In [1]:

```
# BFS
graph = {'A':['B', 'E', 'C'],
         'B':['A', 'D', 'E'],
         'D':['B', 'E'],
         'E':['A', 'D', 'B'],
         'C':['A', 'F', 'G'],
         'F':['C'],
         'G':['C']
         }
visited = []
queue = []

def bfs(visited, graph, start_node, goal_node):
    visited.append(start_node)
    queue.append(start_node)
    while queue:
        m = queue.pop(0)
        print(m)
        if m == goal_node:
            print("Node is Found !!! ")
            break
        else:
            for n in graph[m]:
                if n not in visited:
                    visited.append(n)
                    queue.append(n)

print("The BFS Traversal is : ")
bfs(visited, graph, 'A', 'D')

#DFS
graph = {'A':['B', 'E', 'C'],
         'B':['A', 'D', 'E'],
         'D':['B', 'E'],
         'E':['A', 'D', 'B'],
         'C':['A', 'F', 'G'],
         'F':['C'],
         'G':['C']
         }

visited = []
stack = []
def dfs(graph, start, goal):
    print("DFS traveral is: ")
    stack.append(start)
    visited.append(start)
    while stack:
        node = stack[-1]
        stack.pop()
        print("Node: ", node)
        if node == goal:
            print("Goal node found!")
            return
        for n in graph[node]:
            if n not in visited:
                visited.append(n)
                stack.append(n)
```

```
dfs(graph, 'A', "D")
```

```
The BFS Traversal is :
A
B
E
C
D
Node is Found !!!
DFS traveral is:
Node:  A
Node:  C
Node:  G
Node:  F
Node:  E
Node:  D
Goal node found!
```

In [ ]:

In [2]:

```
import copy
final = [[1,2,3],[4,5,6],[7,8,-1]]
initial = [[1,2,3],[-1,4,6],[7,5,8]]
#function to find heuristic cost
def gn(state, finalstate):
        count = 0
        for i in range(3):
                for j in range(3):
                        if(state[i][j]!=-1):
                                if(state[i][j] != finalstate[i][j]):
                                        count+=1
        return count
def findposofblank(state):
        for i in range(3):
                for j in range(3):
                        if(state[i][j] == -1):
                                return [i,j]
def move_left(state, pos):
        if(pos[1]==0):
                return None
        retarr = copy.deepcopy(state)
        retarr[pos[0]][pos[1]],retarr[pos[0]][pos[1]-1] = retarr[pos[0]][pos[1]-
        return retarr
def move_up(state, pos):
        if(pos[0]==0):
                return None
        retarr = copy.deepcopy(state)
        #for i in state:
                #retarr.append(i)
        retarr[pos[0]][pos[1]],retarr[pos[0]-1][pos[1]] = retarr[pos[0]-1][pos[1
        return retarr
def move_right(state, pos):
        if(pos[1]==2):
                return None
        retarr = copy.deepcopy(state)
        #for i in state:
                #retarr.append(i)
        retarr[pos[0]][pos[1]],retarr[pos[0]][pos[1]+1] = retarr[pos[0]][pos[1]+
        return retarr
def move_down(state, pos):
        if(pos[0]==2):
                return None
        retarr = copy.deepcopy(state)
        retarr[pos[0]][pos[1]],retarr[pos[0]+1][pos[1]] = retarr[pos[0]+1][pos[1
        return retarr
def printMatrix(matricesArray):
        print("")
        counter = 1
        for matrix in matricesArray:
                print("Step {}".format(counter))
                for row in matrix:
                        print(row)
                counter+=1
                print("")
def eightPuzzle(initialstate, finalstate):
        hn=0
        explored = []
        while(True):
```

```
                explored.append(initialstate)
                if(initialstate == finalstate):
                        break
                hn+=1
                left = move_left(initialstate, findposofblank(initialstate))
                right = move_right(initialstate, findposofblank(initialstate))
                up = move_up(initialstate, findposofblank(initialstate))
                down = move_down(initialstate, findposofblank(initialstate))
                fnl=1000
                fnr=1000
                fnu=1000
                fnd=1000
                if(left!=None):
                        fnl = hn + gn(left,finalstate)
                if(right!=None):
                        fnr = hn + gn(right,finalstate)
                if(up!=None):
                        fnu = hn + gn(up,finalstate)
                if(down!=None):
                        fnd = hn + gn(down,finalstate)
                minfn = min(fnl, fnr, fnu, fnd)
                if((fnl == minfn) and (left not in explored)):
                        initialstate = left
                elif((fnr == minfn) and (right not in explored)):
                        initialstate = right
                elif((fnu == minfn) and (up not in explored)):
                        initialstate = up
                elif((fnd == minfn) and (down not in explored)):
                        initialstate = down
        printMatrix(explored)
#eightPuzzle(initial, final)
def main():
        while(True):
                ch = int(input("PRESS 1 to continue and 0 to Exit : "))
                if(not ch):
                        break
                start = []
                print("START STATE\n")
                for i in range(3):
                        arr=[]
                        for j in range(3):
                                a = int(input("Enter element at  {},{}: ".format
                                arr.append(a)
                        start.append(arr)
                final = []
                print("FINAL STATE\n")
                for i in range(3):
                        arr=[]
                        for j in range(3):
                                a = int(input("Enter element at  {},{}: ".format
                                arr.append(a)
                        final.append(arr)
                eightPuzzle(start, final)
main()
```

```
START STATE

FINAL STATE
```

```
Step 1
[1, 2, 3]
[-1, 4, 6]
[7, 5, 8]

Step 2
[1, 2, 3]
[4, -1, 6]
[7, 5, 8]

Step 3
[1, 2, 3]
[4, 5, 6]
[7, -1, 8]

Step 4
[1, 2, 3]
[4, 5, 6]
[7, 8, -1]
```

In [8]:
```
def selectionSort(array, size):
    for ind in range(size):
        min_index = ind
        for j in range(ind + 1, size):
            if array[j] < array[min_index]:
                min_index = j
        array[ind], array[min_index] = array[min_index], array[i

arr = [-2, 45, 0, 11, -9, 88, -97, -202, 747]
size = len(arr)
selectionSort(arr, size)

print("The array after sorting in ascending order by selection s
print(arr)
```

```
The array after sorting in ascending order by selection sort is:
[-202, -97, -9, -2, 0, 11, 45, 88, 747]
```

In [ ]:

In [1]:

```
# Number of queens
n=4
# Matrix
a=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
# Dictionary for backtrack
b={}

# Checking if column is safe
def isColumnSafe(r,c):
    while(r>=0):
        if(a[r][c] == 1):
            return 0
        r = r-1
    return 1

# Checking if left diagonal is safe
def isLeftDiagonalSafe(r,c):
    while(r>=0 and c>=0):
        if(a[r][c] == 1):
            return 0
        r = r-1
        c = c-1
    return 1

# Checking if right diagonal is safe
def isRightDiagonalSafe(r,c):
    while(r>=0 and c<n):
        if(a[r][c]==1):
            return 0
        r = r-1
        c = c+1
    return 1

def isSafe(r,c):
    if(isColumnSafe(r,c) and isLeftDiagonalSafe(r,c) and isRightDiagonalSafe(r,c
        return True
    return False
def chessboard(r,c):
    if(r>=n):
        return
    p = 0
    while c<n:
        p = isSafe(r,c)
        if p == 1:
            a[r][c] = 1
            b.update({r:c})
            break
        c=c+1

    if p==1:
        chessboard(r+1,0)
    else:
        a[r-1][b.get(r-1)]=0
        chessboard(r-1,int(b.get(r-1))+1)
chessboard(0,0)
print("Matrix is:- ",a)
print("Dictionary is:- ",b)
```

```
Matrix is:-  [[0, 1, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 0]]
Dictionary is:-  {0: 1, 1: 3, 2: 0, 3: 2}
```

In [1]:

```
import nltk
from nltk.chat.util import Chat, reflections

pairs=[
    #
    [
        r"my name is (.)",
        ["Hello %1, How are you"]
    ],
    # Or expression
    [
        r"Hi|Hello|Hey there|Hola",
        ["Hello my name is Hiesenberg"]
    ],
    [
        r"what is your name ?",
        ["I am a bot created by Heisenbergwhat. you can call me crazy!",]
    ],
    [
        r"how are you ?",
        ["I'm doing good How about You ?",]
    ],
    [
        r"sorry (.*)",
        ["Its alright","Its OK, never mind",]
    ],
    [
        r"I am fine",
        ["Great to hear that, How can I help you?",]
    ],
    [
        r"I (.*) good",
        ["Nice to hear that","How can I help you?:)",]
    ],
    [
        r"(.*) age?",
        ["I'm a computer program dude Seriously you are asking me this?",]
    ],
    [
        r"what (.*) want ?",
        ["Make me an offer I can't refuse",]
    ],
    [
        r"(.*) created ?",
        ["Raghav created me using Python's NLTK library ","top secret ;)",]
    ],
    [
        r"(.*) (location|city) ?",
        ['Indore, Madhya Pradesh',]
    ],
    [
        r"how is weather in (.*)?",
        ["Weather in %1 is awesome like always","Too hot man here in %1","Too co
    ],
    [
        r"i work in (.*)?",
        ["%1 is an Amazing company, I have heard about it. But they are in huge
    ],
```

```
    [
        r"(.)raining in (.)",
        ["No rain since last week here in %2","Damn its raining too much here in
    ],
    [
        r"how (.) health(.)",
        ["I'm a computer program, so I'm always healthy ",]
    ],
    [
        r"(.*) (sports|game) ?",
        ["I'm a very big fan of Football",]
    ],
    [
        r"who (.*) sportsperson ?",
        ["Messy","Ronaldo","Roony"]
    ],
    [
        r"who (.*) (moviestar|actor)?",
        ["Brad Pitt"]
    ],
    [
        r"i am looking for online guides and courses to learn data science, can
        ["Crazy_Tech has many great articles with each step explanation along wi
    ],
    [
        r"quit",
        ["Thank you for using our intelligence services"]
    ],


]

def chat():
    print("Hey there! I am Heisenberg at your service")
    chat = Chat(pairs)
    chat.converse()

chat()
```

```
Hey there! I am Heisenberg at your service
Hello my name is Hiesenberg
I'm doing good How about You ?
How can I help you?:)
None
Thank you for using our intelligence services
```

In [ ]: