```
#1 BFS QUESTION
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        vertex = queue.popleft()
        print(vertex, end=' ')
        for neighbor in graph[vertex]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
graph = {
    0: [1, 3],
    1: [0, 3, 2],
    2: [4, 5],
    3: [0,1,4],
    4: [2,3,5],
    5: [2,4]
}
start_vertex = 0
print("BFS Traversal:")
bfs(graph, start_vertex)

#2.DFS QUESTION

def dfs(g, start, visited=None):
    if visited is None:
        visited = set()

    visited.add(start)
    print(start)

    for neighbor in g [start]:
        if neighbor not in visited:
            dfs(g, neighbor, visited)

g= {
  'A':['B','S'],
  'B':['A'],
  'S':['A','C','G'],
  'C':['D','E','F','S'],
  'D':['C'],
  'E':['C','H'],
  'F':['C','G'],
  'G':['S','F','H'],
  'H':['E','G']
}

starting_vertex = 'A'
print("\n DFS TRAVERSAL:")

dfs(g, starting_vertex)


#3.QUESTION(Puzzle Diagram)
from copy import deepcopy
import numpy as np
import time

def bestsolution(state):
    bestsol = np.array([], int).reshape(-1, 9)
    count = len(state) - 1
    while count != -1:
        bestsol = np.insert(bestsol, 0, state[count]['puzzle'], 0)
        count = (state[count]['parent'])
    return bestsol.reshape(-1, 3, 3)


def all(checkarray):
    set=[]
    for it in set:
        for checkarray in it:
            return 1
        else:
            return 0


def misplaced_tiles(puzzle,goal):
    mscost = np.sum(puzzle != goal) - 1
    return mscost if mscost > 0 else 0


def coordinates(puzzle):
    pos = np.array(range(9))
    for p, q in enumerate(puzzle):
        pos[q] = p
    return pos


def evaluvate_misplaced(puzzle, goal):
    steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8],  3),
('left', [0, 3, 6], -1),('right', [2, 5, 8],  1)],
                dtype =  [('move',  str, 1),('position', list),
('head', int)])
```

```
    dtstate = [('puzzle',  list),('parent', int),('gn',  int),('hn',
int)]

    costg = coordinates(goal)
    parent = -1
    gn = 0
    hn = misplaced_tiles(coordinates(puzzle), costg)
    state = np.array([(puzzle, parent, gn, hn)], dtstate)

    dtpriority = [('position', int),('fn', int)]

    priority = np.array([(0, hn)], dtpriority)

    while 1:
        priority = np.sort(priority, kind='mergesort', order=['fn',
'position'])
        position, fn = priority[0]
        # sort priority queue using merge sort,the first element is
picked for exploring.
        priority = np.delete(priority, 0, 0)
        puzzle, parent, gn, hn = state[position]
        puzzle = np.array(puzzle)

        blank = int(np.where(puzzle == 0)[0])

        gn = gn + 1
        c = 1
        start_time = time.time()
        for s in steps:
            c = c + 1
            if blank not in s['position']:
                openstates = deepcopy(puzzle)
                openstates[blank], openstates[blank + s['head']] =
openstates[blank + s['head']], openstates[blank]

                if ~(np.all(list(state['puzzle']) == openstates,
1)).any():
                    end_time = time.time()
                    if (( end_time - start_time ) > 2):
                        print(" The 8 puzzle is unsolvable \n")
                        break

                    hn = misplaced_tiles(coordinates(openstates),
costg)
                    q = np.array([(openstates, position, gn, hn)],
dtstate)
                    state = np.append(state, q, 0)
                    fn = gn + hn

                    q = np.array([(len(state) - 1, fn)], dtpriority)
                    priority = np.append(priority, q, 0)

                    if np.array_equal(openstates, goal):
                        print(' The 8 puzzle is solvable \n')
                        return state, len(priority)

    return state, len(priority)


puzzle = []

puzzle.append(2)
puzzle.append(8)
puzzle.append(3)
puzzle.append(1)
puzzle.append(6)
puzzle.append(4)
puzzle.append(7)
puzzle.append(0)
puzzle.append(5)


goal = []

goal.append(1)
goal.append(2)
goal.append(3)
goal.append(8)
goal.append(0)
goal.append(4)
goal.append(7)
goal.append(6)
goal.append(5)


state, visited = evaluvate_misplaced(puzzle, goal)
bestpath = bestsolution(state)
print(str(bestpath).replace('[', ' ').replace(']', ''))
totalmoves = len(bestpath) - 1
print('\nSteps to reach goal:',totalmoves)
visit = len(state) - visited
print('Total nodes visited: ',visit,"\n")
```

```
BFS Traversal:
0 1 3 2 4 5
 DFS TRAVERSAL:
A
B
S
C
```

```
D
E
H
G
F
 The 8 puzzle is solvable

   2 8 3
   1 6 4
   7 0 5

   2 8 3
   1 0 4
   7 6 5

   2 0 3
   1 8 4
   7 6 5

   0 2 3
   1 8 4
   7 6 5

   1 2 3
   0 8 4
   7 6 5

   1 2 3
   8 0 4
   7 6 5

Steps to reach goal: 5
Total nodes visited:  6
```