|  |  |  |  |
| --- | --- | --- | --- |
| A picture containing drawing, stop, room  Description automatically generated | Artificial Intelligence  Practical #6 | | |
|  |  |  |  |
| **Name** | Sandeep Jain | **Roll Number** | 21302C0058 |
| **Subject/Course:** | **Artificial Intelligence** | | |
| **Topic** | **Search Algorithm** | | |
|  | | | |
| 1. Write a program to solve Missionaries and Cannibals problem. | | | |
| **Code:**  start([3,3,left,0,0]).  goal([0,0,right,3,3]).  legal(CL,ML,CR,MR) :-  % is this state a legal one?  ML>=0, CL>=0, MR>=0, CR>=0, (ML>=CL ; ML=0),  (MR>=CR ; MR=0).  % Possible moves:  move([CL,ML,left,CR,MR],[CL,ML2,right,CR,MR2]):-  % Two missionaries cross left to right.  MR2 is MR+2, ML2 is ML-2, legal(CL,ML2,CR,MR2).  move([CL,ML,left,CR,MR],[CL2,ML,right,CR2,MR]):-  % Two cannibals cross left to right.  CR2 is CR+2, CL2 is CL-2, legal(CL2,ML,CR2,MR).  move([CL,ML,left,CR,MR],[CL2,ML2,right,CR2,MR2]):-  % One missionary and one cannibal cross left to right.  CR2 is CR+1, CL2 is CL-1, MR2 is MR+1, ML2 is ML-1, legal(CL2,ML2,CR2,MR2).  move([CL,ML,left,CR,MR],[CL,ML2,right,CR,MR2]):-  % One missionary crosses left to right.  MR2 is MR+1, ML2 is ML-1, legal(CL,ML2,CR,MR2).  move([CL,ML,left,CR,MR],[CL2,ML,right,CR2,MR]):-  % One cannibal crosses left to right.  CR2 is CR+1, CL2 is CL-1, legal(CL2,ML,CR2,MR).  move([CL,ML,right,CR,MR],[CL,ML2,left,CR,MR2]):-  % Two missionaries cross right to left.  MR2 is MR-2, ML2 is ML+2, legal(CL,ML2,CR,MR2).  move([CL,ML,right,CR,MR],[CL2,ML,left,CR2,MR]):-  % Two cannibals cross right to left.  CR2 is CR-2, CL2 is CL+2, legal(CL2,ML,CR2,MR).  move([CL,ML,right,CR,MR],[CL2,ML2,left,CR2,MR2]):-  % One missionary and one cannibal cross right to left.  CR2 is CR-1, CL2 is CL+1, MR2 is MR-1, ML2 is ML+1,  legal(CL2,ML2,CR2,MR2).  move([CL,ML,right,CR,MR],[CL,ML2,left,CR,MR2]):-  % One missionary crosses right to left.  MR2 is MR-1, ML2 is ML+1, legal(CL,ML2,CR,MR2).  move([CL,ML,right,CR,MR],[CL2,ML,left,CR2,MR]):-  % One cannibal crosses right to left.  CR2 is CR-1, CL2 is CL+1, legal(CL2,ML,CR2,MR).  % Recursive call to solve the problem  path([CL1,ML1,B1,CR1,MR1],[CL2,ML2,B2,CR2,MR2],Explored,MovesList)  :-  move([CL1,ML1,B1,CR1,MR1],[CL3,ML3,B3,CR3,MR3]),  not(member([CL3,ML3,B3,CR3,MR3],Explored)),  path([CL3,ML3,B3,CR3,MR3],[CL2,ML2,B2,CR2,MR2],[[CL3,ML3,B3,CR3,  MR3]|Explored],[ [[CL3,ML3,B3,CR3,MR3],[CL1,ML1,B1,CR1,MR1]] |  MovesList ]).  % Solution found  path([CL,ML,B,CR,MR],[CL,ML,B,CR,MR],\_,MovesList):- output(MovesList).  **Output:** | | | |
| 1. Design an application to simulate number puzzle problem | | | |
| **Code:**  class Node:  def \_\_init\_\_(self, data, level, fval):  # Initialize the node with the data ,level of the node and the calculated fvalue  self.data = data  self.level = level  self.fval = fval  def generate\_child(self):  # Generate hild nodes from the given node by moving the blank space  # either in the four direction {up,down,left,right}  x, y = self.find(self.data, '\_')  # val\_list contains position values for moving the blank space in either of  # the 4 direction [up,down,left,right] respectively.  val\_list = [[x, y - 1], [x, y + 1], [x - 1, y], [x + 1, y]]  children = []  for i in val\_list:  child = self.shuffle(self.data, x, y, i[0], i[1])  if child is not None:  child\_node = Node(child, self.level + 1, 0)  children.append(child\_node)  return children  def shuffle(self, puz, x1, y1, x2, y2):  # Move the blank space in the given direction and if the position value are out  # of limits the return None  if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):  temp\_puz = []  temp\_puz = self.copy(puz)  temp = temp\_puz[x2][y2]  temp\_puz[x2][y2] = temp\_puz[x1][y1]  temp\_puz[x1][y1] = temp  return temp\_puz  else:  return None  def copy(self, root):  # copy function to create a similar matrix of the given node  temp = []  for i in root:  t = []  for j in i:  t.append(j)  temp.append(t)  return temp  def find(self, puz, x):  # Specifically used to find the position of the blank space  for i in range(0, len(self.data)):  for j in range(0, len(self.data)):  if puz[i][j] == x:  return i, j  class Puzzle:  def \_\_init\_\_(self, size):  # Initialize the puzzle size by the the specified size,open and closed lists to empty  self.n = size  self.open = []  self.closed = []  def accept(self):  # Accepts the puzzle from the user  puz = []  for i in range(0, self.n):  temp = input().split(" ")  puz.append(temp)  return puz  def f(self, start, goal):  # Heuristic function to calculate Heuristic value f(x) = h(x) + g(x)  return self.h(start.data, goal) + start.level  def h(self, start, goal):  # Calculates the difference between the given puzzles  temp = 0  for i in range(0, self.n):  for j in range(0, self.n):  if start[i][j] != goal[i][j] and start[i][j] != '\_':  temp += 1  return temp  def process(self):  # Accept Start and Goal Puzzle state  print("enter the start state matrix \n")  start = self.accept()  print("enter the goal state matrix \n")  goal = self.accept()  start = Node(start, 0, 0)  start.fval = self.f(start, goal)  # put the start node in the open list  self.open.append(start)  print("\n\n")  while True:  cur = self.open[0]  print("==================================================\n")  for i in cur.data:  for j in i:  print(j, end=" ")  print("")  # if the difference between current and goal node is 0 we have reached the goal node  if (self.h(cur.data, goal) == 0):  break  for i in cur.generate\_child():  i.fval = self.f(i, goal)  self.open.append(i)  self.closed.append(cur)  del self.open[0]  # sort the open list based on f value  self.open.sort(key=lambda x: x.fval, reverse=False)  puz = Puzzle(3)  puz.process()  **Output:**  image | | | |
|  | | | |