PROGRAM 1:

|  |
| --- |
| # Importing copy for deepcopy function |
|  |

|  |
| --- |
| import copy |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| from heapq import heappush, heappop |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| n = 3 |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # bottom, left, top, right |
|  |

|  |
| --- |
| row = [1, 0, -1, 0] |
|  |

|  |
| --- |
| col = [0, -1, 0, 1] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| class priorityQueue: |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_init\_\_(self): |
|  |

|  |
| --- |
| self.heap = [] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def push(self, k): |
|  |

|  |
| --- |
| heappush(self.heap, k) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def pop(self): |
|  |

|  |
| --- |
| return heappop(self.heap) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def empty(self): |
|  |

|  |
| --- |
| if not self.heap: |
|  |

|  |
| --- |
| return True |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| return False |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Node structure |
|  |

|  |
| --- |
| class node: |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_init\_\_(self, parent, mat, empty\_tile\_pos, |
|  |

|  |
| --- |
| cost, level): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| self.parent = parent |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Stores the matrix |
|  |

|  |
| --- |
| self.mat = mat |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| self.empty\_tile\_pos = empty\_tile\_pos |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Stores the number of misplaced tiles |
|  |

|  |
| --- |
| self.cost = cost |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Stores the number of moves so far |
|  |

|  |
| --- |
| self.level = level |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def \_\_lt\_\_(self, nxt): |
|  |

|  |
| --- |
| return self.cost < nxt.cost |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def calculateCost(mat, final) -> int: |
|  |

|  |
| --- |
| count = 0 |
|  |

|  |
| --- |
| for i in range(n): |
|  |

|  |
| --- |
| for j in range(n): |
|  |

|  |
| --- |
| if ((mat[i][j]) and |
|  |

|  |
| --- |
| (mat[i][j] != final[i][j])): |
|  |

|  |
| --- |
| count += 1 |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return count |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def newNode(mat, empty\_tile\_pos, new\_empty\_tile\_pos, |
|  |

|  |
| --- |
| level, parent, final) -> node: |
|  |

|  |
| --- |
| new\_mat = copy.deepcopy(mat) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| x1 = empty\_tile\_pos[0] |
|  |

|  |
| --- |
| y1 = empty\_tile\_pos[1] |
|  |

|  |
| --- |
| x2 = new\_empty\_tile\_pos[0] |
|  |

|  |
| --- |
| y2 = new\_empty\_tile\_pos[1] |
|  |

|  |
| --- |
| new\_mat[x1][y1], new\_mat[x2][y2] = new\_mat[x2][y2], new\_mat[x1][y1] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| cost = calculateCost(new\_mat, final) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| new\_node = node(parent, new\_mat, new\_empty\_tile\_pos, |
|  |

|  |
| --- |
| cost, level) |
|  |

|  |
| --- |
| return new\_node |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Function to print the N x N matrix |
|  |

|  |
| --- |
| def printMatrix(mat): |
|  |

|  |
| --- |
| for i in range(n): |
|  |

|  |
| --- |
| for j in range(n): |
|  |

|  |
| --- |
| print("%d " % (mat[i][j]), end=" ") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| print() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Function to check if (x, y) is a valid |
|  |

|  |
| --- |
| # matrix coordinate |
|  |

|  |
| --- |
| def isSafe(x, y): |
|  |

|  |
| --- |
| return x >= 0 and x < n and y >= 0 and y < n |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Print path from root node to destination node |
|  |

|  |
| --- |
| def printPath(root): |
|  |

|  |
| --- |
| if root == None: |
|  |

|  |
| --- |
| return |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| printPath(root.parent) |
|  |

|  |
| --- |
| printMatrix(root.mat) |
|  |

|  |
| --- |
| print() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def solve(initial, empty\_tile\_pos, final): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| pq = priorityQueue() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| cost = calculateCost(initial, final) |
|  |

|  |
| --- |
| root = node(None, initial, |
|  |

|  |
| --- |
| empty\_tile\_pos, cost, 0) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| pq.push(root) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| while not pq.empty(): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| minimum = pq.pop() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # If minimum is the answer node |
|  |

|  |
| --- |
| if minimum.cost == 0: |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| printPath(minimum) |
|  |

|  |
| --- |
| return |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Generate all possible children |
|  |

|  |
| --- |
| for i in range(4): |
|  |

|  |
| --- |
| new\_tile\_pos = [ |
|  |

|  |
| --- |
| minimum.empty\_tile\_pos[0] + row[i], |
|  |

|  |
| --- |
| minimum.empty\_tile\_pos[1] + col[i], ] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| if isSafe(new\_tile\_pos[0], new\_tile\_pos[1]): |
|  |

|  |
| --- |
| # Create a child node |
|  |

|  |
| --- |
| child = newNode(minimum.mat, |
|  |

|  |
| --- |
| minimum.empty\_tile\_pos, |
|  |

|  |
| --- |
| new\_tile\_pos, |
|  |

|  |
| --- |
| minimum.level + 1, |
|  |

|  |
| --- |
| minimum, final, ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Add child to list of live nodes |
|  |

|  |
| --- |
| pq.push(child) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Driver Code |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| initial = [[1, 2, 3], |
|  |

|  |
| --- |
| [5, 6, 0], |
|  |

|  |
| --- |
| [7, 8, 4]] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| final = [[1, 2, 3], |
|  |

|  |
| --- |
| [5, 8, 6], |
|  |

|  |
| --- |
| [0, 7, 4]] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Blank tile coordinates in |
|  |

|  |
| --- |
| # initial configuration |
|  |

|  |
| --- |
| empty\_tile\_pos = [1, 2] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Function call to solve the puzzle |
|  |

solve(initial, empty\_tile\_pos, final)

OUTPUT:

