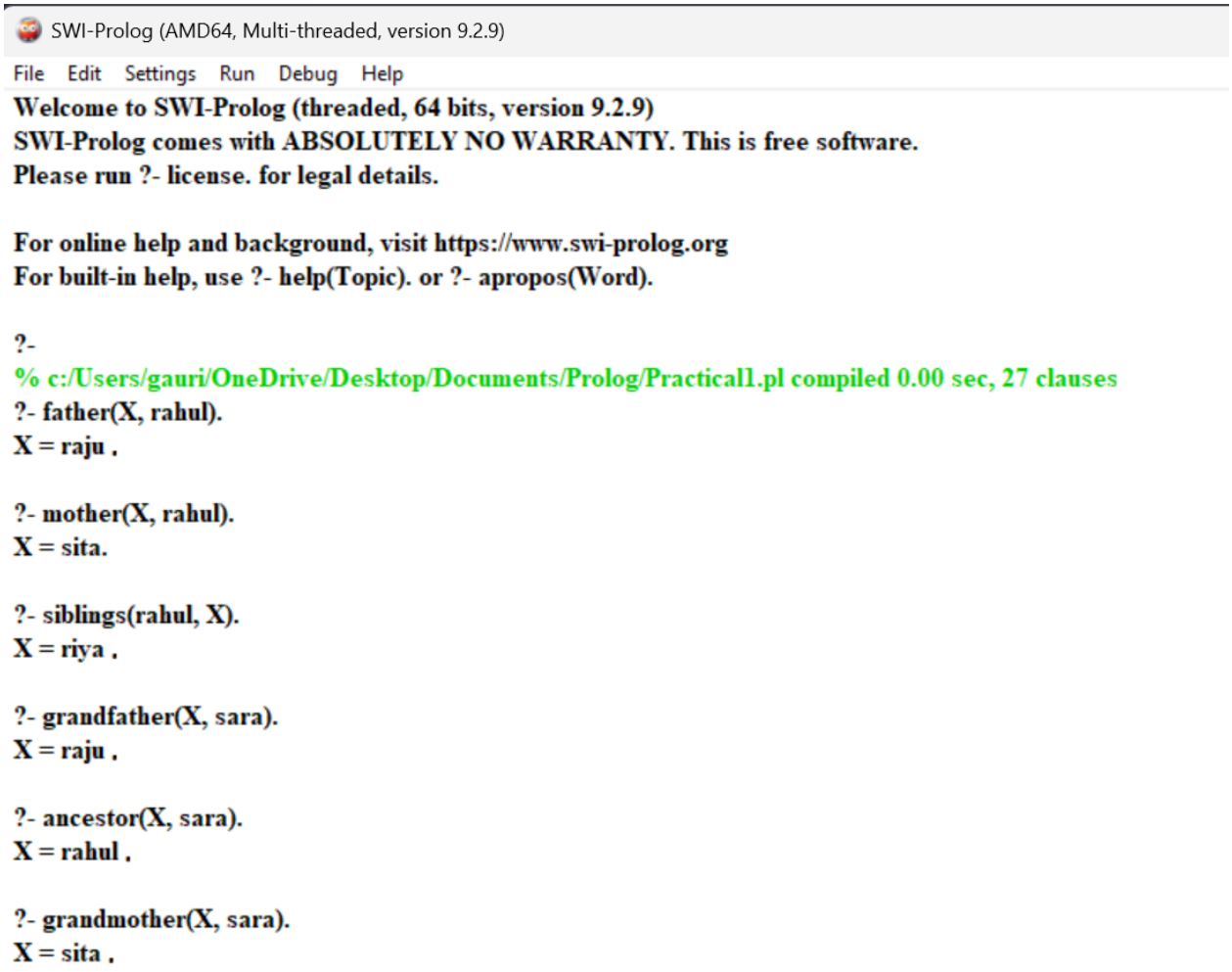
Practical No 1

Aim - Prolog Program to Construct a Family Tree

```
/* ---------- FACTS ---------- */
% gender
male(raju).
male(ram).
male(rahul).
male(aman).
male(karan).

female(sita).
female(geeta).
female(riya).
female(komal).
female(sara).

% parent(child, parent)
parent(rahul, raju).
parent(rahul, sita).
parent(riya, raju).
parent(riya, sita).
parent(aman, ram).
parent(aman, geeta).

parent(karan, ram).
parent(karan, geeta).
parent(sara, rahul).
parent(sara, komal).
```

```
/* ---------- RULES ---------- */
% father(X, Y): X is father of Y
father(F, C) :- parent(C, F), male(F).

% mother(X, Y): X is mother of Y
mother(M, C) :- parent(C, M), female(M).

% siblings(X, Y): X and Y share a parent and are not the same
siblings(X, Y) :-
    parent(X, P),
    parent(Y, P),
    X \= Y.

% grandfather(GF, C): GF is grandfather of C
grandfather(GF, C) :-
    parent(C, P),
    father(GF, P).

% grandmother(GM, C): GM is grandmother of C
grandmother(GM, C) :-
    parent(C, P),
    mother(GM, P).

% ancestor(A, D): A is ancestor of D  (recursive)
ancestor(A, D) :- parent(D, A).
ancestor(A, D) :-
    parent(D, P),
    ancestor(A, P).
```

Output :


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% c:/Users/gauri/OneDrive/Desktop/Documents/Prolog/Practical1.pl compiled 0.00 sec, 27 clauses
?- father(X, rahul).
X = raju .
?- mother(X, rahul).
X = sita.
?- siblings(rahul, X).
X = riya .
?- grandfather(X, sara).
X = raju .
?- ancestor(X, sara).
X = rahul .
?- grandmother(X, sara).
X = sita .

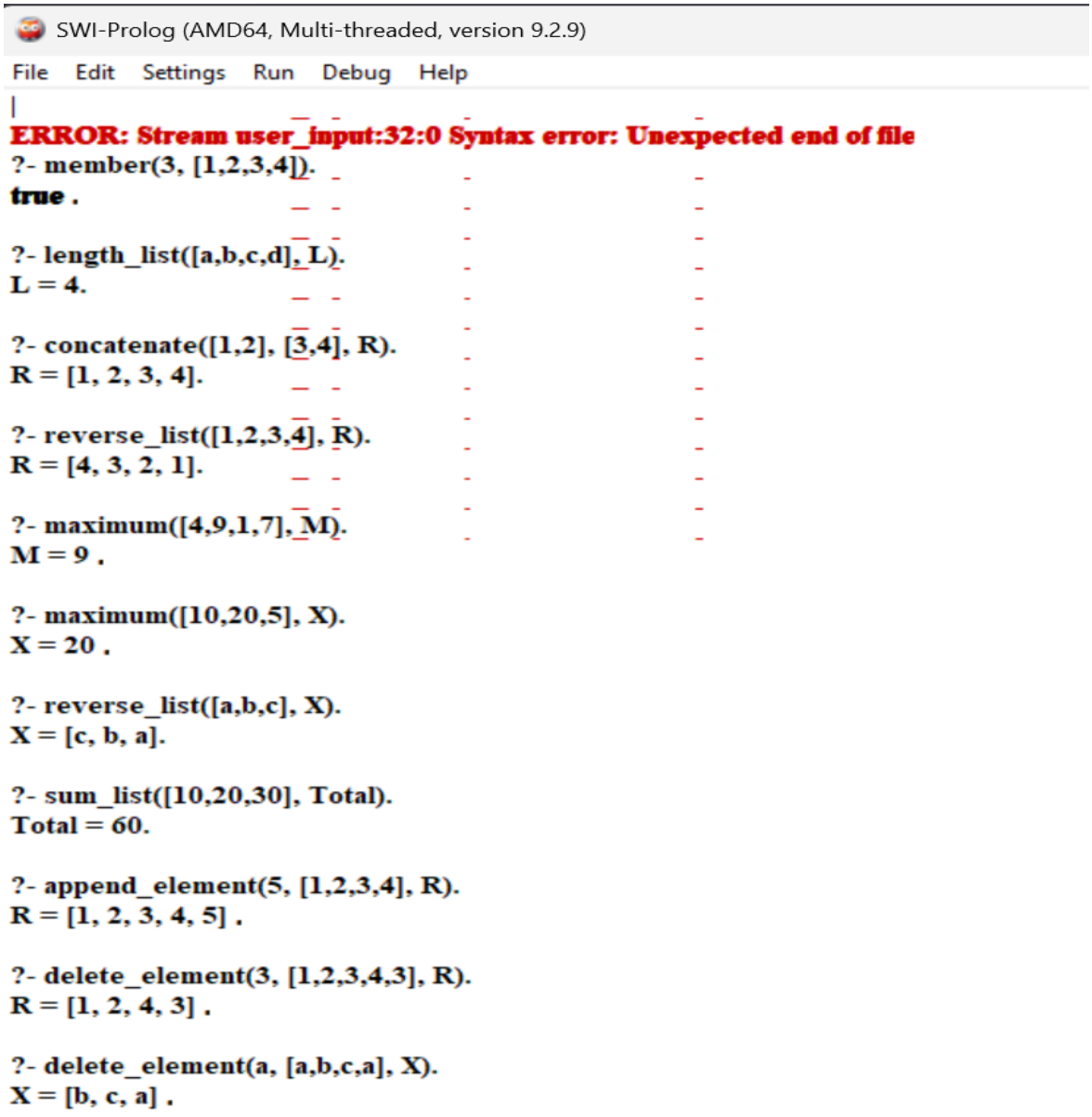
Practical No 2

Aim- Prolog Program Implementing List Operations

```
% 1. member(X, List): Check if X is present in the list
member(X, [X|_]).
member(X, [_|T]) :- member(X, T).
% 2. length_list(List, Length): Find length of list
length_list([], 0).
length_list([_|T], L) :-
    length_list(T, L1),
    L is L1 + 1.
% 3. concatenate(List1, List2, Result): Concatenate two lists
concatenate([], L, L).
concatenate([H|T], L2, [H|R]) :-
    concatenate(T, L2, R).
% 4. reverse_list(List, Reversed): Reverse a list
reverse_list([], []).
reverse_list([H|T], R) :-
    reverse_list(T, RevT),
    concatenate(RevT, [H], R).
% 5. maximum(List, Max): Find maximum element of list
maximum([X], X).
maximum([H|T], Max) :-
    maximum(T, MaxRest), Max is max(H, MaxRest).
% 6. sum_list(List, Sum): Sum of all elements
sum_list([], 0).
sum_list([H|T], S) :-
    sum_list(T, ST),
    S is H + ST.
% 7. append_element(Element, List, Result): Add element at end
append_element(X, [], [X]).
```

```
append_element(X, [H|T], [H|R]) :-
    append_element(X, T, R).
% 8. delete_element(Element, List, Result): Delete first occurrence
delete_element(X, [X|T], T).
delete_element(X, [H|T], [H|R]) :-
    delete_element(X, T, R).
```

Output :



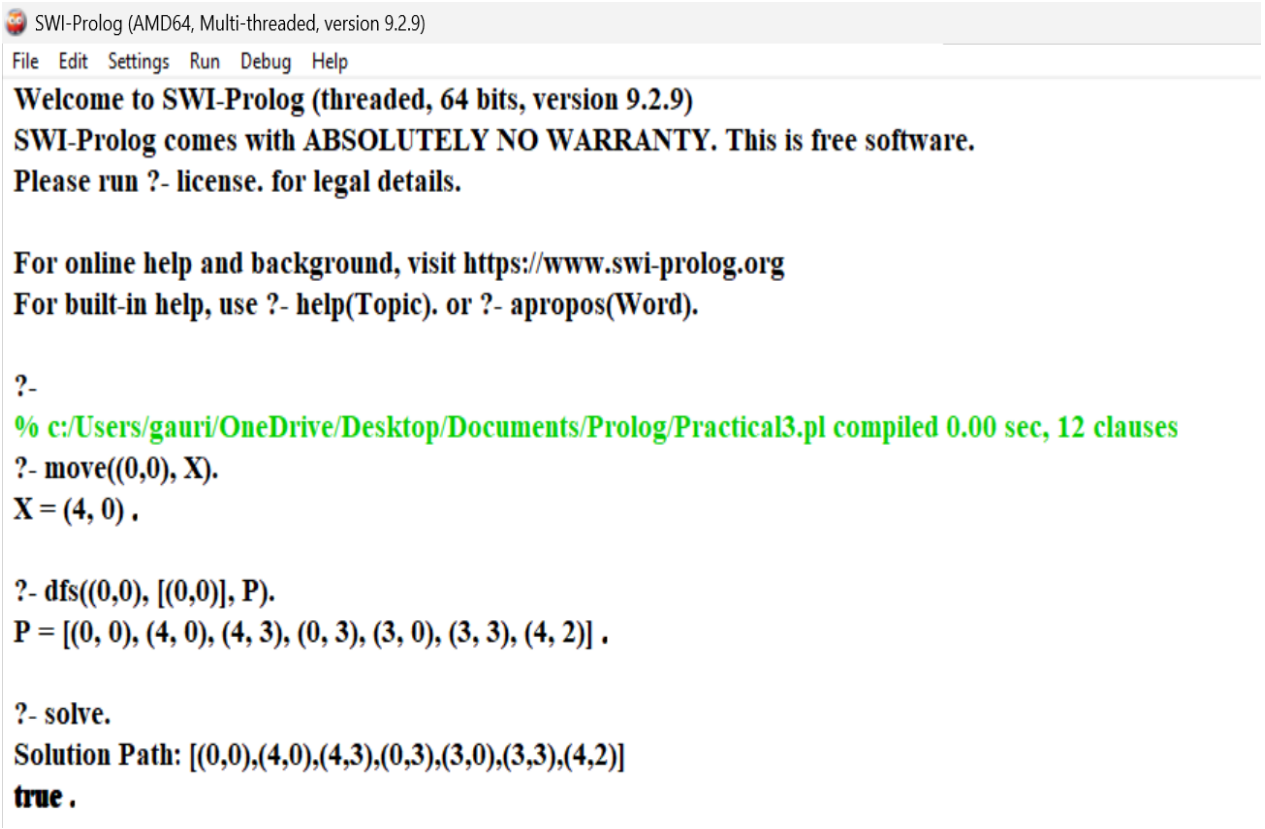
Practical No 3

Aim-Prolog Program to Solve the Water-Jug Puzzle Using DFS

```
% Initial State: (0,0)
start((0,0)).

% Goal State: any jug has 2 liters
goal((2,_)).
goal((_,2)).

% DFS Search
solve :-
    start(Start),
    dfs(Start, []).

% DFS predicate
dfs(State, _) :-
    goal(State),
    write('Goal reached: '), write(State), nl.

dfs(State, Visited) :-
    move(State, NextState),
    \+ member(NextState, Visited),   % avoid repeated states
    write('Move to: '), write(NextState), nl,
    dfs(NextState, [NextState|Visited]).

% ----------------------------------------
% Possible Moves
% ----------------------------------------

% Fill jug X (capacity 4)
```

```
move((_, Y), (4, Y)).

% Fill jug Y (capacity 3)
move((X, _), (X, 3)).

% Empty jug X
move((_, Y), (0, Y)).

% Empty jug Y
move((X, _), (X, 0)).

% Pour X → Y
move((X, Y), (X2, Y2)) :-
    Total is X + Y,
    (   Total =< 3 -> X2 = 0, Y2 = Total      % pour all X into Y
    ;   X2 is Total - 3, Y2 = 3               % fill Y completely
    ).

% Pour Y → X
move((X, Y), (X2, Y2)) :-
    Total is X + Y,
    (   Total =< 4 -> X2 = Total, Y2 = 0      % pour all Y into X
    ;   X2 = 4, Y2 is Total - 4               % fill X completely
    ).
```

Output :


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% c:/Users/gauri/OneDrive/Desktop/Documents/Prolog/Practical3.pl compiled 0.00 sec, 12 clauses
?- move((0,0), X).
X = (4, 0) .
?- dfs((0,0), [(0,0)], P).
P = [(0, 0), (4, 0), (4, 3), (0, 3), (3, 0), (3, 3), (4, 2)] .
?- solve.
Solution Path: [(0,0),(4,0),(4,3),(0,3),(3,0),(3,3),(4,2)]
true .

Practical No 4

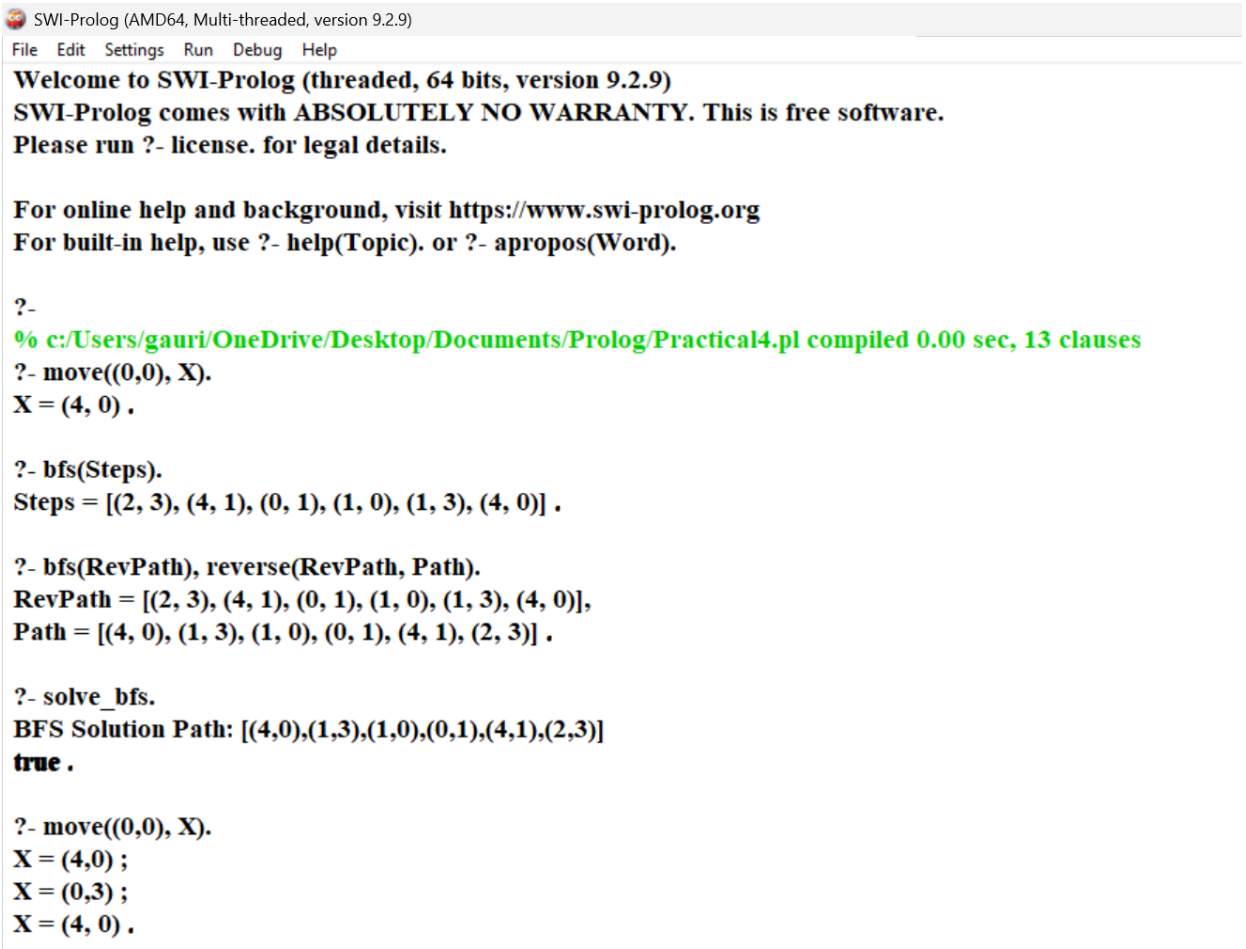
Aim- Prolog Program to Solve the Water-Jug Puzzle Using BFS

% Water Jug Problem using BFS

```
% Jug capacities: X = 4 liters, Y = 3 liters.
capacity(4, 3).

% Valid moves
move((X, Y), (4, Y)) :- capacity(4,_), X < 4.          % Fill X
move((X, Y), (X, 3)) :- capacity(_,3), Y < 3.          % Fill Y
move((X, Y), (0, Y)) :- X > 0.                          % Empty X
move((X, Y), (X, 0)) :- Y > 0.                          % Empty Y

% Pour X → Y
move((X, Y), (X2, Y2)) :-
    capacity(_, Cy),
    X > 0,
    Y < Cy,
    Transfer is min(X, Cy - Y),
    X2 is X - Transfer,
    Y2 is Y + Transfer.

% Pour Y → X
move((X, Y), (X2, Y2)) :-
    capacity(Cx,_),
    Y > 0,
    X < Cx,
    Transfer is min(Y, Cx - X),
    X2 is X + Transfer,
    Y2 is Y - Transfer.
```

```
% BFS Solution
bfs(Solution) :-
    goal(Goal),
    bfs_queue([( (0,0), [] )], [], Goal, Solution).

% Goal state: we need 2 liters in either jug
goal((2,_)).
goal((_,2)).

% BFS queue processing
bfs_queue([ (State, Path) | _ ], _, State, Path).
bfs_queue([ (State, Path) | RestQueue ], Visited, Goal, Solution) :-
    findall((Next, [Next|Path]),
        ( move(State, Next),
          \+ member(Next, Visited),
          \+ member((Next,_), RestQueue)
        ),
        NewStates),
    append(RestQueue, NewStates, UpdatedQueue),
    bfs_queue(UpdatedQueue, [State|Visited], Goal, Solution).
```

Output :


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% c:/Users/gauri/OneDrive/Desktop/Documents/Prolog/Practical4.pl compiled 0.00 sec, 13 clauses
?- move((0,0), X).
X = (4, 0) .
?- bfs(Steps).
Steps = [(2, 3), (4, 1), (0, 1), (1, 0), (1, 3), (4, 0)] .
?- bfs(RevPath), reverse(RevPath, Path).
RevPath = [(2, 3), (4, 1), (0, 1), (1, 0), (1, 3), (4, 0)],
Path = [(4, 0), (1, 3), (1, 0), (0, 1), (4, 1), (2, 3)] .
?- solve_bfs.
BFS Solution Path: [(4,0),(1,3),(1,0),(0,1),(4,1),(2,3)]
true .
?- move((0,0), X).
X = (4,0) ;
X = (0,3) ;
X = (4, 0) .

Practical No 5

Aim- Prolog Program to Solve the Towers of Hanoi Puzzle

```
/* ---------- Towers of Hanoi ---------- */
/* hanoi(N, Source, Auxiliary, Target) */
% N = number of disks
% Source = initial peg
% Auxiliary = helper peg
% Target = destination peg
hanoi(0, _, _, _) :- !.   % Base case: no disks, do nothing
hanoi(N, Source, Auxiliary, Target) :-
    N > 0,
    M is N - 1,
    hanoi(M, Source, Target, Auxiliary),          % Move N-1 disks to Auxiliary
    format('Move disk ~w from ~w to ~w~n', [N, Source, Target]),  % Move largest disk
    hanoi(M, Auxiliary, Source, Target).         % Move N-1 disks to Target
```

Output :

```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File   Edit   Settings   Run   Debug   Help

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/gauri/OneDrive/Desktop/Documents/Prolog/Practical5.pl compiled 0.00 sec, 2 clauses
?- hanoi(3, left, middle, right).
Move disk 1 from left to right
Move disk 2 from left to middle
Move disk 1 from right to middle
Move disk 3 from left to right
Move disk 1 from middle to left
Move disk 2 from middle to right
Move disk 1 from left to right
true.

?- hanoi(4, left, middle, right).
Move disk 1 from left to middle
Move disk 2 from left to right
Move disk 1 from middle to right
Move disk 3 from left to middle
Move disk 1 from right to left
Move disk 2 from right to middle
Move disk 1 from left to middle
Move disk 4 from left to right
Move disk 1 from middle to right
Move disk 2 from middle to left
Move disk 1 from right to left
Move disk 3 from middle to right
Move disk 1 from left to middle
Move disk 2 from left to right
Move disk 1 from middle to right
true.
```

Practical No. 6

Aim- Prolog Program to Solve the 8-Queens Problem

```
queens(N, Solution) :-
    range(1, N, Ns),
    place_queens(Ns, [], Solution).

/* Generate numbers from Low to High */
range(High, High, [High]) :- !.
range(Low, High, [Low|Rest]) :-
    Low < High,
    L1 is Low + 1,
    range(L1, High, Rest).

/* Place queens one by one */
place_queens([], Solution, Solution).
place_queens(Available, Partial, Solution) :-
    select(Row, Available, Remaining),
    safe(Row, Partial, 1),
    place_queens(Remaining, [Row|Partial], Solution).

/* Check if queen placement is safe */
safe(_, [], _) :- !.
safe(Row, [R|Rest], Dist) :-
    Row =\= R,              % not same row
    abs(Row - R) =\= Dist,    % not on diagonal
    D1 is Dist + 1,
    safe(Row, Rest, D1).

solve_queens(N) :-
    queens(N, Solution),
    write('One solution: '), write(Solution), nl
```

Output :

SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)

File Edit Settings Run Debug Help

```
?-
% c:/Users/gauri/OneDrive/Desktop/Documents/Prolog/Practical6.pl compiled 0.00 sec, 8 clauses
?- queens(8, Solution).
Solution = [4, 2, 7, 3, 6, 8, 5, 1] ;
Solution = [5, 2, 4, 7, 3, 8, 6, 1] ;
Solution = [3, 5, 2, 8, 6, 4, 7, 1] ;
Solution = [3, 6, 4, 2, 8, 5, 7, 1] ;
Solution = [5, 7, 1, 3, 8, 6, 4, 2] ;
Solution = [4, 6, 8, 3, 1, 7, 5, 2] ;
Solution = [3, 6, 8, 1, 4, 7, 5, 2] ;
Solution = [5, 3, 8, 4, 7, 1, 6, 2] ;
Solution = [5, 7, 4, 1, 3, 8, 6, 2] .

?- solve_queens(8).
One solution: [4,2,7,3,6,8,5,1]
true .

?- solve_queens(4).
One solution: [3,1,4,2]
true .

?- findall(S, queens(8, S), AllSolutions).
AllSolutions = [[4, 2, 7, 3, 6, 8, 5, 1], [5, 2, 4, 7, 3, 8, 6|...], [3, 5, 2, 8, 6, 4|...], [3, 6, 4, 2, 8|...], [5, 7, 1, 3|...], [4, 6, 8|...], [3, 6|..
.], [5|...], [...|...]|...].
```

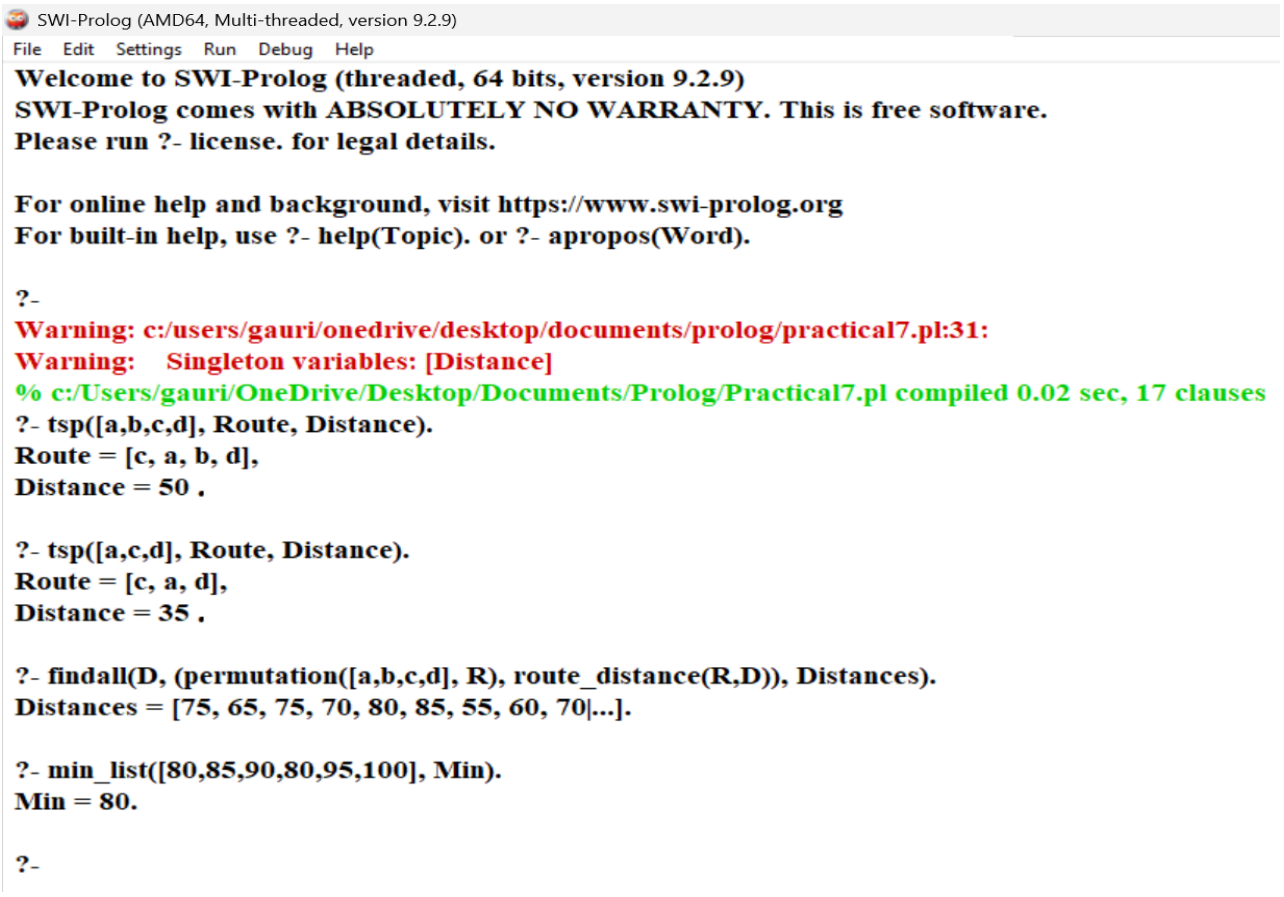
Practical No. 7

Aim- Prolog Program to Solve the Traveling Salesman Problem (TSP)

```
/* ---------- Traveling Salesman Problem (TSP) ---------- */
/* distance(City1, City2, Distance) */
distance(a, b, 10).
distance(a, c, 15).
distance(a, d, 20).
distance(b, a, 10).
distance(b, c, 35).
distance(b, d, 25).
distance(c, a, 15).
distance(c, b, 35).
distance(c, d, 30).
distance(d, a, 20).
distance(d, b, 25).
distance(d, c, 30).

/* compute distance between consecutive cities in a route */
route_distance([_], 0).
route_distance([City1, City2|Rest], Distance) :-
    distance(City1, City2, D),
    route_distance([City2|Rest], DR),
    Distance is D + DR.

/* generate all permutations of cities */
permutation([], []).
permutation(List, [H|Perm]) :-
    select(H, List, Rest),
    permutation(Rest, Perm).
```

```
/* solve TSP */
tsp(Cities, ShortestRoute, MinDistance) :-
    permutation(Cities, Route),
    route_distance(Route, Distance),
    findall(D, (permutation(Cities, R), route_distance(R, D)), Distances),
    min_list(Distances, MinDistance),
    route_distance(Route, MinDistance),
    ShortestRoute = Route.
```

Output :



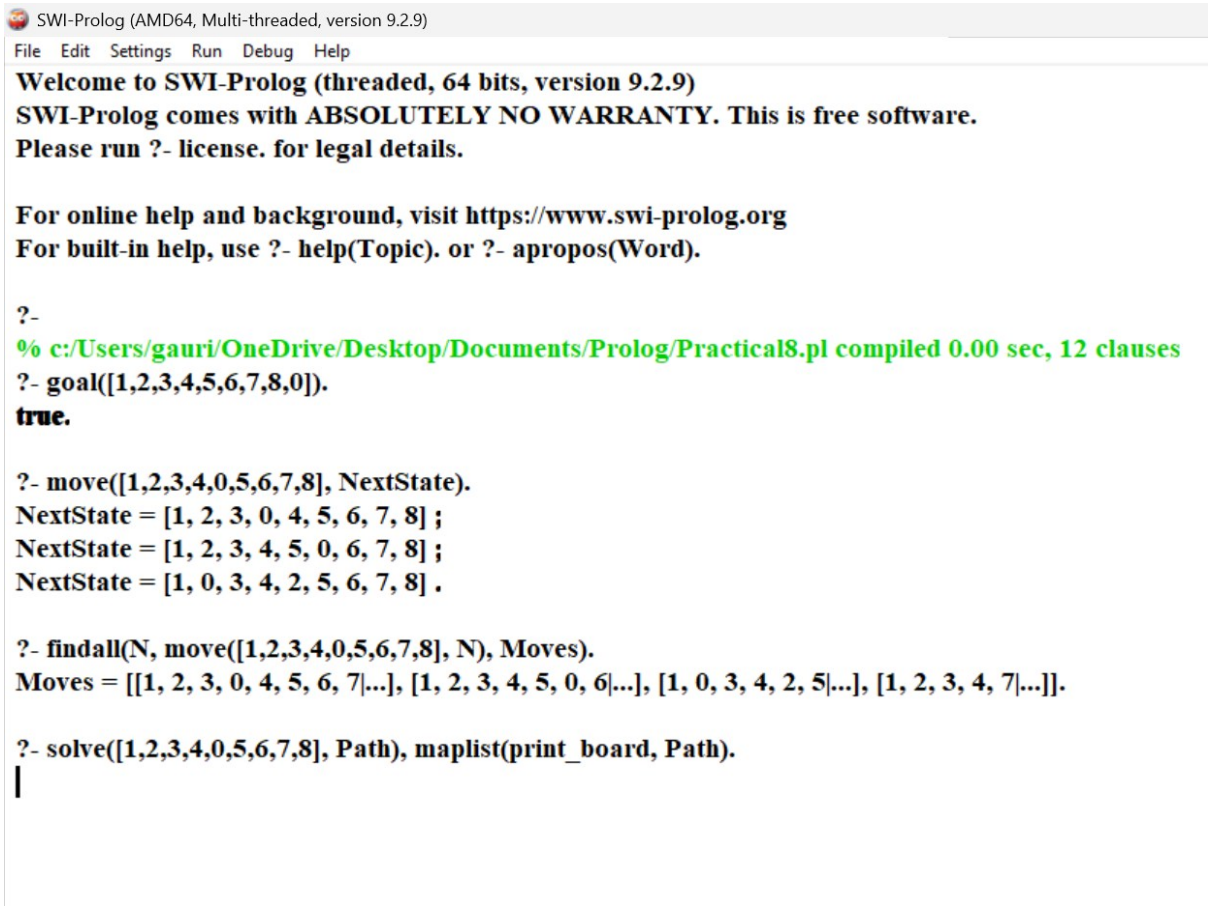
Practical No-8

Aim-Prolog Program to Implement the 8-Puzzle Game

```
/* ---------- 8-Puzzle Problem (DFS) ---------- */
/* Goal state */
goal([1,2,3,4,5,6,7,8,0]).
/* Move blank (0) left */
move(State, NextState) :-
    nth0(Pos, State, 0),
    Pos \= 0, Pos \= 3, Pos \= 6,
    LeftPos is Pos - 1,
    swap(State, Pos, LeftPos, NextState).

/* Move blank (0) right */
move(State, NextState) :-
    nth0(Pos, State, 0),
    Pos \= 2, Pos \= 5, Pos \= 8,
    RightPos is Pos + 1,
    swap(State, Pos, RightPos, NextState).

/* Move blank (0) up */
move(State, NextState) :-
    nth0(Pos, State, 0),
    Pos > 2,
    UpPos is Pos - 3,
    swap(State, Pos, UpPos, NextState).

/* Move blank (0) down */
move(State, NextState) :-
    nth0(Pos, State, 0),
    Pos < 6,
```

```
    DownPos is Pos + 3,
    swap(State, Pos, DownPos, NextState).

/* Swap two positions in a list */
swap(List, I, J, Swapped) :-
    nth0(I, List, ElemI),
    nth0(J, List, ElemJ),
    set_nth(List, I, ElemJ, Temp),
    set_nth(Temp, J, ElemI, Swapped).

/* Set the N-th element of a list */
set_nth([_|T], 0, Elem, [Elem|T]).
set_nth([H|T], N, Elem, [H|R]) :-
    N > 0, N1 is N - 1,
    set_nth(T, N1, Elem, R).

/* DFS search */
dfs(State, _, [State]) :- goal(State), !.
dfs(State, Visited, [State|Path]) :-
    move(State, Next),
    \+ member(Next, Visited),
    dfs(Next, [Next|Visited], Path).

/* Solve puzzle from initial state */
solve(InitialState, Path) :-
    dfs(InitialState, [InitialState], Path).

/* Print 3x3 board */
print_board([A,B,C,D,E,F,G,H,I]) :-
    format('~w ~w ~w~n', [A,B,C]),
    format('~w ~w ~w~n', [D,E,F]),
    format('~w ~w ~w~n~n', [G,H,I]).
```

Output :


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% c:/Users/gauri/OneDrive/Desktop/Documents/Prolog/Practical8.pl compiled 0.00 sec, 12 clauses
?- goal([1,2,3,4,5,6,7,8,0]).
true.
?- move([1,2,3,4,0,5,6,7,8], NextState).
NextState = [1, 2, 3, 0, 4, 5, 6, 7, 8] ;
NextState = [1, 2, 3, 4, 5, 0, 6, 7, 8] ;
NextState = [1, 0, 3, 4, 2, 5, 6, 7, 8] .
?- findall(N, move([1,2,3,4,0,5,6,7,8], N), Moves).
Moves = [[1, 2, 3, 0, 4, 5, 6, 7|...], [1, 2, 3, 4, 5, 0, 6|...], [1, 0, 3, 4, 2, 5|...], [1, 2, 3, 4, 7|...]].
?- solve([1,2,3,4,0,5,6,7,8], Path), maplist(print_board, Path).