274 with stops [3026, 4100, 1174, 1965, 1966, 1967, 228, 229, 230, 231, 232, 233, 2031, 148, 149, 150, 151, 22514, 22515, 154, 155, 156, 157, 158, 159, 160, 161, 162,

Checking route: 10386 with stops [484, 485, 486, 487, 149, 150, 151, 22514, 22515, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 5094, 164, 165, 166, 167, 168, 169, 2266, 3531, 3

Checking route: 233 with stops [4495, 21, 1145, 1146, 1147, 1148, 1914, 1915, 1916, 22516, 1918, 1919, 1920, 1921, 1922, 1923, 3985, 1924, 1925, 1926, 1927, 2937, 2938, 2939, 22517,

Checking route: 10387 with stops [1655, 1656, 1657, 612, 510, 638, 2641, 2642, 2643, 2656, 2657, 2658, 4937, 2659, 20177, 3125, 4875, 3126, 3127, 2620, 2019]

Checking route: 118 with stops [69, 1351, 4347, 374, 375, 376, 377, 378, 22519, 379, 380]

Checking route: 10388 with stops [1285, 1286, 1287, 1288, 1289, 830, 1290, 1291, 1292, 1293, 1294, 1295, 1114, 1115, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306,

Checking route: 67 with stops [752, 22520, 22521, 755, 2852, 2853, 2854, 2855, 2856, 270, 271, 0, 1, 3102, 3103, 3104]

Checking route: 1627 with stops [239, 1244, 241, 242, 243, 1245, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262]

Checking route: 1047 with stops [752, 22520, 22521, 755, 756, 757]

Checking route: 10389 with stops [5116, 2934, 2935, 2936, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 22516, 1918, 1919, 1920, 1921, 1922, 1923, 3985, 1924, 1925, 1926, 1927, 192
```

Intermediate steps for Brute force method

```
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('2132','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10541','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1297','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1651','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10662','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1003','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1117','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1117','1177')
INFO:pyDatalog.pyEngine:New fact : !=('2573','1177') is True
INFO:pyDatalog.pyEngine:New fact : DirectRoute('2573','1177','1117')
INFO:pyDatalog.pyEngine:New fact : _pyD_query356('1117')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('2057','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10001','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10001','1177')
INFO:pyDatalog.pyEngine:New fact : DirectRoute('2573','1177','10001')
INFO:pyDatalog.pyEngine:New fact : _pyD_query356('10001')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1407','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1407','1177')
INFO:pyDatalog.pyEngine:New fact : DirectRoute('2573','1177','1407')
INFO:pyDatalog.pyEngine:New fact : _pyD_query356('1407')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10580','2573')
[1117, 1407, 10001]
```

`Intermediate steps for FOL based reasoning method` (c) Comparing the overall number of steps logically,

Brute force: Iterates over routes to stops dictionary => checks if start and end stops are present in routes => checks if start and end stop index present in correct order => appends route to direct routes list => returns direct routes list.

FOL based reasoning: Initialising Datalog terms and predicates => Adding route data to Datalog => Initialise datalog predicates => Populate the knowledge base => Query direct route => Return direct route.

Even though the number of intermediate steps is lower for FOL based reasoning, overall steps are higher due to the complexity of initialising datalog predicates and knowledge base as well as the inference that the library has to perform. THis is also a reason why FOL has larger time and memory consumption.

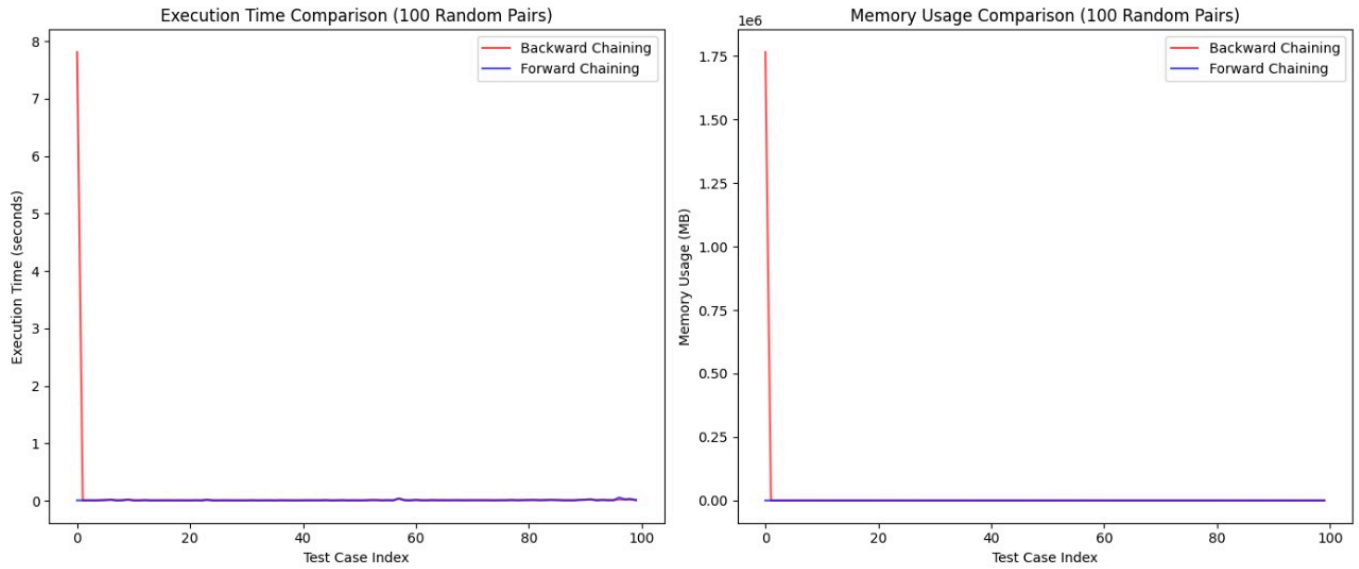
Question 3

-> Forward Chainining: in code file

-> Backward Chainining: in code file

-> (a) Time complexity and Space complexity comparison:

```
Backward Chaining Average Execution Time: 0.08695717334747315 seconds
Backward Chaining Average Memory Usage: 17673.33171875 MB
Forward Chaining Average Execution Time: 0.00922823190689087 seconds
Forward Chaining Average Memory Usage: 18.810693359375 MB
```



As we can notice, the average execution time and average memory usage for backward chaining is much higher than forward chaining. This is also justified by the nuber of intermediate and overall steps required for both the methods.

(b) Number of intermediate steps: Number of intermediate steps for forward chaining is 19 and for backward chaining is 23. This is because forward chaining starts with the given facts and iterates over the rules to find the conclusion. On the other hand, backward chaining starts with the goal and iterates over the rules to find the facts that support the goal.\

```
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1859','22540')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('32','22540')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('2095','22540')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10004','22540')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10642','22540')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10156','22540')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10153','22540')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10153','4686')
INFO:pyDatalog.pyEngine:New fact : !=('22540','4686') is True
INFO:pyDatalog.pyEngine:New fact : DirectRoute('22540','4686','10153')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10153','4686')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('579','4686')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1407','4686')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1407','2573')
INFO:pyDatalog.pyEngine:New fact : !=('4686','2573') is True
INFO:pyDatalog.pyEngine:New fact : DirectRoute('4686','2573','1407')
INFO:pyDatalog.pyEngine:New fact : !=('10153','1407') is True
INFO:pyDatalog.pyEngine:New fact : OptimalRoute('22540','2573','10153','4686','1407')
INFO:pyDatalog.pyEngine:New fact : _pyD_query357('10153','1407')
[(10153, 4686, 1407)]
```

Intermediate steps for Forward chaining method

```
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('2132','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10541','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1297','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1651','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10662','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1003','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1117','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('2057','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10001','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1407','2573')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1407','4686')
INFO:pyDatalog.pyEngine:New fact : !=('2573','4686') is True
INFO:pyDatalog.pyEngine:New fact : DirectRoute('2573','4686','1407')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10153','4686')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10153','22540')
INFO:pyDatalog.pyEngine:New fact : !=('4686','22540') is True
INFO:pyDatalog.pyEngine:New fact : DirectRoute('4686','22540','10153')
INFO:pyDatalog.pyEngine:New fact : !=('1407','10153') is True
INFO:pyDatalog.pyEngine:New fact : OptimalRoute('2573','22540','1407','4686','10153')
INFO:pyDatalog.pyEngine:New fact : _pyD_query358('1407','10153')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('579','4686')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('1407','4686')
INFO:pyDatalog.pyEngine:New fact : RouteHasStop('10580','2573')
[(1407, 4686, 10153)]
```

`Intermediate steps for Backward chaining method` (c) Comparing the overall number of steps logically,

Forward chaining: Iterates over rules to find the conclusion => checks if the conclusion is present in the facts => appends the conclusion to the facts list => returns the facts list.

Backward chaining: Iterates over rules to find the facts that support the goal => checks if the facts are present in the given facts => appends the facts to the goal list => returns the goal list.

Therefore, the overall steps and the time and memory consumption for backward chaining is much higher than forward chaining.