**AI(2180703)**

Tutorial-7

Name : Yogesh Bavishi

Enrollment No.: 170200107003

Division/Batch: E/E1

Q: Write a program to solve 8 puzzle problem using Prolog.

**Code(pract7.pl):**

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,Explored\_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).

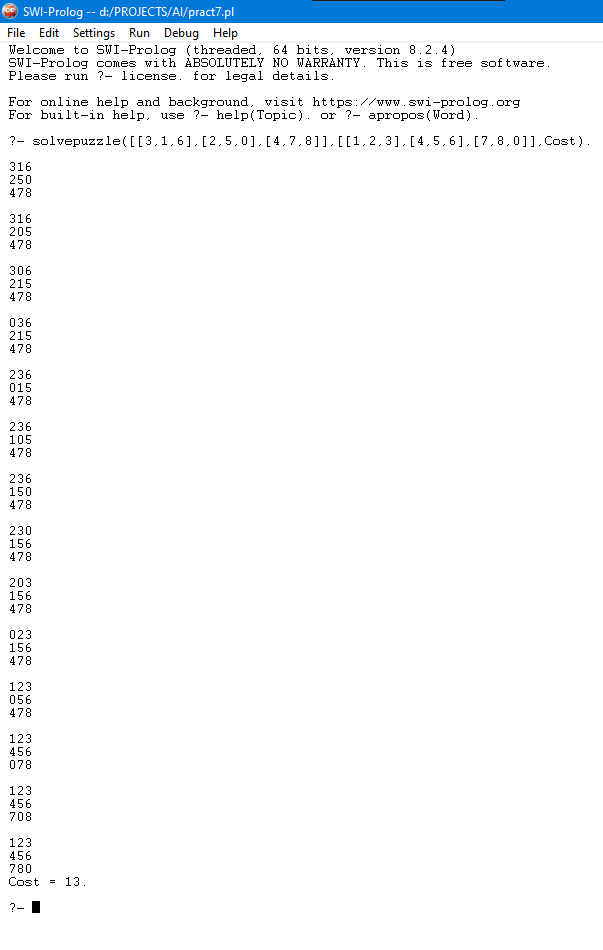
aux\_swap\_tiles([Head|Tail],Limit,Iterator,[Head|Tail2],X,List\_to\_explore\_more):-

    Iterator<Limit,

    New\_iterator is Iterator+1,

    aux\_swap\_tiles(Tail,Limit,New\_iterator,Tail2,X,List\_to\_explore\_more).

**Output :**

****