Artificial Intelligence Lab

LIST OF EXPERIMENTS

1. WAP in Prolog to have an introduction of Prolog fundamentals: constants, predicates, arguments, variables.

Ans. % Constants
% Constants are atomic values that do not change.
% Constants in Prolog are represented as atoms or numbers.

```
% Example atoms
constant(atom).
constant(123).
constant('This is a constant string.').

% Predicates
% Predicates are relationships or rules defined in Prolog.
% Predicates consist of a functor and arguments.

% Example predicates
happy(john).
likes(john, mary).
parent(john, alice).
parent(mary, alice).

% Rules and arguments
% Rules in Prolog define relationships or conditions.
% Arguments are placeholders for variables or constants.

% Example rule
mortal(X) :- human(X). % X is mortal if X is human

% Variables
% Variables in Prolog are denoted by an initial uppercase letter or an underscore.
% They are placeholders that can unify with constants or other variables.

% Example variable usages
human(socrates).
human(alice).
human(bob).
human(X) :- person(X). % X is human if X is a person

% Query examples
% Queries allow us to ask questions or find solutions based on defined predicates and
rules.

% Example queries
% Is John happy?
% Query: happy(john).
```

% Output: true (if there is a fact stating John is happy)

% Who does John like?
% Query: likes(john, Who).
% Output: Who = mary (if there is a fact stating John likes Mary)

% Is Alice a mortal?
% Query: mortal(alice).
% Output: true (if there is a fact or rule stating all humans are mortal)

% Who are the humans?
% Query: human(Who).
% Output: Who = socrates ; Who = alice ; Who = bob (if there are facts stating these individuals are human)

2. WAP in Prolog to have an introduction of Tests, Backtracking.

Ans. % Facts: Define relationships and properties

```
likes(john, pizza).
likes(john, sushi).
likes(mary, sushi).
likes(alice, chocolate).

female(mary).
female(alice).
male(john).

% Rules: Define logical relationships and conditions

likes_food(X, Y) :-
   likes(X, Y),
   food(Y).

food(pizza).
food(sushi).
food(chocolate).

% Queries and tests

% Test: Is John male?
% Query: male(john).
% Output: true (if the fact male(john) is defined)

% Test: Is Mary a female?
% Query: female(mary).
% Output: true (if the fact female(mary) is defined)

% Test: Does John like sushi?
% Query: likes(john, sushi).
% Output: true (if the fact likes(john, sushi) is defined)
```

```
% Test: What food does John like?
% Query: likes_food(john, Food).
% Output: Food = pizza ; Food = sushi (if John likes either pizza or sushi)

% Test: Who likes sushi?
% Query: likes(_, sushi).
% Output: john ; mary (if either John or Mary likes sushi)

% Test with backtracking: Find all people who like some food
% Query: likes(X, Food).
% Output: X = john, Food = pizza ;
%         X = john, Food = sushi ;
%         X = mary, Food = sushi ;
%         X = alice, Food = chocolate. (if these facts are defined)
```

3. WAP in Prolog to have an introduction of Recursion.

Ans. % Factorial of 0 is 1.

```
factorial(0, 1).

% Define factorial calculation using recursion
factorial(N, Result) :-
    N > 0,              % Ensure N is a positive integer
    N1 is N - 1,        % Decrease N by 1 for the next recursion
    factorial(N1, SubResult),    % Recursive call with N1
    Result is N * SubResult.     % Calculate factorial

% Query examples

% Calculate factorial of 5
% Query: factorial(5, Result).
% Output: Result = 120 (5! = 120)

% Calculate factorial of 0
% Query: factorial(0, Result).
% Output: Result = 1 (0! = 1)

% Calculate factorial of a negative number
% Query: factorial(-3, Result).
% Output: false (Factorial is not defined for negative integers)
```

4. WAP in Prolog to have an introduction of State-Space Search: DFS

Ans: % Define edges in the graph

```
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(c, g).
```

```
% Depth-First Search (DFS) implementation
dfs(Node, Visited) :-
    dfs_helper(Node, [], Visited).

% Helper predicate for DFS
dfs_helper(Node, VisitedNodes, FinalVisited) :-
    \+ member(Node, VisitedNodes), % Ensure Node hasn't been visited yet
    write(Node), nl,               % Print the current node
    dfs_neighbors(Node, VisitedNodes, NewVisited), % Explore neighbors
    append([Node], NewVisited, FinalVisited). % Update visited nodes

% Explore neighbors of a node
dfs_neighbors(Node, VisitedNodes, FinalVisited) :-
    edge(Node, NextNode),              % Find a neighbor
    dfs_helper(NextNode, VisitedNodes, FinalVisited). % Continue DFS from neighbor

% Example query to perform DFS from node 'a'
% Query: dfs(a, VisitedNodes).
% Output:
% a
% b
% d
% e
% c
% f
% g
% VisitedNodes = [g, f, c, e, d, b, a]
```

5. WAP in Prolog to have an introduction of State-Space Search: BFS

Ans: % Define edges in the graph

```
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(c, g).

% Breadth-First Search (BFS) implementation
bfs(Start, Visited) :-
    bfs([Start], [], Visited).

% Base case for BFS
bfs([], Visited, Visited).

% Main BFS predicate
bfs([CurrentNode|Rest], VisitedSoFar, Visited) :-
    \+ member(CurrentNode, VisitedSoFar), % Ensure CurrentNode not visited yet
    write(CurrentNode), nl,              % Print the current node
    findall(NextNode, edge(CurrentNode, NextNode), Neighbors), % Find neighbors
    append(Rest, Neighbors, NewQueue),     % Add neighbors to the queue
```

```
    append(VisitedSoFar, [CurrentNode], UpdatedVisited), % Update visited nodes
    bfs(NewQueue, UpdatedVisited, Visited). % Recursive call for next level

% Example query to perform BFS from node 'a'
% Query: bfs(a, VisitedNodes).
% Output:
% a
% b
% c
% d
% e
% f
% g
% VisitedNodes = [a, c, b, g, f, e, d]
```

6. Write a program to implement Nearest Neighbour classification technique

Ans: % Define training data

```
% For simplicity, assume 2D points with their respective classes

% Example training data with points and their classes
point(2, 4, 'ClassA').
point(4, 6, 'ClassA').
point(3, 3, 'ClassB').
point(6, 2, 'ClassB').

% Calculate Euclidean distance between two points
distance(X1, Y1, X2, Y2, Distance) :-
    Distance is sqrt((X2 - X1) ** 2 + (Y2 - Y1) ** 2).

% Nearest Neighbor classification
nearest_neighbor(X, Y, Class) :-
    findall(Distance-Class, (point(X1, Y1, Class), distance(X, Y, X1, Y1, Distance)),
Distances),
    keysort(Distances, Sorted), % Sort distances
    Sorted = [First|_], % Get the nearest point
    First = _-Class. % Extract the class of the nearest point

% Example queries
% Classify a point (5, 5)
% Query: nearest_neighbor(5, 5, Class).
% Output: Class = 'ClassA' (if the nearest neighbor belongs to 'ClassA')

% Classify a point (2, 2)
% Query: nearest_neighbor(2, 2, Class).
% Output: Class = 'ClassB' (if the nearest neighbor belongs to 'ClassB')
```

7. Write a program to implement Nearest Neighbour classification technique.

Ans: % Define the training data

```
% For simplicity, let's assume a few training points with their respective classes
```

```
% Example training data with points and their classes
point(2, 4, 'ClassA').
point(4, 6, 'ClassA').
point(3, 3, 'ClassB').
point(6, 2, 'ClassB').

% Calculate Euclidean distance between two points
distance(X1, Y1, X2, Y2, Distance) :-
    Distance is sqrt((X2 - X1) ** 2 + (Y2 - Y1) ** 2).

% Nearest Neighbor classification
nearest_neighbor(X, Y, Class) :-
    findall(Distance-Class, (point(X1, Y1, Class), distance(X, Y, X1, Y1, Distance)),
Distances),
    keysort(Distances, Sorted), % Sort distances
    Sorted = [First|_], % Get the nearest point
    First = _-Class. % Extract the class of the nearest point

% Example queries
% Classify a point (5, 5)
% Query: nearest_neighbor(5, 5, Class).
% Output: Class = 'ClassA' or 'ClassB' (if the nearest neighbor belongs to either 'ClassA'
or 'ClassB')

% Classify a point (2, 2)
% Query: nearest_neighbor(2, 2, Class).
% Output: Class = 'ClassA' or 'ClassB' (if the nearest neighbor belongs to either 'ClassA'
or 'ClassB')
```