ASSIGNMENT-1(a)

PROBLEM STATEMENT: Implement **BFS** (Breadth-first Search) algorithms to find the path from source to goal node for a given graph.

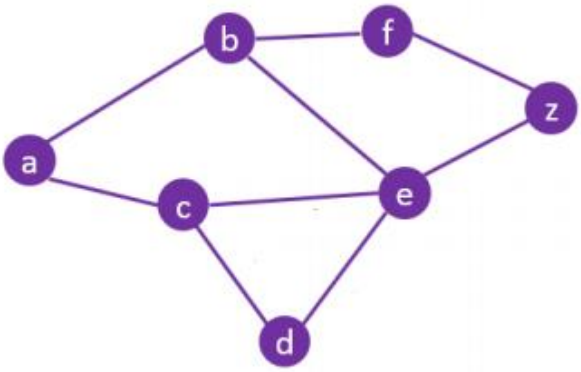
SOURCE CODE (Python3): I basically divided this program in 2 parts-
PART 1: In this part I took input from user to create **adjacent matrix** for a graph in which user want to perform **BFS** and **return** to the 2nd part**.**

```
import numpy as np

def get_input():
    no_of_nodes = int(input("Enter no. of nodes for tree: "))
    print("Please enter the adjacent matrix for your tree...")

    adjacent_matrix = np.zeros((no_of_nodes, no_of_nodes), dtype=int)
    node_list = []

    for i in range(no_of_nodes):    # TODO: get node name from user
        while 1:
            node = input(f"Enter name for node{i+1}: ")
            if node not in node_list:
                node_list.append(node)
                break
            else:
                print("The node already exist...")

    print("\nPlease enter 0 for 'No' and 1 for 'Yes'...")
    for i in range(len(node_list)):  # TODO: get all connected graph from user(undirected)
        for j in range(len(node_list)):
            # ! remember a node cannot connect with itself
            if node_list[i] == node_list[j] or adjacent_matrix[i][j] == 1:
                continue

            if adjacent_matrix[j][i] == 1:  # ! we tried to get undirected graph
                # ! so if node1 connected with node2 the vice-versa also true
                adjacent_matrix[i][j] = 1
                continue

            while 1:
                check_connection = int(
                    input(f"Is node '{node_list[j]}' connected with '{node_list[i]}': "))

                if check_connection == 1 or check_connection == 0:
                    adjacent_matrix[i][j] = check_connection
                    adjacent_matrix[j][i] = check_connection
                    break
                else:
                    print("Wrong input....try again")

    return adjacent_matrix, node_list
```

PART 2: In this part I took input from user to get starting node and ending node from user and perform **BFS.**

```
import get_input

adjacent_matrix, node_list = get_input.get_input() # TODO: get data from part-1
queue = []
visited_node = []

while 1:
   start_node = input("\nEnter your starting node: ")
   end_node = input("Enter your end node: ")
   # ! remember startnode and end node must match with node names
   if start_node not in node_list and end_node not in node_list:
      print("Wrong input try again...")
   else:
      break

# TODO: PUSH the starting node into queue
queue.append(node_list.index(start_node))

while len(queue) > 0:
   top_element = queue.pop(0)   # TODO: POP last element from QUEUE
   if top_element not in visited_node:  # TODO: if the node is visited then we ignore it
      visited_node.append(top_element)

      # TODO: PUSH all nodes into QUEUE if it is not visited
      for i in range(len(node_list)):
         if adjacent_matrix[top_element][i] == 1 and i not in visited_node:
            queue.append(i)

      if node_list[top_element] == end_node:
         print("We found the end node....")
         print("*** Our visited nodes are ***")

         # TODO: print the output in Style
         for i in visited_node:
            print(f"| ({node_list[i]})", end=" ")
         print("|")
         break
```

OUTPUT: Here, find the path from source to goal node for a given graph where the source node is **'a'** and the goal node is **'z'**.



```
D:\PROGRAM\PYTHON\CS-50\AI>python -u "d:\PROGRAM\PYTHON\CS-50\AI\bfs_code.py"
Enter no. of nodes for tree: 7
Please enter the adjacent matrix for your tree...
Enter name for node1: a
Enter name for node2: b
Enter name for node3: c
Enter name for node4: d
Enter name for node5: e
Enter name for node6: f
Enter name for node7: z

Please enter 0 for 'No' and 1 for 'Yes'...
Is node 'b' connected with 'a': 1
Is node 'c' connected with 'a': 1
Is node 'd' connected with 'a': 0
Is node 'e' connected with 'a': 0
Is node 'f' connected with 'a': 0
Is node 'z' connected with 'a': 0
Is node 'c' connected with 'b': 0
Is node 'd' connected with 'b': 0
Is node 'e' connected with 'b': 1
Is node 'f' connected with 'b': 1
Is node 'z' connected with 'b': 0
Is node 'b' connected with 'c': 0
Is node 'd' connected with 'c': 1
Is node 'e' connected with 'c': 1
Is node 'f' connected with 'c': 0
Is node 'z' connected with 'c': 0
Is node 'a' connected with 'd': 0
Is node 'b' connected with 'd': 0
Is node 'e' connected with 'd': 1
Is node 'f' connected with 'd': 0
Is node 'z' connected with 'd': 0
Is node 'a' connected with 'e': 0
Is node 'f' connected with 'e': 0
Is node 'z' connected with 'e': 1
Is node 'a' connected with 'f': 0
Is node 'c' connected with 'f': 0
Is node 'd' connected with 'f': 0
Is node 'e' connected with 'f': 0
Is node 'z' connected with 'f': 1
Is node 'a' connected with 'z': 0
Is node 'b' connected with 'z': 0
Is node 'c' connected with 'z': 0
Is node 'd' connected with 'z': 0

Enter your starting node: a
Enter your end node: z
We found the end node....
*** Our visited nodes are ***
| (a) | (b) | (c) | (e) | (f) | (d) | (z) |
```

ASSIGNMENT-1(b)

PROBLEM STATEMENT: Implement **DFS** (Depth-first Search) algorithms to find the path from source to goal node for a given graph.

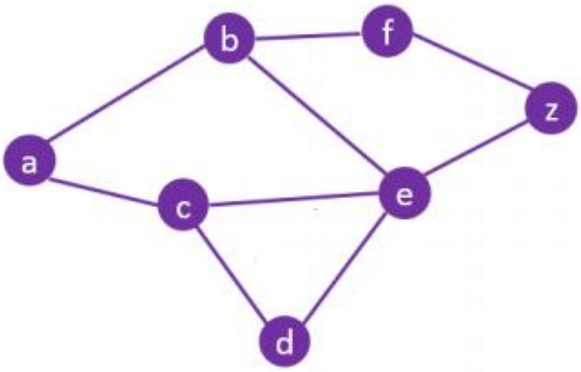
SOURCE CODE (Python3): I basically divided this program in 2 parts-
PART 1: In this part I took input from user to create **adjacent matrix** for a graph in which user want to perform **DFS** and **return** to the 2nd part**.**

```
import numpy as np

def get_input():
    no_of_nodes = int(input("Enter no. of nodes for tree: "))
    print("Please enter the adjacent matrix for your tree...")

    adjacent_matrix = np.zeros((no_of_nodes, no_of_nodes), dtype=int)
    node_list = []

    for i in range(no_of_nodes):    # TODO: get node name from user
        while 1:
            node = input(f"Enter name for node{i+1}: ")
            if node not in node_list:
                node_list.append(node)
                break
            else:
                print("The node already exist...")

    print("\nPlease enter 0 for 'No' and 1 for 'Yes'...")
    for i in range(len(node_list)):  # TODO: get all connected graph from user(undirected)
        for j in range(len(node_list)):
            # ! remember a node cannot connect with itself
            if node_list[i] == node_list[j] or adjacent_matrix[i][j] == 1:
                continue

            if adjacent_matrix[j][i] == 1:  # ! we tried to get undirected graph
                # ! so if node1 connected with node2 the vice-versa also true
                adjacent_matrix[i][j] = 1
                continue

            while 1:
                check_connection = int(
                    input(f"Is node '{node_list[j]}' connected with '{node_list[i]}': "))

                if check_connection == 1 or check_connection == 0:
                    adjacent_matrix[i][j] = check_connection
                    adjacent_matrix[j][i] = check_connection
                    break
                else:
                    print("Wrong input....try again")

    return adjacent_matrix, node_list
```

PART 2: In this part I took input from user to get starting node and ending node from user and perform **DFS.**

```
import get_input

# adjacent_matrix, node_list = get_input.get_input()  # TODO: get data from part-1
stack = []
visited_node = []

while 1:
    start_node = input("\nEnter your starting node: ")
    end_node = input("Enter your end node: ")
    # ! remember startnode and end node must match with node names
    if start_node not in node_list and end_node not in node_list:
        print("Wrong input try again...")
    else:
        break

# TODO: PUSH the starting node into stack
stack.append(node_list.index(start_node))

while len(stack) > 0:
    top_element = stack.pop()   # TODO: POP top element from STACK
    if top_element not in visited_node:  # TODO: if the node is visited then we ignore it
        visited_node.append(top_element)

        # TODO: PUSH all nodes into STACK if it is not visited
        for i in range(len(node_list)):
            if adjacent_matrix[top_element][i] == 1 and i not in visited_node:
                stack.append(i)

        if node_list[top_element] == end_node:
            print("We found the end node....")
            print("*** Our visited nodes are ***")

            # TODO: print the output in Style
            for i in visited_node:
                print(f"| ({node_list[i]})", end=" ")
            print("|")
            break
```

OUTPUT: Here, find the path from source to goal node for a given graph where the source node is **‘a’** and the goal node is **‘z’**.



```
D:\PROGRAM\PYTHON\CS-50\AI>python -u "d:\PROGRAM\PYTHON\CS-50\AI\dfs_code.py"
Enter no. of nodes for tree: 7
Please enter the adjacent matrix for your tree...
Enter name for node1: a
Enter name for node2: b
Enter name for node3: c
Enter name for node4: d
Enter name for node5: e
Enter name for node6: f
Enter name for node7: z

Please enter 0 for 'No' and 1 for 'Yes'...
Is node 'b' connected with 'a': 1
Is node 'c' connected with 'a': 1
Is node 'd' connected with 'a': 0
Is node 'e' connected with 'a': 0
Is node 'f' connected with 'a': 0
Is node 'z' connected with 'a': 0
Is node 'c' connected with 'b': 0
Is node 'd' connected with 'b': 0
Is node 'e' connected with 'b': 1
Is node 'f' connected with 'b': 1
Is node 'z' connected with 'b': 0
Is node 'b' connected with 'c': 0
Is node 'd' connected with 'c': 1
Is node 'e' connected with 'c': 1
Is node 'f' connected with 'c': 0
Is node 'z' connected with 'c': 0
Is node 'a' connected with 'd': 0
Is node 'b' connected with 'd': 0
Is node 'e' connected with 'd': 1
Is node 'f' connected with 'd': 0
Is node 'z' connected with 'd': 0
Is node 'a' connected with 'e': 0
Is node 'f' connected with 'e': 0
Is node 'z' connected with 'e': 1
Is node 'a' connected with 'f': 0
Is node 'c' connected with 'f': 0
Is node 'd' connected with 'f': 0
Is node 'e' connected with 'f': 0
Is node 'z' connected with 'f': 1
Is node 'a' connected with 'z': 0
Is node 'b' connected with 'z': 0
Is node 'c' connected with 'z': 0
Is node 'd' connected with 'z': 0

Enter your starting node: a
Enter your end node: z
We found the end node....
*** Our visited nodes are ***
| (a) | (c) | (e) | (z) |
```

ASSIGNMENT-2

PROBLEM STATEMENT: Consider the following goal state of 8-puzzle problem. Given any arbitrary state (to be supplied by the user), compute the heuristic value of the input state for the following heuristic functions.

a) Number of misplaced tiles (h1).

b) Manhattan distance (h2).

1	2	3
4	5	6
7	8	

SOURCE CODE (Python3):

```
import numpy as np
import math

goal_state = np.array([[1, 2, 3], [4, 5, 6], [7, 8, ' ']])
print(“This is your goal state:", goal_state)

input_state = [''] * (3**2)
input_state = np.array(input_state)
input_state.shape = (3, 3)

print("Now, Enter your elements from top-left:\nEnter space to let a cell empty...")
for i in range(3):
   for j in range(3):
      while 1:
         element = input(f"Enter element for point({i}, {j}): ")
         if element in goal_state and element not in input_state:
            input_state[i][j] = element
            break
         else:
            print("Given data already exist or not matched with output data...")

print("Your input state is:\n", input_state)

misplaced_tiles = 0
manhatten_distance = 0
for i in range(3):
   for j in range(3):
      if goal_state[i][j] != input_state[i][j]:
         misplaced_tiles += 1
         for x1 in range(3):
            for x2 in range(3):
               if goal_state[i][j] == input_state[x1][x2]:
                  manhatten_distance += math.ceil(
                     math.sqrt((x1-i)**2 + (x2-j)**2))

print("No. of misplaced tiles(h1): ", misplaced_tiles)
print("Manhatten Distance(h2): ", manhatten_distance)
```

OUTPUT:

```
D:\PROGRAM\PYTHON\AI>python –u "d:\PROGRAM\PYTHON\AI\8_puzzle_problem.py"
This is your goal state:
[['1' '2' '3']
 ['4' '5' '6']
 ['7' '8' ' ']]
Now, Enter your elements from top-left:
Enter space to let a cell empty...
Enter element for point(0, 0): 1
Enter element for point(0, 1): 2
Enter element for point(0, 2): 3
Enter element for point(1, 0): 4
Enter element for point(1, 1):
Enter element for point(1, 2): 6
Enter element for point(2, 0):
Given data already exist or not matched with output data...
Enter element for point(2, 0): 7
Enter element for point(2, 1): 8
Enter element for point(2, 2): 5
Your input state is:
 [['1' '2' '3']
 ['4' ' ' '6']
 ['7' '8' '5']]
No. of misplaced tiles(h1):  2
Manhatten Distance(h2):  4
```