AI LAB EXAM

ROLL NO: U21CS052
NAME : PANCHAL GUNGUN PARESH

Que1.According to the rules of the Billy Badger Fan Club, an applicant is acceptable for membership provided that: 1. The applicant must have two proposers who are members of the club. 2. The applicant must be aged between 18 and 30 years of age (inclusive). 3. Each proposer must have been a member for at least two years. Each proposer must not be a parent of the applicant. Write a Prolog program that includes a rule for deciding whether an applicant is acceptable for membership, illustrated with a sample database. Note: create your own database that includes your own predicates like proposer, age, membership or parent

```
/* Sample database */
proposer(john).
proposer(sarah).
proposer(mike).

age(john, 25).
age(sarah, 28).
age(mike, 32). /* Mike is too old */
```

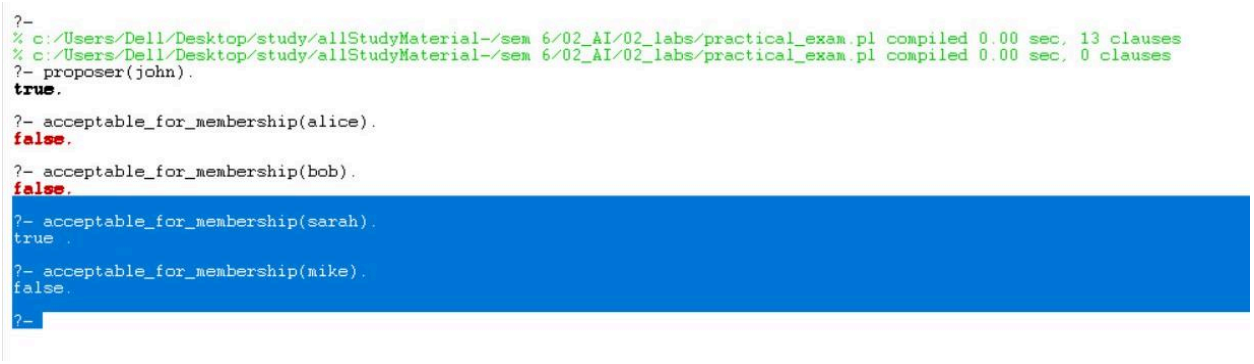
```
membership(john, 3).
membership(sarah, 2).
membership(mike, 5).

parent(john, alice).
parent(sarah, bob).
parent(mike, carol).

/* Rules */
acceptable_for_membership(Applicant) :-
    proposer(Proposer1),
    proposer(Proposer2),
    Proposer1 \= Proposer2, /* Ensures
proposers are different */
    age(Applicant, Age),
    Age >= 18,
    Age =< 30,
    membership(Proposer1, Years1),
    membership(Proposer2, Years2),
    Years1 >= 2,
    Years2 >= 2,
    not(parent(Proposer1, Applicant)),
    not(parent(Proposer2, Applicant)).

/* Example Query */
```

```
/* acceptable_for_membership(john). */
```



Que2.

Implement A* algorithm in python. Solve following problem as shown in given figure. Find shortest path between a to z.

```python
import heapq
def astar(start, goal, graph, heuristic):
    """
    A* algorithm implementation.
    Args:
        start: Start node.
        goal: Goal node.
        graph: Graph represented as a dictionary of dictionaries.
        heuristic: Heuristic function.
    Returns:
        Path from start to goal.
    """
    # Initialize open and closed lists.
    open_list = [(0, start)]
    closed_list = set()
```

```
    # Initialize g-scores and parents.
    g_scores = {start: 0}
    parents = {start: None}
    while open_list:
        # Get node with lowest f-score.
        f_score, current = heapq.heappop(open_list)
        # Check if goal node is reached.
        if current == goal:
            path = []
            while current:
                path.append(current)
                current = parents[current]
            return path[::-1]
        # Add current node to closed list.
        closed_list.add(current)
        # Explore neighbors.
        for neighbor in graph[current]:
            # Ignore neighbors in closed list.
            if neighbor in closed_list:
                continue
            # Calculate tentative g-score.
            tentative_g_score = g_scores[current] +
graph[current][neighbor]
            # Add neighbor to open list if not already in it.
            if neighbor not in [n[1] for n in open_list]:
                heapq.heappush(open_list, (tentative_g_score +
heuristic(neighbor, goal), neighbor))
            # Update neighbor's g-score if new path is better.
            elif tentative_g_score < g_scores[neighbor]:
                index = [n[1] for n in open_list].index(neighbor)
                open_list[index] = (tentative_g_score +
heuristic(neighbor, goal), neighbor)
            # Update parent and g-score.
            parents[neighbor] = current
            g_scores[neighbor] = tentative_g_score
    # No path found.
    return None
```

```
# Example graph.
graph = {
    'A': {'B': 4, 'C': 3},
    'B': {'F': 5, 'E': 12,'A':4},
    'C': {'D': 7,'E':10,'A':3},
    'D': {'E': 2},
    'E': {'Z': 5,'B':12},
    'F': {'Z': 16,'B':5},
    'Z': {'F':16}
}
# Heuristic function.
def heuristic(node, goal):
    h_scores = {
        'A': 14,
        'B': 12,
        'C': 11,
        'D': 6,
        'E': 4,
        'F': 11,
        'Z': 0
    }
    return h_scores[node]
# Find path from A to G.
path = astar('A', 'Z', graph, heuristic)
print(path)
```

```
3
13
16
12
17
['A', 'C', 'D', 'E', 'Z']
```