AI ASSIGNMENT — Knowledge Representation, Reasoning and Planning
Deadline: 11:59 PM, 04/11/2024

Total Marks : 100 Marks Weightage: 10%

Instructions:

1. Assignments are to be attempted individually.
2. Submit the assignment as a single zipped folder (assignment2 RollNumber.zip) containing a PDF file (report RollNumber.pdf) for the theory questions, anal ysis, and methodology of the programming tasks, along with a Python file (code RollNumber.py) for the programming solutions. Failure to follow the nam ing convention for the files will result in zero marks, so ensure that your submission adheres to the required format.
3. Please read the instructions given in the questions carefully. In case of any ambiguity, post your queries on Google Classroom at least a week before the deadline. No TA will be responsible for responding to the queries after this.
4. A part of the assignment evaluation involves automatic testing of your submitted code on private test cases. Please make sure that you do not change the structure of the methods provided in the boilerplate code.
5. You will be provided with a separate test script containing public test cases to help you ensure that your code runs correctly. The same script will be used during evaluations for both public and private test cases. Make sure your code runs successfully with this script to avoid any issues during evaluation.
6. All the TAs will strictly follow the rubric provided. No requests will be enter tained related to the scoring strategy.
7. The use of generative tools (such as ChatGPT, Gemini, etc.) is strictly prohibited. Failure to comply may result in severe consequences related to plagia rism.
8. Extension and Penalty clause:
    - Submissions made even 1 minute past the deadline on Google Classroom will be marked as late. To avoid this, please ensure your work is submitted at least 5 minutes before the deadline.
    - Not explaining the answers properly will lead to zero marks.

Theory (30 marks)

1. The traffic light at the busy intersection has a dramatic life—it's always flipping between colors, and it needs your help to formalize its behavior. Here's what we know:

   - At any given moment, the traffic light is either green, yellow, or red. *It's never in more than one state at a time.*
   - The traffic light switches from green to yellow, yellow to red, and red to green. *There's no creative jumping around in the sequence—it sticks to its routine.*
   - The traffic light cannot remain in the same state for more than 3 con secutive cycles. *It gets bored easily.*

   Represent these rules (highlighted in bold) clearly using Propositional Logic (PL). (1+2+2) = 5 marks

2. You've been hired as the official color master of a strange colored graph-based world where nodes dream of standing out. Some of them have already chosen their colors, while others are still undecided and waiting for your expert guidance!

   Let $\{c_1, ..., c_k\}$ be a non-empty and finite set of colors. A partially colored directed graph is a structure N, R, C where

   - N: A non-empty set of nodes, *some of whom already have colors, while others are still figuring out their fashion choices.*
   - R: A set of directed edges representing connections between these nodes—*basically, who's linked to whom.*
   - C: A color palette, $\{c_1, ..., c_k\}$, that the nodes can choose from. However, not all the nodes are necessarily colored, and each node has at most one color. *No outfit changes!*

   But, like any good world, there are rules (and we all know rules make things more fun). Here's what the nodes have agreed upon:

   1. Connected nodes don't have the same color. *No node wants to be caught wearing the same outfit as their neighbor—it's the ultimate faux pas!*
   2. Exactly two nodes are allowed to wear yellow. *Yellow is rare, and only two nodes can pull it off.*
   3. Starting from any red node, you can reach a green node in no more than 4 steps. *The red nodes have been told to keep a close eye on the greens.*
   4. For every color in the palette, there is at least one node with this color. *No color should be left behind; each deserves at least one representative.*
   5. The nodes are divided into exactly $|C|$ disjoint non-empty cliques, one for each color. *Each color gets its own squad, and no clique is left empty.*

Develop a First-Order Logic (FOL) language and set of axioms that formalize these rules. Make sure to represent each of the statements (highlighted in bold) precisely and help the nodes of this graph live in harmony, following the rules of their colorful world! (10 marks)

3. Our friends in the ocean are having an intellectual argument about reading, literacy, and intelligence. Here are the statements of interest:

   • Whoever can read is literate *(easy enough, right?)*

   • Dolphins, unfortunately, are not literate *(they've tried, though)*. • Some dolphins are intelligent *(think of them as the Socrates of the sea)*. • Some who are intelligent cannot read *(cue the dolphin groans)*.

   • There exists a dolphin who is both intelligent and can read (finally, a hero!), but for every intelligent dolphin, if it can read, it must be that it is not literate (plot twist!).

   Represent these statements (highlighted in bold) using both PL and FOL. Define appro priate propositional variables and predicates to capture this dolphin debate. (2.5*2=5 marks)

   But wait! We also need you to resolve a deep-sea conundrum: check whether the fourth and fifth statements are satisfiable using resolution refutation. Let the dolphins be your muse as you swim through logic! (5*2=10 marks) Note: to check the satisfiability of the fourth sentence use only the first three sentences and when you prove the fifth sentence, use remaining fours

Computational (70 marks)

Ah, Delhi buses—the mighty chariots of commuters! In this assignment, you will be build ing a transit data application to navigate the complex network of routes, trips, and stops. The challenge? Ensure your application can effectively handle this data, providing clear and accurate transit information. For this, you are required to use the GTFS static data from Delhi's Open Transit Data (OTD). The dataset consists of several static data files: routes.txt, trips.txt, stop times.txt, stops.txt, and fare rules.txt. For your con venience, the attributes you will use and the relationships between these files are illustrated in Figure 1.

Key terms:

- Start Stop: The stop ID where your journey begins.
- End Stop: The stop ID where your journey concludes.
- Route: A sequence of bus stops, identified by their stop IDs, that form a specific path from the Start Stop to the End Stop.
- Intermediate/Via Stop: A stop ID where the user is required to pause during the journey. This stop is essential for the route and may also serve as a point for

transferring between routes.

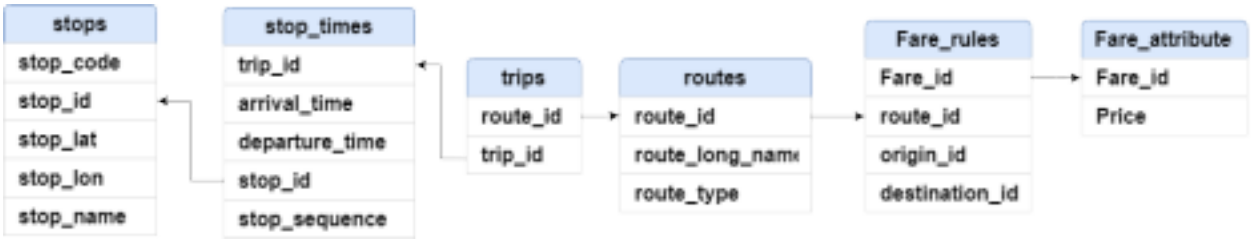

Figure 1: Static data file structure for Delhi buses

- Interchange: The process of switching from one route to another at a specific stop ID. An interchange can occur at an Intermediate/Via Stop but is not limited to it.

The boilerplate code can be downloaded from this link. *The output of a few public test cases will be provided to you. The final scoring will be done using multiple public and private test cases.*

1. *Data Loading and Knowledge Base Creation* (10 marks)
   Think of this part as preparing the bus system's "brain"—organizing everything so that the application can later reason and plan effectively.

   - Load the provided OTD static data. Ensure all data is stored in well-structured Python data types. Convert data types as necessary (e.g., time as datetime objects, IDs as strings).
   - Set up the knowledge base (KB) for reasoning and planning tasks. For this you would need to create the following dictionaries: route to stops = *{*route id: [list of stop ids]*}*, trip to route = *{*trip id: [list of route ids]*}*, stop trip count = *{*stop id: count of trips stopping there*}*.
   - If your KB is correctly set up, then you should be able to answer the following: (a) Top 5 busiest routes based on the number of trips.

     (b) Top 5 stops with the most frequent trips.

     (c) The top 5 busiest stops based on the number of routes passing through them. (d) The top 5 pairs of stops (start and end) that are connected by exactly one direct route, sort them by the combined frequency of trips passing through both stops.

   Additionally, create a graph representation using plotly for the knowledge base you created, and use the route to stops mapping for the same.

2. *Reasoning* (30 marks)
   Now that you have organized the data, it's time to put that knowledge to good use. Your goal is to implement a function DirectRoute(start stop, end stop) that takes a *start stop* and *end stop* as inputs and returns all the direct routes between them (i.e., no interchanges).

Input Format - (start stop id, end stop id), *Output Format* - [list of route id's]

You are required to implement using the following methods:

1. *Brute-Force Approach*: Develop a straightforward reasoning algorithm from scratch. Note: *The logic is procedural. We systematically outline the steps for reasoning.* For example:

```
if directRoute(x, y):
    adddirectRoute(x, y)
else:
    dontAdd(x, y)
```

2. *FOL Library-Based Reasoning*: Utilize the PyDatalog library to implement it. For this, you need to create terms, define the predicates, add facts to the above knowl edge base, and then define query functions. Note: *The logic is declarative. We define relationships (facts) and rules, and the system infers the rest.*
   For example:

```
    create_terms(AddToKB, ValidRoute, Fun1, Fun2, X, Y, Z)
DirectRoute(X, Y) <= Fun1(X, Z) & Fun2(Z, Y)
+ AddToKB(x, y)
queryDirectRoute(start, end)
```

In your analysis of both implementations, (a) evaluate time complexity in terms of execution time (measured in seconds) and memory usage (measured in megabytes, MB), (b) examine the intermediate steps in your reasoning process, and (c) compare the overall number of steps involved in both implementations.

3. *Planning* (30 marks)

In this task, you will plan optimalRoute for users traveling between two bus stops. The goal is to optimize the routes based on constraints using:

1. *Forward Chaining*
2. *Backward Chaining*

Use the PyDatalog library for the same. The constraints to optimize the routes are as follows:

1. INCLUDE VIA STOP: The path from the start-stop to the end stop must include at least one intermediate stop (via stop). If no valid via stop is found, the result

should be an empty list.

2. NUM INTERCHANGE: The optimal path returned should only contain one route interchange.

In your analysis of both implementations, (a) evaluate time complexity in terms of execution time (measured in seconds) and memory usage (measured in megabytes, MB), (b) examine the intermediate steps in your reasoning process, and (c) compare the overall number of steps involved in both implementations.

4. Bonus Questions (20 marks)

1. Implement the previous planning problem using Planning Domain Definition Language (PDDL) using forward chaining. Use the PyDatalog library for the same.

   Key aspects:

   • Initial State: Define the initial state as the stop where the journey begins (e.g., start stop id).
   • Goal State: The goal state is to reach the destination stop (e.g., end stop id). • Action: You have two primary actions in this route planning:
   – Board a route: This action allows you to board a specific route at a stop. Example: Action('board route', R, X) means boarding route R at stop X. – Transfer between routes: This action allows you to switch from one route to another at a stop. Example: Action('transfer route', R_1, R_2, Z) means transferring from route R_1 to route R_2 at stop Z (the transfer stop).

   Print the current state information at each step. In your analysis, (a) evaluate time complexity in terms of execution time (measured in seconds) and memory usage (measured in megabytes, MB), (b) examine the intermediate steps in your reasoning process, and (c) compare the overall number of steps involved in both implementations. Do all algorithms (forward chaining, backward chaining, and PDDL) produce the same optimal route, or do some produce suboptimal routes due to the way constraints are applied?

2. Extend the previous optimalRoute planning problem by considering the fare attribute (Fare). Return the optimal path possible between the given start stop id and end stop id, which costs within the given initial fare, considering the constraints as initial fare and max transfers (maximum route interchange possible). Return the empty list if the path route fare exceeds the given initial fare.
   Note: Use fare rules and fare attributes static files to create the knowledge base in which route id maps to the fare. This task can be computationally expensive, so optimize it using pruning techniques that eliminate irrelevant or redundant paths in the solution space to enhance overall efficiency. For example, in a search algorithm like depth-first search, if you encounter a node that has

already been visited, you can prune that path from further exploration, thereby reducing the total number of nodes processed and speeding up the search.