Batch: C1	**Roll No.: 16010121069**
Experiment / assignment / tutorial No. 5	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: 5 To implement temporal difference approach

AIM: T-D learning and prediction

Expected OUTCOME of Experiment: (Mention CO attained here): CO3

Books/ Journals/ Websites referred:

Richard S. Sutton and Andrew G. Barto, "*Reinforcement Learning:An Introduction",* The MIT Press,Second Edition, 2018

Pre Lab/ Prior Concepts:

TD methods learn directly from episodes of experience. TD is model-free: no knowledge of MDP transitions / rewards is required. TD learns from incomplete episodes, by bootstrapping. TD updates a guess towards a guess. TD can learn before knowing the final outcome. TD can learn online after every step.

Chosen Problem Statement: Driving Home

Policy:

The policy defines how the agent (the driver) selects actions at each state. For example, the agent might decide which route to take at each intersection or whether to drive faster or slower. In an epsilon-greedy policy:

- With a probability 1−ϵ1 - \epsilon1−ϵ, the agent chooses the action that minimizes the expected travel time based on its current knowledge (exploitation).
- With a probability ϵ\epsilonϵ, the agent explores other routes to potentially find better options (exploration).

Reward function:

The reward function quantifies feedback based on the time taken or cost associated with each route:

- **Negative Reward**: The reward could be the negative of the time taken to complete a segment of the journey. The agent receives larger penalties (negative rewards) for longer routes.
- **Positive Reward**: In some cases, reaching home within a specific time frame might offer a positive reward, encouraging the agent to optimize the driving time.

This guides the agent toward minimizing the total travel time or cost.

Value function:

The value function estimates the expected total reward (in this case, minimizing travel time) starting from a given state and following a specific policy. For example, the value at a state represents the expected total time remaining to reach home. The value function helps the agent decide which route is optimal based on current knowledge.

- **State Value Function (V(s))**: The expected remaining travel time starting from state s.
- **Action Value Function (Q(s, a))**: The expected remaining time when taking action a at state s.

Model of the environment:

The environment models the external conditions that affect the agent's driving decisions, such as:

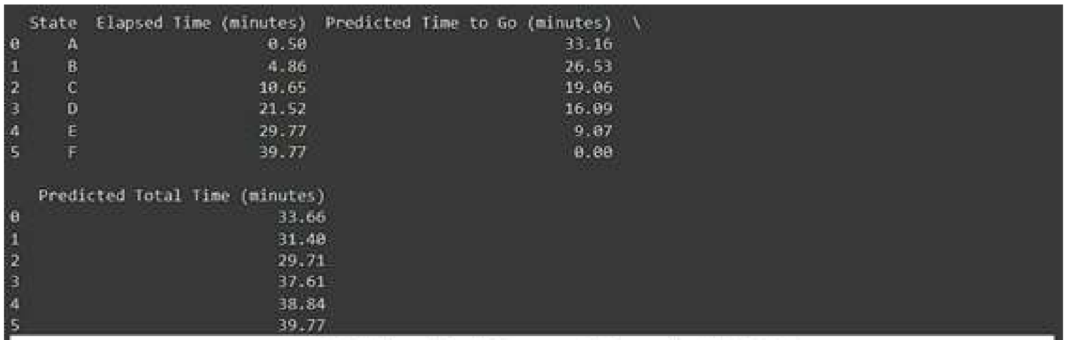
- Road Network: The available routes between the starting point and home.
- Traffic Conditions: Variable traffic that can cause delays or speed up certain routes.
- Weather: External conditions like rain or snow that may slow down driving.
- Signals and Rules: Stop signs, speed limits, or traffic lights that influence the time taken at intersections.

The agent interacts with this environment by selecting actions (routes, speeds), receiving rewards (based on time or efficiency), and observing the next state (new location and conditions).

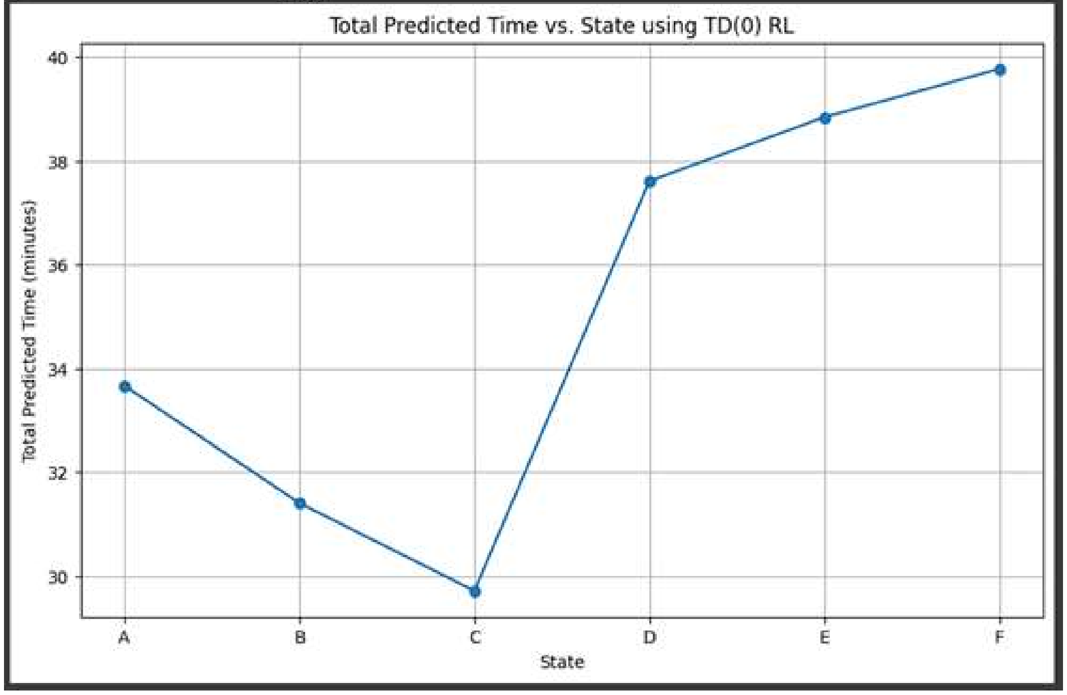
Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Define states
states = ['A', 'B', 'C', 'D', 'E', 'F']
num_states = len(states)

# Define initial elapsed times with a 5 min gap
initial_elapsed_times = [0, 5, 10, 20, 30, 40]

# Add noise to the elapsed times
np.random.seed(42)  # For reproducibility
noise_elapsed = np.random.normal(0, 1, num_states)
final_elapsed_times = np.maximum(np.array(initial_elapsed_times) +
noise_elapsed, 0)

# Define fixed predicted times from each state to the final state
predicted_times_to_go = [30, 25, 20, 15, 10, 0]

# Add noise to the predicted times
noise_predicted = np.random.normal(0, 2, num_states)
final_predicted_times_to_go = np.maximum(np.array(predicted_times_to_go) +
noise_predicted, 0)

# Calculate total predicted times
total_predicted_times = final_elapsed_times + final_predicted_times_to_go

# Round the times to two decimal places
final_elapsed_times = np.round(final_elapsed_times, 2)
final_predicted_times_to_go = np.round(final_predicted_times_to_go, 2)
total_predicted_times = np.round(total_predicted_times, 2)

# Initialize value function
V = {state: 0 for state in states}

# Parameters
alpha = 0.1  # Learning rate
gamma = 1.0  # Discount factor
num_episodes = 1000
```

```python
# TD(0) Learning
for episode in range(num_episodes):
    state_idx = 0  # Start from state 'A'
    while state_idx < num_states - 1:
        state = states[state_idx]
        next_state = states[state_idx + 1]

        reward = -total_predicted_times[state_idx]  # Negative because we minimize time
        if next_state == 'F':
            V[state] += alpha * (reward - V[state])  # Update value function for terminal state
        else:
            V[state] += alpha * (reward + gamma * V[next_state] - V[state])  # TD(0) update

        state_idx += 1

# Create a DataFrame for tabular output
data = {
    'State': states,
    'Elapsed Time (minutes)': final_elapsed_times,
    'Predicted Time to Go (minutes)': final_predicted_times_to_go,
    'Predicted Total Time (minutes)': total_predicted_times
}
df = pd.DataFrame(data)

# Print the DataFrame
print(df)

# Plot the graph
plt.figure(figsize=(10, 6))
plt.plot(states, total_predicted_times, marker='o')
plt.xlabel('State')
plt.ylabel('Total Predicted Time (minutes)')
plt.title('Total Predicted Time vs. State using TD(0) RL')
plt.grid(True)
plt.show()
```



```
  State  Elapsed Time (minutes)  Predicted Time to Go (minutes)  \
0     A                    0.50                           33.16
1     B                    4.86                           26.53
2     C                   10.65                           19.06
3     D                   21.52                           16.09
4     E                   29.77                            9.07
5     F                   39.77                            0.00

   Predicted Total Time (minutes)
0                           33.66
1                           31.40
2                           29.71
3                           37.61
4                           38.84
5                           39.77
```



Conclusion:

Successfully implement the Driving Home problem using Temporal difference learning.

Post Lab Descriptive Questions:

List advantages and disadvantages of MC and TD approach.

Ans.

Aspect	Monte Carlo (MC)	Temporal Difference (TD)
Updates	Updates occur after the entire episode is completed.	Updates occur after each time step (on-the-fly).
Exploration Requirement	Requires complete exploration of episodes.	Can learn from incomplete episodes, more practical.
Convergence	Converges slowly since it updates only after episodes.	Converges faster due to step-by-step updates.
Stability	More stable because it averages over complete returns.	Can be less stable due to dependency on immediate rewards.
Bias vs. Variance	High variance, but no bias (true return estimation).	Lower variance, but introduces bias due to bootstrapping.
Learning in Non-Episodic Tasks	Not applicable to non-episodic tasks.	Can be applied to both episodic and non-episodic tasks.
Simplicity	Simple to implement for episodic tasks.	Requires more complexity (e.g., discount factor).
Memory Requirements	Needs memory of the entire episode before updates.	Requires less memory, updating values step by step.
Efficiency	Less efficient due to episodic updates.	More efficient since updates happen during the episode.

Date: ______________ **Signature of faculty in-charge**