Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vellore – 632014, Tamil Nadu, India

SCHOOL OF COMPUTER SCIENCE AND ENGINEERNG

Fall Semester 2024-25

BCSE306L – Artificial Intelligence

DIGITAL ASSIGNMENT-2

Submission on or before: 22-11-2024

Submission Requirements: (This is Submission task, instead of Quiz-2)

A hard copy of the code and the report should be printed and submitted to the Faculty @ SJT510-A31.

Assignment-II Details
Questions: **Objective 1:** To solve the 8 Puzzle problem using the Hill Climbing algorithm in Python, with heuristic guidelines and analyze its limitations, including local maxima and plateaus. **Question:** Implement the Hill Climbing algorithm in Python to solve the 8 Puzzle problem, starting from a given initial configuration and aiming to reach a specified goal state. Use a heuristic function, either the Manhattan distance or the number of misplaced tiles, to guide the search process. Test your implementation with different initial configurations, document the steps taken to reach the solution, and analyze cases where the algorithm fails due to local maxima or plateaus. Discuss any observed limitations of the Hill Climbing approach in solving the 8 Puzzle problem. --- **Objective 2:** To create Prolog rules for determining student eligibility for scholarships and exam permissions based on attendance, integrating these rules with a REST API and web app for querying eligibility and debar status. **Question:** Create a system in Prolog to determine student eligibility for scholarships and exam permissions based on attendance data stored in a CSV file. Write Prolog rules to evaluate whether a student qualifies for a scholarship or is permitted for exams, using specified attendance thresholds. Load the CSV data into Prolog, define the rules, and expose these eligibility checks through a REST API that can be accessed by a web app. Develop a simple web interface that allows users to input a student ID and check their eligibility status. Additionally, write a half-page comparison on the differences between SQL and Prolog for querying, focusing on how each handles data retrieval, logic-based conditions, and use case suitability. ***Note:*** 1. CSV data sample (Student_ID, Attendance_percentage, CGPA) 2. Load data in Proglog

(:- use_module(library(csv)).

csv_read_file("data.csv", Rows, [functor(student), arity(4)]), mpalist(assert, Rows).

3. Define rules in Prolog

   Eligible_for_Scholarship(Student_ID) :- student(Student_ID,_, Attendance_percentage, CGPS), Attendance_Percentage>=75, CGPA >=9.0.

   Permitted_for_exam(Student_ID) :- student(Student_ID, _, Attendance, _), Attendance >= 75.

4. REST_API with HTTP Library
5. Web App to call the API.

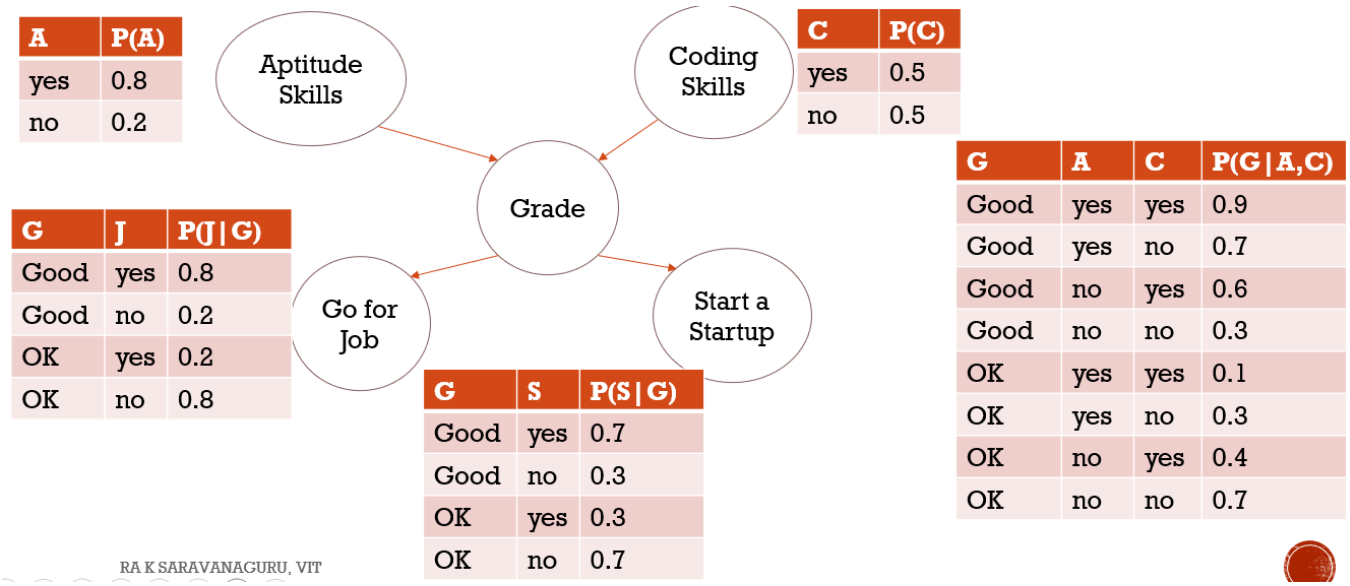
Objective 3: To use Monte Carlo simulation in Python to solve inference problems in a given Bayesian Belief Network (BBN) and analyze the results.

Question:

Given a Bayesian Belief Network (BBN) with defined nodes and conditional probability distributions, implement a Monte Carlo simulation in Python to perform inference on this network. Select a target node for which you want to compute the probability given evidence on one or more other nodes. Run the Monte Carlo simulation by generating random samples and estimating the conditional probability of the target node based on these samples. Provide the computed probability and discuss the accuracy of the result by comparing it to known values (if available) or explaining how sample size affects convergence in your simulation.

Problem :

As discussed in the class



A	P(A)
yes	0.8
no	0.2

C	P(C)
yes	0.5
no	0.5

G	A	C	P(G | A,C)
Good	yes	yes	0.9
Good	yes	no	0.7
Good	no	yes	0.6
Good	no	no	0.3
OK	yes	yes	0.1
OK	yes	no	0.3
OK	no	yes	0.4
OK	no	no	0.7

G	J	P(J | G)
Good	yes	0.8
Good	no	0.2
OK	yes	0.2
OK	no	0.8

G	S	P(S | G)
Good	yes	0.7
Good	no	0.3
OK	yes	0.3
OK	no	0.7

Sample Python Code with Cloudy, Sprinkler, Rain and WetGrass case study:

import numpy as np

Define conditional probabilities

P_Cloudy = 0.5

P_Sprinkler_given_Cloudy = {True: 0.1, False: 0.5}

```
P_Rain_given_Cloudy = {True: 0.8, False: 0.2}
P_WetGrass_given_Sprinkler_Rain = {
    (True, True): 0.99,
    (True, False): 0.90,
    (False, True): 0.80,
    (False, False): 0.00
}

# Monte Carlo simulation to estimate P(WetGrass=True | Rain=True)
def monte_carlo_simulation(num_samples=10000):
    count_wet_grass_given_rain = 0
    count_rain = 0

    for _ in range(num_samples):
        # Sample Cloudy
        cloudy = np.random.rand() < P_Cloudy

        # Sample Sprinkler given Cloudy
        sprinkler = np.random.rand() < P_Sprinkler_given_Cloudy[cloudy]

        # Sample Rain given Cloudy
        rain = np.random.rand() < P_Rain_given_Cloudy[cloudy]

        # Sample WetGrass given Sprinkler and Rain
        wet_grass = np.random.rand() < P_WetGrass_given_Sprinkler_Rain[(sprinkler, rain)]

        # Check if Rain is True and accumulate counts
        if rain:
            count_rain += 1
            if wet_grass:
                count_wet_grass_given_rain += 1
    # Calculate conditional probability
    if count_rain == 0:
        return 0  # Avoid division by zero
    return count_wet_grass_given_rain / count_rain
# Run simulation
estimated_probability = monte_carlo_simulation()
print(f"Estimated P(WetGrass=True | Rain=True): {estimated_probability}")
```