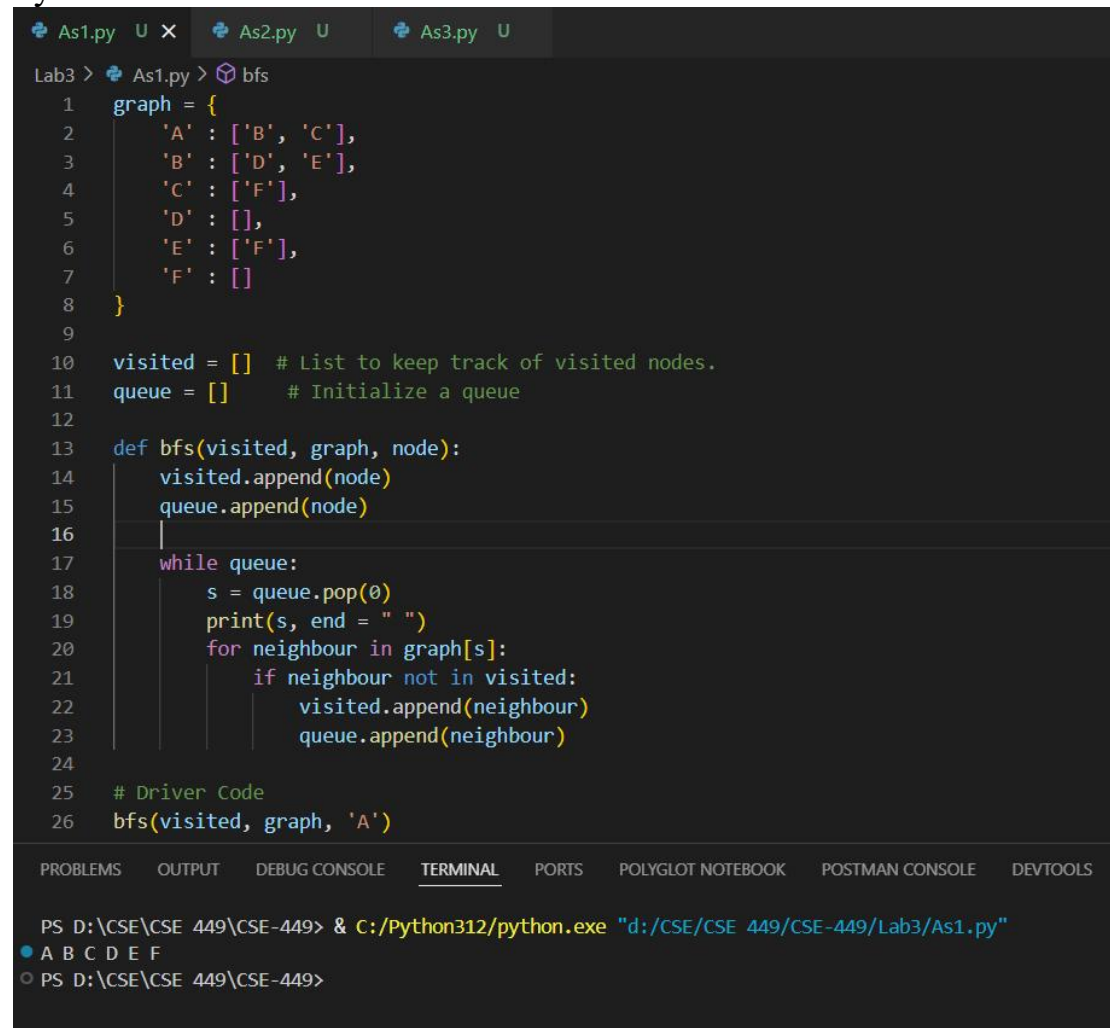


CSE 449 Lab 3

Name: Nguyễn Hữu Hà
Student Id: 2031200078

Assignment 1: Write a Program to Implement Breath First Search using Python



```
Lab3 > As1.py > bfs
1 graph = {
2     'A' : ['B', 'C'],
3     'B' : ['D', 'E'],
4     'C' : ['F'],
5     'D' : [],
6     'E' : ['F'],
7     'F' : []
8 }
9
10 visited = [] # List to keep track of visited nodes.
11 queue = []    # Initialize a queue
12
13 def bfs(visited, graph, node):
14     visited.append(node)
15     queue.append(node)
16
17     while queue:
18         s = queue.pop(0)
19         print(s, end = " ")
20         for neighbour in graph[s]:
21             if neighbour not in visited:
22                 visited.append(neighbour)
23                 queue.append(neighbour)
24
25 # Driver Code
26 bfs(visited, graph, 'A')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POLYGLOT NOTEBOOK POSTMAN CONSOLE DEVTOOLS

```
PS D:\CSE\CSE 449\CSE-449> & C:/Python312/python.exe "d:/CSE/CSE 449/CSE-449/Lab3/As1.py"
A B C D E F
PS D:\CSE\CSE 449\CSE-449>
```

Assignment 2: Write a Program to Implement Depth First Search using Python

```
As1.py U  As2.py U X  As3.py U
Lab3 > As2.py > ...
1  # Using a Python dictionary to act as an adjacency list
2  graph = {
3      'A': ['B', 'C'],
4      'B': ['D', 'E'],
5      'C': ['F'],
6      'D': [],
7      'E': ['F'],
8      'F': []
9  }
10
11  visited = set() # Set to keep track of visited nodes.
12  def dfs(visited, graph, node):
13      if node not in visited:
14          print(node) # Print the node if it hasn't been visited yet
15          visited.add(node) # Mark the node as visited
16          for neighbour in graph[node]: # Recur for all the neighbours of the node
17              dfs(visited, graph, neighbour)
18
19  # Drivers code
20  dfs(visited, graph, 'A') # Start the DFS from node 'A'

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POLYGLOT NOTEBOOK  POSTMAN CONSOLE  DEVTOOLS

PS D:\CSE\CSE 449\CSE-449> & C:/Python312/python.exe "d:/CSE/CSE 449/CSE-449/Lab3/As2.py"
A
B
D
E
F
C
PS D:\CSE\CSE 449\CSE-449> 
```

Assignment 3: Write a Program to Implement Tic-Tac-Toe game using Python

As1.py U As2.py U X As3.py U X

Lab3 > As3.py > play_game

```
1  # Tic-Tac-Toe program using
2  # random number in Python
3
4  # importing all necessary libraries
5  import numpy as np
6  import random
7  from time import sleep
8
9  # Creates an empty board
10 def create_board():
11     return(np.array([[0, 0, 0],
12                     [0, 0, 0],
13                     [0, 0, 0]]))
14
15 # Check for empty places on the board
16 def possibilities(board):
17     l = []
18     for i in range(len(board)):
19         for j in range(len(board)):
20             if board[i][j] == 0:
21                 l.append((i, j))
22     return(l)
23
24 # Select a random place for the player
25 def random_place(board, player):
26     selection = possibilities(board)
27     current_loc = random.choice(selection)
28     board[current_loc] = player
29     return(board)
30
31 # Check wheter the player has three
32 # of their marks in a horizontal row
33 def row_win(board, player):
34     for x in range(len(board)):
35         win = True
36         for y in range(len(board)):
37             if board[x,y] != player:
38                 win = False
39                 continue
40         if win == True:
41             return win
42     return win
43
44 # Check wheter the player has three
45 # of their marks in a vertical row
46 def col_win(board, player):
```

```

47     for x in range(len(board)):
48         win = True
49         for y in range(len(board)):
50             if board[y][x] != player:
51                 win = False
52                 continue
53         if win == True:
54             return win
55     return win
56
57     # Check wheter the player has three
58     # of their marks in a diagonal row
59     def diag_win(board, player):
60         win = True
61         for x in range(len(board)):
62             if board[x,x] != player:
63                 win = False
64         if win:
65             return win
66     win = True
67     if win:
68         for x in range(len(board)):
69             y = len(board) - 1 - x
70             if board[x,y] != player:
71                 win = False
72     return win
73
74     # Evaluates wheter there is
75     # a winner or a tie
76     def evaluate(board):
77         winner = 0
78         for player in [1, 2]:
79             if (row_win(board, player) or
80                 col_win(board, player) or
81                 diag_win(board, player)):
82                 winner = player
83     if np.all(board != 0) and winner == 0:
84         return -1
85     else:
86         return winner
87

```

```

88     # Main function to start the game
89     def play_game():
90         board, winner, counter = create_board(), 0, 1
91         print(board)
92         sleep(2)
93         while winner == 0:
94             for player in [1, 2]:
95                 board = random_place(board, player)
96                 print("Board after " + str(counter) + " move")
97                 print(board)
98                 sleep(2)
99                 counter += 1
100                winner = evaluate(board)
101                winner = evaluate(board)
102                if winner != 0:
103                    break
104            return winner
105
106     # Driver code
107     print("Winner is: " + str(play_game()))

```

● PS D:\CSE\CSE 449\CSE-449> & C:/Python312/python.exe "d:/CSE/CSE 449/CSE-449/Lab3/As3.py"

```

[[0 0 0]
 [0 0 0]
 [0 0 0]]
Board after 1 move
[[0 1 0]
 [0 0 0]
 [0 0 0]]
Board after 2 move
[[0 1 0]
 [0 0 0]
 [2 0 0]]
Board after 3 move
[[1 1 0]
 [0 0 0]
 [2 0 0]]
Board after 4 move
[[1 1 0]
 [0 0 2]
 [2 0 0]]
Board after 5 move
[[1 1 0]
 [0 0 2]
 [2 0 1]]
Board after 6 move
[[1 1 0]
 [0 2 2]
 [2 0 1]]
Board after 7 move
[[1 1 0]
 [0 2 2]
 [2 1 1]]
Board after 8 move
[[1 1 2]
 [0 2 2]
 [2 1 1]]
Winner is: 2

```