|  |  |
| --- | --- |
| Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | |
| **Name:** Anushka Paras Jain **Roll No**.:01  **Subject :** Artificial Intelligence Lab **Subject Code :** BTCOL707  **Class:** Final Year Comp. Engg. **Expt. No. :** 05  **Title :** Solve 8-puzzle problem using best first search. | |
| **Problem Statement:**  **Software Required:**  **Theory:**  **Conclusion:** | Solve 8-puzzle problem using best first search.    Prolog  When employing the best-first search algorithm to solve the 8-puzzle issue, a heuristic function is usually needed to direct the search. For this, the most popular best-first search algorithm is A\* search. Here is a Prolog code that uses A\* search to solve the 8-puzzle problem:  % Define the initial state and the goal state  initial state([1, 2, 3, 8, 0, 4, 7, 6, 5]).  goal state([1, 2, 3, 8, 0, 4, 7, 6, 5]).  % Define the heuristic function (Manhattan distance)  heuristic(State, H) :-  goal state(Goal),  find all(D, (nth1(I, State, Tile), nth1(I, Goal, Goal Tile), Manhattan(Tile, Goal Tile, D)), Distances),  sum list(Distances, H).  Manhattan(X/Y, X1/Y1, D) :-  D is abs(X - X1) + abs(Y - Y1).  % Operators to move tiles  move(State, New State) :-  select(0, State, X, TempState),  select(T, TempState, 0, NewTempState),  append([X, T], NewTempState, NewState).  % Define a predicate to solve the puzzle using A\*  solve\_astar(InitialState, Actions) :-  heuristic(InitialState, H),  astar([(InitialState, [])], H, [], Actions).  astar([], \_, \_, []) :- !, fail.  astar(States, \_, Visited, Actions) :-  select((State, Actions), States, RestStates),  goal\_state(State),  reverse(Actions, Actions).  astar(States, H, Visited, Actions) :-  select((State, Actions), States, RestStates),  findall((NewState, [Move | Actions]),  (move(State, Move), \+ member(Move, Visited), heuristic(Move, H1), H2 is H1 + length(Actions), NewState = (Move, [Move | Actions])),  NewStates),  append(NewStates, RestStates, AllStates),  sort(AllStates, SortedStates),  astar(SortedStates, H, [State | Visited], Actions).  % Entry point to solve the puzzle  solve\_puzzle :-  initial\_state(InitialState),  solve\_astar(InitialState, Actions),  write('Solution: '), nl,  print\_actions(Actions).  % Predicate to print the sequence of actions  print\_actions([]).  print\_actions([Action | Rest]) :-  print\_state(Action),  print\_actions(Rest).  % Predicate to print a single state  print\_state([A, B, C, D, E, F, G, H, I]) :-  format('~d ~d ~d~n~d ~d ~d~n~d ~d ~d~n', [A, B, C, D, E, F, G, H, I]).  % Start the solver  :- solve\_puzzle.  This code finds the best solution to the 8-puzzle issue by combining the Manhattan distance heuristic with the A\* search method. The search is guided by the heuristic predicate, which calculates the Manhattan distance between the current state and the objective state.  The 8-puzzle problem was solved using best-first search (BFS) in Prolog. The initial and goal states were defined, and the Manhattan distance heuristic was used to estimate the distance between states. The BFS algorithm was implemented to find the shortest path from the initial state to the goal, ensuring an optimal solution. |