Batch: IAI-2 **Experiment Number: 6**

Roll Number: 16010422234 **Name: Chandana Ramesh Galgali**

Aim of the Experiment: Write a program for implementation of Prolog program on 8-Puzzle.

Program/Steps:

To solve an 8-puzzle you need to call the predicate solvepuzzle with the following parameters

- First parameter is the initial state.
- Second parameter is the goal state.
- Third parameter is a variable called Cost that returns the total cost to reach the desired configuration.

Example
solvepuzzle([[1,3,4],[8,0,5],[7,2,6]], [[1,2,3],[8,0,4],[7,6,5]],Cost).
If the puzzle is solvable it returns the total cost with each step to solve it.
If the puzzle is not solvable it returns "No soution".

Code:

```
can_it_move_left(Left):-
Left >= 0,
Left \= 2,
Left \= 5.

can_it_move_right(Right):-
8 >= Right,
Right \= 3,
Right \= 6.

can_it_move_down(Down):-
Down < 9.

can_it_move_up(Up):-
Up > 0.

countInversions(_,[],Inversions):-
    Inversions is 0.

countInversions(Number,[Head|Tail],Inversions):-
```

```
        Number>Head,
        Count is 1,
        countInversions(Number,Tail,Aux_inversions),
        Inversions is Count+Aux_inversions.

countInversions(Number,[Head|Tail],Inversions):-
        Number<Head,
        Count is 0,
        countInversions(Number,Tail,Aux_inversions),
        Inversions is Count+Aux_inversions.

issolvable([],A):-
        A is 0.

issolvable([Head|Tail],Inversions):-
        countInversions(Head,Tail,Aux_inversions),
        issolvable(Tail,Next_inversions),
        Inversions is Next_inversions+Aux_inversions.

iseven(Number):-
        0 is mod(Number,2).

solvepuzzle(Initial_state,Goal_state,Result):-
        flatten(Initial_state, List_initial_state),
        delete(List_initial_state, 0, X),
        issolvable(X,Inversions),
        0 is mod(Inversions,2),
        flatten(Goal_state, List_goal_state),
        delete(List_goal_state, 0, Y),
        issolvable(Y,Inversions_two),
        0 is mod(Inversions_two,2),
        empty_heap(Inital_heap),
        Explored_set = [List_initial_state],

astar([List_initial_state,0],List_goal_state,Goal_state,Inital_heap,Explored_set,Iterations),
        copy_term(Iterations, Result),
        !.

solvepuzzle(Initial_state,Goal_state,Result):-
        flatten(Initial_state, List_initial_state),
```

```
        delete(List_initial_state, 0, X),
        issolvable(X,Inversions),
        \+0 is mod(Inversions,2),
        flatten(Goal_state, List_goal_state),
        delete(List_goal_state, 0, Y),
        issolvable(Y,Inversions_two),
        \+0 is mod(Inversions_two,2),
        empty_heap(Inital_heap),
        Explored_set = [List_initial_state],

astar([List_initial_state,0],List_goal_state,Goal_state,Inital_heap,Explored_set,Iterations),
        copy_term(Iterations, Result),
        !.

solvepuzzle(_,_,Result):-
        Result = 'No solution'.

create_explored_set(Old_Set,Element,X):-
        Aux = [Element],
        append(Old_Set,Aux,X).

divide_list([Head|_],Head).

print_element([],_).

print_element([Head|Tail],I):-
        0 is mod(I,3),
        Newi=I+1,
        nl,
        print(Head),
        print_element(Tail,Newi).

print_element([Head|Tail],I):-
        Newi=I+1,
        print(Head),
        print_element(Tail,Newi).

print_list([],_).

print_list([Head|Tail],I):-
```

```
        number(Head),
        print_list(Tail,I).

print_list([Head|Tail],I):-
        Newi=I+1,
        print_list(Tail,Newi),
        print_element(Head,0),
        nl.

create_list_with_new_cost([],_,_,_).

create_list_with_new_cost([Head|Tail],Iterator,Pos_cost,[New_cost|Tail]):-
        Iterator == Pos_cost,
        New_iterator is Iterator+1,
        New_cost is Head + 1,
        create_list_with_new_cost(Tail,New_iterator,Pos_cost,Tail).

create_list_with_new_cost([Head|Tail],Iterator,Pos_cost,[Head|Tail2]):-
        New_iterator is Iterator+1,
        create_list_with_new_cost(Tail,New_iterator,Pos_cost,Tail2).

astar([Head|Tail],Head,_,_,_,Result):-
        append([Head],Tail,Fathers),
        print_list(Fathers,0),
        length(Tail,Aux),
        Result is Aux-1.

astar(State,Goal_state,Grid_goal_state,Priority_queue,Explored_set,Result):-
        divide_list(State,State_to_esplore),
        nth0(Position_blank_tile,State_to_esplore, 0),
        length(State,Pos_cost),
        nth1(Pos_cost, State, Cost),
        New_cost is Cost + 1,
        create_list_with_new_cost(State,1,Pos_cost,New_state),

findcombinations(New_state,Grid_goal_state,Position_blank_tile,0,Priority_queue,New_cost,Ex
plored_set,New_priority_queue),
        get_from_heap(New_priority_queue, _, P, Next_priority_queue),
        divide_list(P,Explored),
        create_explored_set(Explored_set,Explored,New_explored_set),
```

```
    astar(P,Goal_state,Grid_goal_state,Next_priority_queue,New_explored_set,Result).

astar(_,_,_,Priority_queue,_,Result):-
    empty_heap(Priority_queue),
    Result = 'No solution'.

findcost([],_,_,Nextcost):-
    Nextcost is 0.

findcost([Head|Tail],Matrixinitialstate ,Matrixgoalstate, Cost):-
    Head == 0,
    findcost(Tail,Matrixinitialstate ,Matrixgoalstate, Nextcost),
    Cost is 0 + Nextcost.

findcost([Head|Tail], Matrixinitialstate ,Matrixgoalstate, Cost):-
    matrix(Matrixgoalstate,K,L,Head),
    matrix(Matrixinitialstate,I,J,Head),
    Manhattan_distance is abs(I-K) + abs(J-L),
    findcost(Tail,Matrixinitialstate,Matrixgoalstate,Nextcost),
    Cost is Manhattan_distance + Nextcost.

convert_to_matrix(Lista,Nueva_lista):-
    aux_convert_to_matrix(Lista,1,N1,T1),
    aux_convert_to_matrix(T1,1,N2,T2),
    aux_convert_to_matrix(T2,1,N3,_),
    append([N1],[N2],Aux),
    append(Aux,[N3],Nueva_lista),
    !.

aux_convert_to_matrix([Head|Tail], Iterator, [Head|Tail2], Sobra):-
    Iterator < 3,
    Nuevoi is Iterator+1,
    aux_convert_to_matrix(Tail,Nuevoi,Tail2, Sobra).

aux_convert_to_matrix([Head|Tail], Iterator, [Head], Tail):-
    0 is mod(Iterator,3).

create_list_of_explored_states(List,Element,New_list):-
    Aux = [Element],
    append(Aux,List,New_list).
```

```
findcombinations(State,Matrix_goal_state,Position_blank_tile,0,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue):-
        divide_list(State,State_to_esplore),
        Left is Position_blank_tile - 1,
        can_it_move_left(Left),
        swap_tiles(State_to_esplore, Position_blank_tile, Left, Permutation_left),
        \+member(Permutation_left,Explored_set),
        convert_to_matrix(Permutation_left,Matrix_per_left),
        findcost(Permutation_left,Matrix_per_left,Matrix_goal_state,Cost),
        create_list_of_explored_states(State,Permutation_left,State_with_fathers),
        New_cost is Cost_move_grid + Cost,
        add_to_heap(Old_priority_queue,New_cost,State_with_fathers,Aux_priority_queue),

findcombinations(State,Matrix_goal_state,Position_blank_tile,1,Aux_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue).

findcombinations(State,Matrix_goal_state,Position_blank_tile,0,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue):-

findcombinations(State,Matrix_goal_state,Position_blank_tile,1,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue).

findcombinations(State,Matrix_goal_state,Position_blank_tile,1,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue):-
        divide_list(State,State_to_esplore),
        Right is Position_blank_tile + 1,
        can_it_move_right(Right),
        swap_tiles(State_to_esplore, Position_blank_tile, Right, Permutation_right),
        \+member(Permutation_right,Explored_set),
        convert_to_matrix(Permutation_right,Matrix_per_right),
        findcost(Permutation_right,Matrix_per_right,Matrix_goal_state,Cost),
        create_list_of_explored_states(State,Permutation_right,State_with_fathers),
        New_cost is Cost_move_grid + Cost,
        add_to_heap(Old_priority_queue,New_cost,State_with_fathers,Aux_priority_queue),

findcombinations(State,Matrix_goal_state,Position_blank_tile,2,Aux_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue).

findcombinations(State,Matrix_goal_state,Position_blank_tile,1,Old_priority_queue,Cost_move
```

```
_grid,Explored_set,New_priority_queue):-

findcombinations(State,Matrix_goal_state,Position_blank_tile,2,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue).

findcombinations(State,Matrix_goal_state,Position_blank_tile,2,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue):-
        divide_list(State,State_to_esplore),
        Down is Position_blank_tile + 3,
        can_it_move_down(Down),
        swap_tiles(State_to_esplore, Position_blank_tile, Down, Permutation_down),
        \+member(Permutation_down,Explored_set),
        convert_to_matrix(Permutation_down,Matrix_per_down),
        findcost(Permutation_down,Matrix_per_down,Matrix_goal_state,Cost),
        create_list_of_explored_states(State,Permutation_down,State_with_fathers),
        New_cost is Cost_move_grid + Cost,
        add_to_heap(Old_priority_queue,New_cost,State_with_fathers,Aux_priority_queue),

findcombinations(State,Matrix_goal_state,Position_blank_tile,3,Aux_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue).

findcombinations(State,Matrix_goal_state,Position_blank_tile,2,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue):-

findcombinations(State,Matrix_goal_state,Position_blank_tile,3,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue).

findcombinations(State,Matrix_goal_state,Position_blank_tile,3,Old_priority_queue,Cost_move
_grid,Explored_set,New_priority_queue):-
        divide_list(State,State_to_esplore),
        Up is Position_blank_tile -3,
        can_it_move_up(Up),
        swap_tiles(State_to_esplore, Position_blank_tile, Up, Permutation_up),
        \+member(Permutation_up,Explored_set),
        convert_to_matrix(Permutation_up,Matrix_per_up),
        findcost(Permutation_up,Matrix_per_up,Matrix_goal_state,Cost),
        create_list_of_explored_states(State,Permutation_up,State_with_fathers),
        New_cost is Cost_move_grid + Cost,
        add_to_heap(Old_priority_queue,New_cost,State_with_fathers,Aux_priority_queue),
```

```
findcombinations(State,Matrix_goal_state,Position_blank_tile,4,Aux_priority_queue,Cost_move_grid,Explored_set,New_priority_queue).

findcombinations(State,Matrix_goal_state,Position_blank_tile,3,Old_priority_queue,Cost_move_grid,Explored_set,New_priority_queue):-

findcombinations(State,Matrix_goal_state,Position_blank_tile,4,Old_priority_queue,Cost_move_grid,Explored_set,New_priority_queue).

findcombinations(_,_,_,4,Old_priority_queue,_,_,New_priority_queue):-
        copy_term(Old_priority_queue,New_priority_queue).

matrix(M, X, Y, Element) :-
    nth0(X, M, R),
    nth0(Y, R, Element).

swap_tiles(List,Zero,Move,Nl):-
        Ayuda is Move+1,
        Zero==Ayuda,
        nth0(Move,List, Number_to_find),
        aux_swap_tiles(List,Move,0,New_list,_,List_to_explore_more),
        append(New_list,[0],Nl_aux),
        append(Nl_aux,[Number_to_find],Nl_aux2),
        delete(List_to_explore_more, 0, X),
        append(Nl_aux2,X,Nl),
        !.

swap_tiles(List,Zero,Move,Nl):-
        Ayuda is Move-1,
        Zero==Ayuda,
        nth0(Move,List,Number_to_find),
        aux_swap_tiles(List,Zero,0,New_list,_,List_to_explore_more),
        append(New_list,[Number_to_find],Nl_aux),
        append(Nl_aux,[0],Nl_aux2),
        delete(List_to_explore_more, Number_to_find, X),
        append(Nl_aux2,X,Nl),
        !.

swap_tiles(List,Zero,Move,Nl):-
        Ayuda is Move+1,
```

```
        Ayuda_dos is Move-1,
        Zero<Move,
        \+Zero==Ayuda,
        \+Zero==Ayuda_dos,
        nth0(Move,List, Number_to_find),
        aux_swap_tiles(List,Zero,0,New_list,Current_iterator,List_to_explore_more),
        append(New_list,[Number_to_find],Nl_aux),

aux_swap_tiles(List_to_explore_more,Move,Current_iterator+1,New_list_two,_,List_to_explore
_more_two),
        append(Nl_aux,New_list_two,Nl_aux_two),
        append(Nl_aux_two,[0],Nl_aux_three),
        append(Nl_aux_three,List_to_explore_more_two,Nl),
        !.

swap_tiles(List,Zero,Move,Nl):-
        Ayuda is Move+1,
        Ayuda_dos is Move-1,
        Zero>Move,
        \+Zero==Ayuda,
        \+Zero==Ayuda_dos,
        nth0(Move,List, Number_to_find),
        aux_swap_tiles(List,Move,0,New_list,Current_iterator,List_to_explore_more),
        append(New_list,[0],Nl_aux),

aux_swap_tiles(List_to_explore_more,Zero,Current_iterator+1,New_list_two,_,List_to_explore_
more_two),
        append(Nl_aux,New_list_two,Nl_aux_two),
        append(Nl_aux_two,[Number_to_find],Nl_aux_three),
        append(Nl_aux_three,List_to_explore_more_two,Nl),
        !.

aux_swap_tiles([_|Tail],Limit,Iterator,[],X,Tail):-
        Iterator==Limit,
        copy_term(Iterator,X).

aux_swap_tiles([Head|Tail],Limit,Iterator,[Head|Tail2],X,List_to_explore_more):-
        Iterator<Limit,
        New_iterator is Iterator+1,
        aux_swap_tiles(Tail,Limit,New_iterator,Tail2,X,List_to_explore_more).
```

Output/Result:

```
solvepuzzle([[1,2,3],[4,0,5],[7,8,6]], [[1,2,3],[4,5,6],[8,7,0]],Cost).
Cost = 'No solution'
```

```
?- solvepuzzle([[1,2,3],[4,0,5],[7,8,6]], [[1,2,3],[4,5,6],[8,7,0]],Cost).
```

Examples History Solutions table results Run!

```
solvepuzzle([[1,2,3],[4,0,5],[7,8,6]], [[1,2,3],[4,5,6],[7,8,0]],Cost).

123
405
786

123
450
786

123
456
780
Cost = 2
```

```
?- solvepuzzle([[1,2,3],[4,0,5],[7,8,6]], [[1,2,3],[4,5,6],[7,8,0]],Cost).
```

Examples History Solutions table results Run!

Outcomes: Ability to formally state the problem and develop the appropriate proof for a given logical deduction problem.

Conclusion (Based on the Results and outcomes achieved):

The experiment on implementing an 8-Puzzle solver with Prolog demonstrated Prolog's efficacy in complex problem-solving, leveraging its backtracking and state space search capabilities. This implementation highlighted Prolog's potential in logic programming and AI applications, emphasizing its strength in managing state transitions and achieving goals logically. The success of this project not only showcased Prolog's adaptability to challenging puzzles but also underscored its broader applicability in developing intelligent systems, marking a promising avenue for future AI and computational logic explorations.

References:

1. Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Second Edition, Pearson Publication
2. Luger, George F. Artificial Intelligence : Structures and strategies for complex problem solving, 2009, 6th Edition, Pearson Education
3. Ivan Bratko, Prolog Programming for AI, 2011, 4th Edition, Pearson Publication

---