**Task#01:**

def dfs(graph, start, goal):

visited = set()

stack = [start]

print("Start -> ",start)

while stack:

node = stack.pop()

print("Visted -> ",graph[node])

if node == goal:

print("Goal -> ",goal)

return True

if node not in visited:

visited.add(node)

stack.extend(graph[node])

return False

graph = {

"A": ["B", "D"],

"B": ["C", "E"],

"C": [],

"D": ["E", "H", "G"],

"E": ["C", "F"],

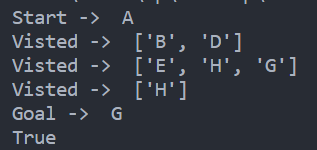
"F": [],

"G": ["H"]

}

print(dfs(graph, "A", "G"))

Output:



**Task#03  
from queue import Queue**

**class EightQ:**

**def \_\_init\_\_(self, initial\_state, goal\_state):**

**self.initial\_state = initial\_state**

**self.goal\_state = goal\_state**

**def get\_successors(self, state):**

**successors = []**

**row, col = self.find\_blank(state)**

**if row > 0:**

**successor = [row[:] for row in state]**

**successor[row][col], successor[row-1][col] = successor[row-1][col], successor[row][col]**

**successors.append(successor)**

**if row < 2:**

**successor = [row[:] for row in state]**

**successor[row][col], successor[row+1][col] = successor[row+1][col], successor[row][col]**

**successors.append(successor)**

**if col > 0:**

**successor = [row[:] for row in state]**

**successor[row][col], successor[row][col-1] = successor[row][col-1], successor[row][col]**

**successors.append(successor)**

**if col < 2:**

**successor = [row[:] for row in state]**

**successor[row][col], successor[row][col+1] = successor[row][col+1], successor[row][col]**

**successors.append(successor)**

**return successors**

**def find\_blank(self, state):**

**for row in range(3):**

**for col in range(3):**

**if state[row][col] == 0:**

**return row, col**

**def perform\_bfs(self):**

**visited = set()**

**queue = Queue()**

**queue.put(self.initial\_state)**

**while not queue.empty():**

**state = queue.get()**

**if state == self.goal\_state:**

**return state**

**visited.add(tuple(map(tuple, state)))**

**successors = self.get\_successors(state)**

**for successor in successors:**

**if tuple(map(tuple, successor)) not in visited:**

**queue.put(successor)**

**return None**

**initial\_state = [[1, 2, 3],**

**[4, 0, 5],**

**[6, 7, 8]]**

**goal\_state = [[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 0]]**

**puzzle = EightQ(initial\_state, goal\_state)**

**solution = puzzle.perform\_bfs()**

**if solution:**

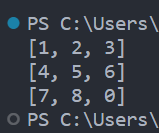
**for row in solution:**

**print(row)**

**else:**

**print("No solution found.")**

Output:



**Task#03**

**from queue import PriorityQueue**

**graph = {**

**"A": ["B", "D"],**

**"B": ["C", "E"],**

**"C": [],**

**"D": ["E", "H", "G"],**

**"E": ["C", "F"],**

**"F": [],**

**"G": ["H"]**

**}**

**def greedy\_best\_first\_search(graph, start):**

**visited = set()**

**queue = PriorityQueue()**

**queue.put((0, start))**

**while not queue.empty():**

**shush, node = queue.get()**

**if node == "C":**

**return True**

**if node not in visited:**

**visited.add(node)**

**for neighbor in graph[node]:**

**h = len([n for n in graph[neighbor] if n != "C"])**

**queue.put((h, neighbor))**

**return False**

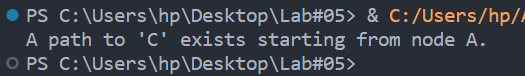
**if greedy\_best\_first\_search(graph, "A"):**

**print("A path to 'C' exists starting from node A.")**

**else:**

**print("No path to 'C' exists starting from node A.")**

**Output:**

****