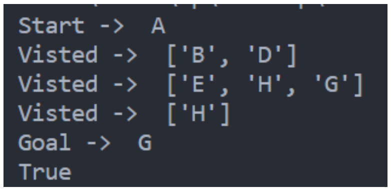
Muhammad Waleed
20b-115-se SE-B
AI Lab#05
Sir Nasir Ud Deen

Task#01:

```python
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:



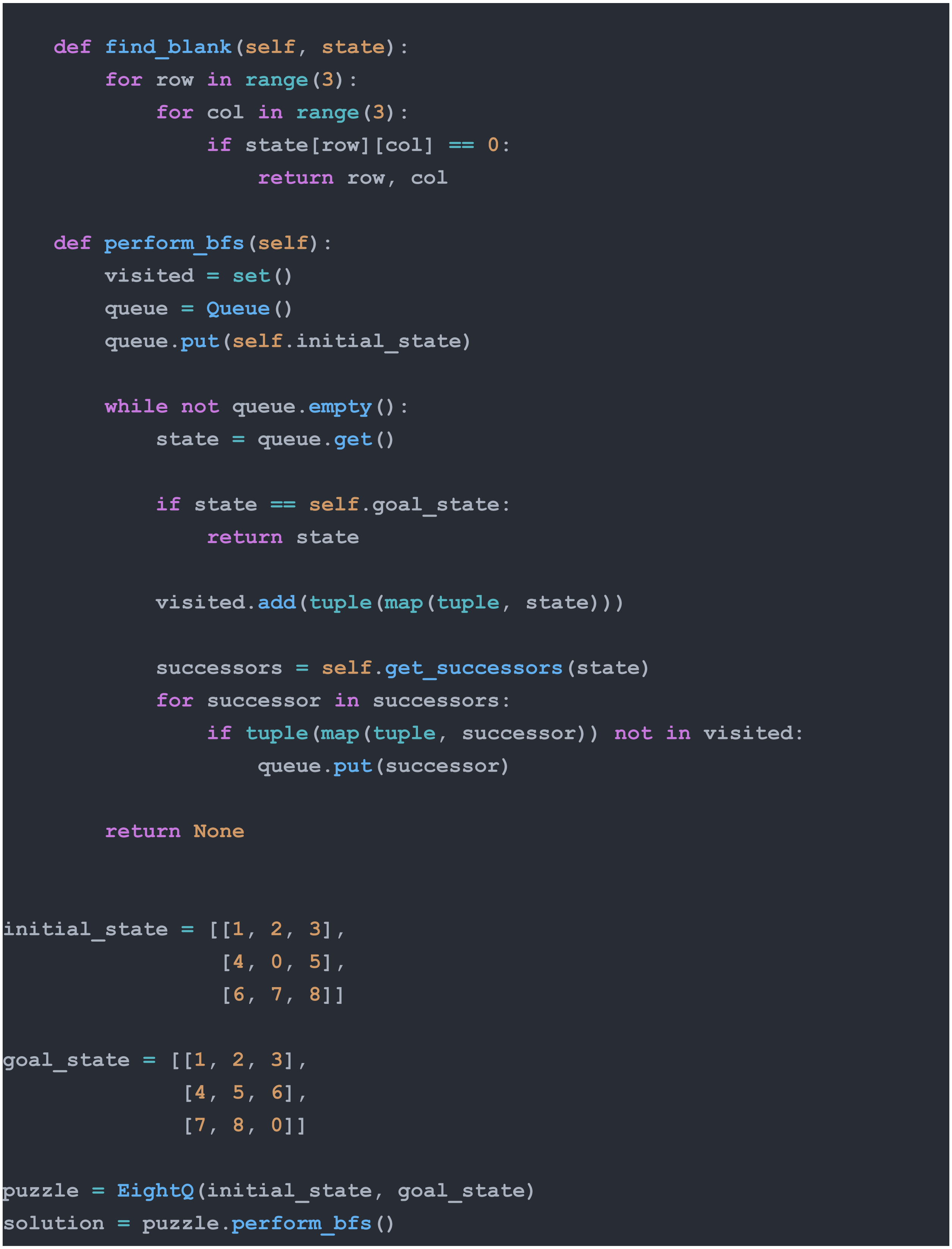
```
Start ->  A
Visted ->  ['B', 'D']
Visted ->  ['E', 'H', 'G']
Visted ->  ['H']
Goal ->  G
True
```

Muhammad Waleed
20b-115-se SE-B
AI Lab#05
Sir Nasir Ud Deen

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
```

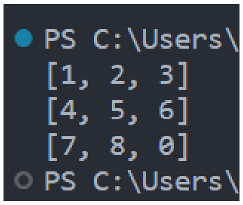
Muhammad Waleed
20b-115-se SE-B
AI Lab#05
Sir Nasir Ud Deen



```
    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:

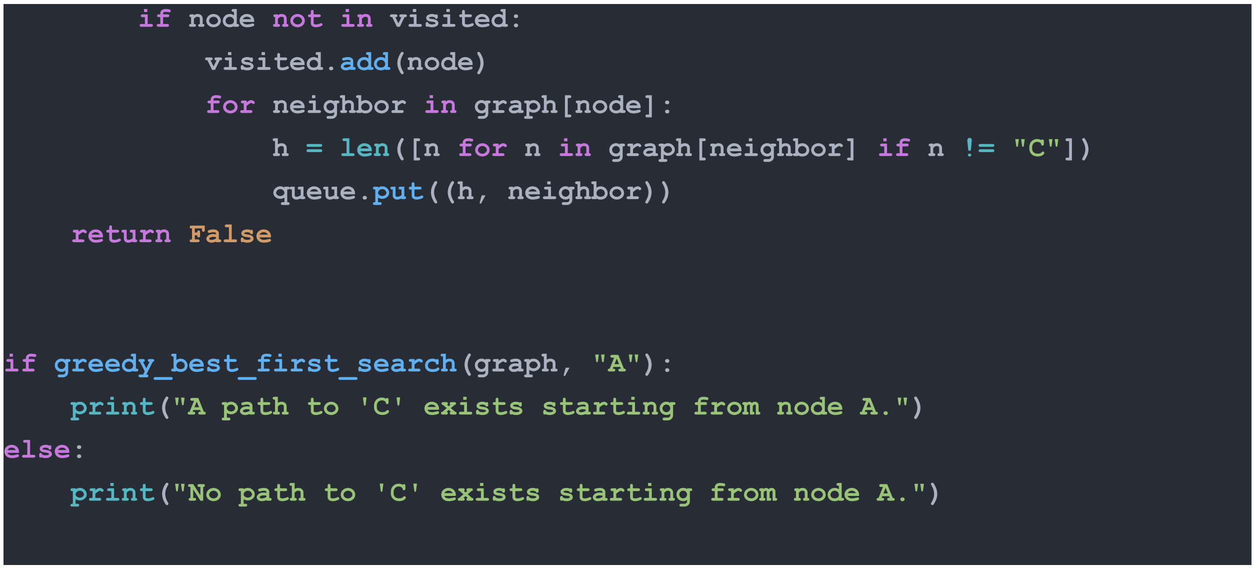


```
PS C:\Users\
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
PS C:\Users\
```

Task#03

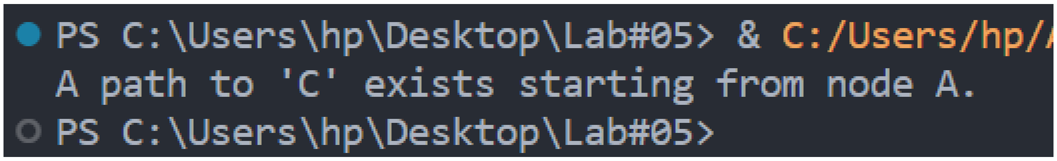
```
from queue import PriorityQueue

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
```

Muhammad Waleed
20b-115-se SE-B
AI Lab#05
Sir Nasir Ud Deen



```
        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:



```
● PS C:\Users\hp\Desktop\Lab#05> & C:/Users/hp/
  A path to 'C' exists starting from node A.
○ PS C:\Users\hp\Desktop\Lab#05>
```