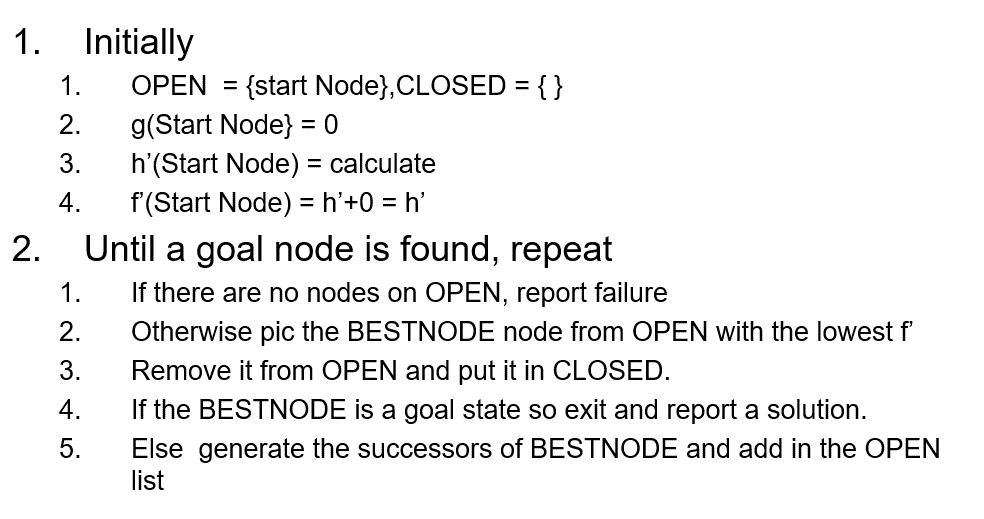
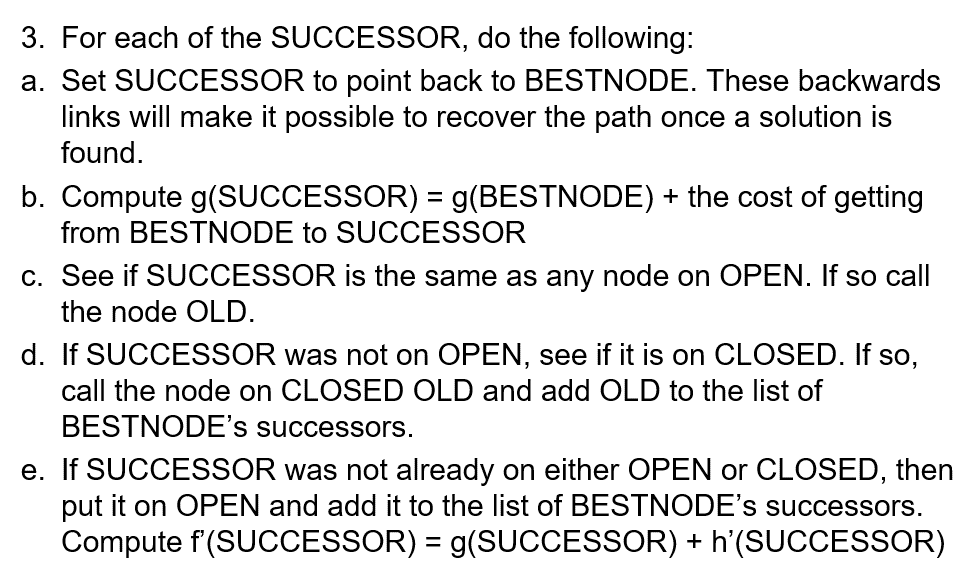
Artificial Intelligence (CS308)

Lab Test

**U19CS012**

Q.) Implement **8 Puzzle** problem with **Heuristic** **Algorithms** in PROLOG.





**Code**

*% problem specific part*

*%  Edit this as Per the Your Requirement*

*% test:-go([1,2,3,4,0,5,7,8,6],[1,2,3,4,5,6,7,8,0]).*

*% test:-go([1,2,3,0,8,5,4,7,6],[1,2,3,4,5,6,7,8,0]).*

test*:-*go([2,3,0,1,8,5,4,7,6],[1,2,3,4,5,6,7,8,0])*.*

*% move blank cell right*

*% S is current state*

*% Snew is next state*

*% the same is in left, up, down.*

move(S,Snew)*:-*

    right(S,Snew)*.*

*% move blank cell right*

*% first parameter is current state*

*% Snew is next state*

right([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*:-*

    R3>0,

    R6>0,

    R9>0,

    blank\_right([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*.*

*% move blank cell right*

*% first parameter is current state*

*% Snew is next state*

blank\_right([R1,R2,R3,R4,R5,R6,R7,R8,R9],S)*:-*

    nth0(N,[R1,R2,R3,R4,R5,R6,R7,R8,R9],0),

    Z is N+1,

    nth0(Z,[R1,R2,R3,R4,R5,R6,R7,R8,R9],R),

    substitute(R,[R1,R2,R3,R4,R5,R6,R7,R8,R9],10,Q),

    substitute(0,Q,R,V),

    substitute(10,V,0,S)*.*

move(S,Snew)*:-*

    left(S,Snew)*.*

left([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*:-*

    R1>0,

    R4>0,

    R7>0,

    blank\_left([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*.*

blank\_left([R1,R2,R3,R4,R5,R6,R7,R8,R9],S)*:-*

    nth0(N,[R1,R2,R3,R4,R5,R6,R7,R8,R9],0),

    Z is N-1,

    nth0(Z,[R1,R2,R3,R4,R5,R6,R7,R8,R9],R),

    substitute(R,[R1,R2,R3,R4,R5,R6,R7,R8,R9],10,Q),

    substitute(0,Q,R,V),

    substitute(10,V,0,S)*.*

move(S,Snew)*:-*

    down(S,Snew)*.*

down([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*:-*

    R7>0,

    R8>0,

    R9>0,

    blank\_down([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*.*

blank\_down([R1,R2,R3,R4,R5,R6,R7,R8,R9],S)*:-*

    nth0(N,[R1,R2,R3,R4,R5,R6,R7,R8,R9],0),

    Z is N+3,

    nth0(Z,[R1,R2,R3,R4,R5,R6,R7,R8,R9],R),

    substitute(R,[R1,R2,R3,R4,R5,R6,R7,R8,R9],10,Q),

    substitute(0,Q,R,V),

    substitute(10,V,0,S)*.*

move(S,Snew)*:-*

    up(S,Snew)*.*

up([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*:-*

    R1>0,

    R2>0,

    R3>0,

    blank\_up([R1,R2,R3,R4,R5,R6,R7,R8,R9],Snew)*.*

blank\_up([R1,R2,R3,R4,R5,R6,R7,R8,R9],S)*:-*

*%  get position of blank cell*

    nth0(N,[R1,R2,R3,R4,R5,R6,R7,R8,R9],0),

    Z is N-3,

*%  get element in pos Z*

    nth0(Z,[R1,R2,R3,R4,R5,R6,R7,R8,R9],R),

*% substitute element of pos Z with blank cell “0”*

    substitute(R,[R1,R2,R3,R4,R5,R6,R7,R8,R9],10,Q),

    substitute(0,Q,R,V),

    substitute(10,V,0,S)*.*

*%  substitutes the first parameter with the third parameter and the third parameter with the first parameter in the second parameter (list) and produces a new list (forth parameter).*

*% e.g. substitute(1, [1,2,3,1,4], 4, X) will make X=[4,2,3,4,1]*

*% first parameter is value to be substituted by third parameter or replaces it.*

*% second parameter is the given list to be substituted.*

*% third parameter is the value that will replace the first parameter or will be substituted by the first parameter*

*% forth parameter is the new list after substitution.*

substitute(\_, [], \_, [])*:-!.*

substitute(X, [X|T], Y, [Y|T1])*:-*

  substitute(X, T, Y, T1),*!.*

substitute(X, [Y|T], Y, [X|T1])*:-*

  substitute(X, T, Y, T1),*!.*

substitute(X, [H|T], Y, [H|T1])*:-*

  substitute(X, T, Y, T1)*.*

*%  end of specific part*

*% General Algorithm to Solve the 8 Queens using Heusteric Search*

*% query of user and takes start state and next state*

go(Start,Goal)*:-*

      getHeuristic(Start, H, Goal),

      path([[Start,null, 0, H, H]],[],Goal)*.% open, closed, goal, path\_cost, heuristic, total cost*

*% main predicate that takes open list, closed list and goal state*

path([], \_, \_)*:-*

      write('No solution'),nl,*!.*

path(Open, Closed, Goal)*:-*

      getBestChild(Open, [Goal, Parent, PC, H, TC], RestOfOpen),

      write('A solution is found'),  nl ,

      printsolution([Goal,Parent, PC, H, TC], Closed),*!.*

path(Open, Closed, Goal)*:-*

      getBestChild(Open, [State, Parent, PC, H, TC], RestOfOpen),

      getchildren(State, Open, Closed, Children, PC, Goal),

      addListToOpen(Children , RestOfOpen, NewOpen),

      path(NewOpen, [[State, Parent, PC, H, TC] | Closed], Goal)*.*

*% gets Children of State that aren't in Open or Close*

getchildren(State, Open ,Closed , Children, PC, Goal)*:-*

      bagof(X, moves( State, Open, Closed, X, PC, Goal), Children)*.*

getchildren(\_,\_,\_, [],\_,\_)*.*

*% adds children to open list (without best child) to form new open list*

addListToOpen(Children, [], Children)*.*

addListToOpen(Children, [H|Open], [H|NewOpen])*:-*

      addListToOpen(Children, Open, NewOpen)*.*

*% gets the best state of the open list and another list without this best state*

*% first parameter is the open list*

*% second parameter is the best child*

*% third parameter is the open list without the best child*

getBestChild([Child], Child, [])*.*

getBestChild(Open, Best, RestOpen)*:-*

  getBestChild1(Open, Best),

  removeFromList(Best, Open, RestOpen)*.*

*% gets the best state of the open list*

getBestChild1([State], State)*.*

getBestChild1([State|Rest], Best)*:-*

  getBestChild1(Rest, Temp),

  getBest(State, Temp, Best)*.*

*% compares two states with each other (according to their Total cost) and returns the state with lower total cost TC*

getBest([State, Parent, PC, H, TC], [\_, \_, \_, \_, TC1], [State, Parent, PC, H, TC])*:-*

  TC < TC1, *!.*

getBest([\_, \_, \_, \_, \_], [State1, Parent1, PC1, H1, TC1], [State1, Parent1, PC1, H1, TC1])*.*

*% removes an element (usually the best state) from a list (open list) and returns a new list*

removeFromList(\_, [], [])*.*

removeFromList(H, [H|T], V)*:-*

*!*, removeFromList(H, T, V)*.*

removeFromList(H, [H1|T], [H1|T1])*:-*

  removeFromList(H, T, T1)*.*

*% gets next state given the current state*

moves( State, Open, Closed,[Next,State, NPC, H, TC], PC, Goal)*:-*

      move(State,Next),

      \+ member([Next, \_, \_, \_, \_],Open),

      \+ member([Next, \_, \_, \_, \_],Closed),

      NPC is PC + 1,

      getHeuristic(Next, H, Goal),

      TC is NPC + H*.*

*% calculate heuristic of some state*

*% here it is calculated as number of misplaced numbers*

getHeuristic([], 0, [])*:-!.*

getHeuristic([H|T1],V,[H|T2])*:-!*,

  getHeuristic(T1,V, T2)*.*

getHeuristic([\_|T1],H,[\_|T2])*:-*

  getHeuristic(T1,TH, T2),

  H is TH + 1*.*

*% prints the path from start state to goal state*

printsolution([State, null, PC, H, TC],\_)*:-*

      write(State), write(' PC: '), write(PC), write(' H:'), write(H), write(' TC: '), write(TC), nl*.*

printsolution([State, Parent, PC, H, TC], Closed)*:-*

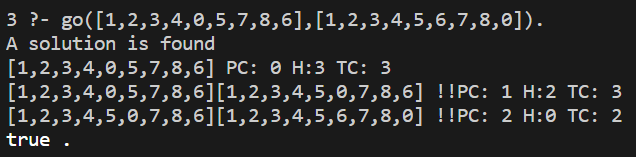
      member([Parent, GrandParent, PC1, H1, TC1], Closed),

      printsolution([Parent, GrandParent, PC1, H1, TC1], Closed),

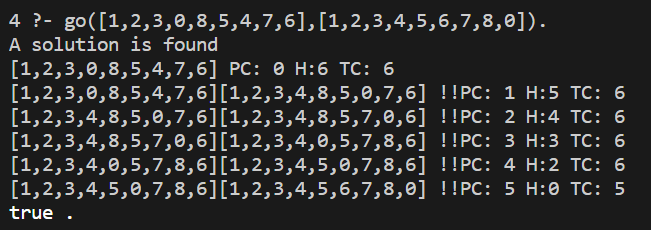
      write(Parent),    write(State), write(' !!PC: '), write(PC), write(' H:'), write(H), write(' TC: '), write(TC), nl*.*

**Output**

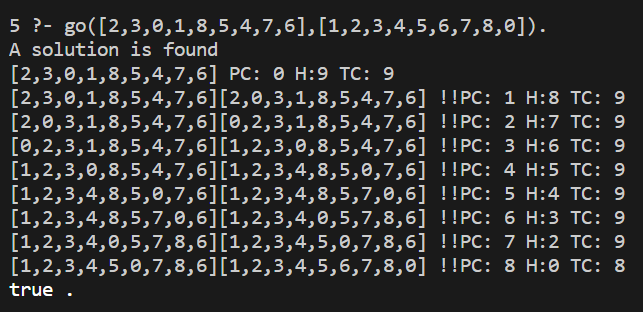
**Easy**: puzzle(state(1,2,3,4,\*,5,7,8,6)).



**Medium**: puzzle(state(1,2,3,\*,8,5,4,7,6)).



**Hard**: puzzle(state(2,3,\*,1,8,5,4,7,6)).



**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA