```
#ENPM661 Spring 2023
#Brendan Neal
#Project 1

#Goal: Solve 8 Piece Puzzle using BFS

#Note: I am using lists and arrays because that is what I am most comfortable with since
new to Python

import numpy as np
import copy

##-----------Initializing Data Structures----------------##
Node_State_i = [] #Node State Matrix (3x3)
Node_Index_i = [] #node_i index
Parent_Node_Index_i = [] #parent node i index
Node_Index_i.append(1) #Starting first node index
Parent_Node_Index_i.append(0)

Unexplored_Nodes = [] #Nodes that have not been explored yet. THIS IS MY OPEN LIST!!!!
Explored_Nodes = [] #Nodes that have been visited already. THIS IS MY CLOSED LIST!!!!
Discovered_Nodes = [] #Nodes discovered after blank tile moves
New_Node = [] #After shifting node for saving into Discoverd Nodes
Backtrack_Path = [] #Save Backtrack Path


##--Note: I used lists all the way through, since the array data structure and indexing
confusing to me.--#


##---------------Initializing Known Test States-----------##
#Comment in and out as Needed

# Known Test State 1
#Initial_State = [1, 6, 7, 2, 0, 5, 4, 3, 8] #WORKS
#Goal_State = [1, 4, 7, 2, 5, 8, 3, 0, 6] #WORKS


#Known Test State 2
#Initial_State = [4, 7, 8, 2, 1, 5, 3, 6, 0] #WORKS
#Goal_State = [1, 4, 7, 2, 5, 8, 3, 6, 0] #WORKS


##---------Defining Find Blank Tile Function-------------##

def FindBlankTileList(InitialStateList):
    blank_tile = InitialStateList.index(0)
    return blank_tile

##----------------Defining Actions-----------------------##

##-------------------------------------------------------------------------------------
##Note: I included the logic of "If Possible" in my search function since I understood i
better from a logic standpoint
##-------------------------------------------------------------------------------------
```

```
#To move right, you move "forward" in the list 1 slot
def ActionMoveRight(CurrentNode):
    Newnode = CurrentNode.copy()
    position = Newnode.index(0)
    temp = Newnode[position]
    Newnode[position] = Newnode[position + 1]
    Newnode[position + 1] = temp
    return Newnode

#To move left, you move "backward" in the list 1 slot
def ActionMoveLeft(CurrentNode):
    Newnode = CurrentNode.copy()
    position = Newnode.index(0)
    temp = Newnode[position]
    Newnode[position] = Newnode[position-1]
    Newnode[position - 1] = temp
    return Newnode

#To move down, you move "forward" in the list 3 slots. For example, list position 2 [0,1
moves to list position 5 [1,1]
def ActionMoveDown(CurrentNode):
    Newnode = CurrentNode.copy()
    position = Newnode.index(0)
    temp = Newnode[position]
    Newnode[position] = Newnode[position + 3]
    Newnode[position + 3] = temp
    return Newnode

#To move down, you move "backward" in the list 3 slots. For example, list position 7 [2,
moves to list position 4 [1,0]
def ActionMoveUp(CurrentNode):
    Newnode = CurrentNode.copy()
    position = Newnode.index(0)
    temp = Newnode[position]
    Newnode[position] = Newnode[position - 3]
    Newnode[position - 3] = temp
    return Newnode


##--------------------------------------Defining Search
Function--------------------------------------##
def BFS_Search(Current_Node, ListOfExploredNodes):

    placeholder = copy.deepcopy(Current_Node) #I have to use deepcopy here because I do
want to change the original object. This helps in the searching process.
    BlankTilePos = FindBlankTileList(placeholder) #Use function to fild blank tile.

    if(BlankTilePos == 0):
        node_right = ActionMoveRight(placeholder) #only options are to move right or dow
        node_below = ActionMoveDown(placeholder)
        if node_right not in ListOfExploredNodes:
            New_Node.append(node_right)             #If statements check if node already
exists.
        if node_below not in ListOfExploredNodes:
            New_Node.append(node_below)
```

```
        return New_Node

    if(BlankTilePos == 1):
        node_right = ActionMoveRight(placeholder)
        node_left = ActionMoveLeft(placeholder)
        node_below = ActionMoveDown(placeholder)
        if node_left not in ListOfExploredNodes:
            New_Node.append(node_left) #Options are to move right, left, or down.
        if node_right not in ListOfExploredNodes:
            New_Node.append(node_right)             #If statements check if node already
exists.
        if node_below not in ListOfExploredNodes:
            New_Node.append(node_below)
        return New_Node

    if(BlankTilePos == 2):
        node_below = ActionMoveDown(placeholder)
        node_left = ActionMoveLeft(placeholder)
        if node_below not in ListOfExploredNodes:
            New_Node.append(node_below) #Options are to move left or down.
        if node_left not in ListOfExploredNodes:             #If statements check if node
already exists.
            New_Node.append(node_left)
        return New_Node

    if(BlankTilePos == 3):
        node_right = ActionMoveRight(placeholder)
        node_above = ActionMoveUp(placeholder)
        node_below = ActionMoveDown(placeholder)
        if node_right not in ListOfExploredNodes:
            New_Node.append(node_right) #options are to move right, up, or down
        if node_above not in ListOfExploredNodes:
            New_Node.append(node_above)             #If statements check if node already
exists.
        if node_below not in ListOfExploredNodes:
            New_Node.append(node_below)
        return New_Node


    if(BlankTilePos == 4):
        node_right = ActionMoveRight(placeholder)
        node_above = ActionMoveUp(placeholder)
        node_below = ActionMoveDown(placeholder) #options are to move in any direction.
        node_left = ActionMoveLeft(placeholder)
        if node_right not in ListOfExploredNodes:
            New_Node.append(node_right)
        if node_above not in ListOfExploredNodes:
            New_Node.append(node_above)             #If statements check if node already
exists.
        if node_below not in ListOfExploredNodes:
            New_Node.append(node_below)
        if node_left not in ListOfExploredNodes:
            New_Node.append(node_left)
        return New_Node

    if(BlankTilePos == 5):
```

```
        node_left = ActionMoveLeft(placeholder)
        node_above = ActionMoveUp(placeholder)
        node_below = ActionMoveDown(placeholder) #options are to move left, up, or down.
        if node_left not in ListOfExploredNodes:
            New_Node.append(node_left)
        if node_above not in ListOfExploredNodes:
            New_Node.append(node_above)             #If statements check if node already
exists.
        if node_below not in ListOfExploredNodes:
            New_Node.append(node_below)
        return New_Node

    if(BlankTilePos == 6):
        node_right = ActionMoveRight(placeholder)
        node_above = ActionMoveUp(placeholder)
        if node_right not in ListOfExploredNodes:
            New_Node.append(node_right) #options are to move right or up.
        if node_above not in ListOfExploredNodes:
            New_Node.append(node_above)             #If statements check if node already
exists.
        return New_Node

    if(BlankTilePos == 7):
        node_right = ActionMoveRight(placeholder)
        node_above = ActionMoveUp(placeholder)
        node_left = ActionMoveLeft(placeholder) #options are to move up, left, or right.
        if node_right not in ListOfExploredNodes:
            New_Node.append(node_right)
        if node_above not in ListOfExploredNodes:
            New_Node.append(node_above)             #If statements check if node already
exists.
        if node_left not in ListOfExploredNodes:
            New_Node.append(node_left)
        return New_Node

    if(BlankTilePos == 8):
        node_left = ActionMoveLeft(placeholder)
        node_above = ActionMoveUp(placeholder) #options are to move up or left.
        if node_left not in ListOfExploredNodes:
            New_Node.append(node_left)             #If statements check if node already
exists.
        if node_above not in ListOfExploredNodes:
            New_Node.append(node_above)
        return New_Node


##------------------------Define Path Generator-----------------##

def path_generator(start_state, searchresults, taken_path):
    global Parent_Node_Index_i #need to set to global variables in order to access them
inside the function without including them as inputs.
    global Node_Index_i

    path_temp = [] #initialize temporary path
    path_temp.append(searchresults) #append all the search results.
    next_idx = 2
```

```
    for i in range(len(taken_path)):
        Node_Index_i.append(next_idx)

        for j in range(len(taken_path)): #For all the nodes in the taken path
            if path_temp[i] == taken_path[j][0]: #If the temporary path at this i index
matches the taken path at this j index, append the taken path to the temp.
                path_temp.append(taken_path[j][1])

                last_parent=Parent_Node_Index_i[-1] #Also, index the previous parent. It
backwards search after all.

                if taken_path[j][1] != taken_path[j-1][1]: #Increment parent index if th
a new parent.
                    Parent_Node_Index_i.append(last_parent+1)
                else:
                    Parent_Node_Index_i.append(last_parent)
                break

        if path_temp[i] == start_state: #Determine if the current parent is start, and b
the loop if it is
            break
        next_idx = next_idx + 1 #Next index for loop

    path = []

    for i in reversed(path_temp): #Reverse the saved path because currently it is still
backwards
        path.append(i)

    return path

def GetInitialState(): #Pull data from the terminal to be sorted into lists.
    print("Enter values for Initial Column 1, separated by spaces and hit enter: ")
    row1=[int(x) for x in input().split()]
    print("Enter values for Initial Column 2, separated by spaces and hit enter: ")
    row2=[int(x) for x in input().split()]
    print("Enter values for Initial Column 3, separated by spaces and hit enter: ")
    row3=[int(x) for x in input().split()]
    InitState = row1 + row2 + row3
    return InitState

def GetGoalState(): #Pull data from the terminal to be sorted into lists.
    print("Enter values for Goal Column 1, separated by spaces and hit enter: ")
    row1=[int(x) for x in input().split()]
    print("Enter values for Goal Column, separated by spaces and hit enter: ")
    row2=[int(x) for x in input().split()]
    print("Enter values for Goal Column 3, separated by spaces and hit enter: ")
    row3=[int(x) for x in input().split()]
    GoalState= row1 + row2 + row3
    return GoalState

##------------------Define File Creation Function-------------------##
def FileMaker(Explored, Path, Parent_Index, Node_Index):

    Doc1 = open('Nodes.txt', 'w')
```

```
    for explore in range(len(Explored)):
        for e in Explored[explore]:
            TempWrite = ' '.join(map(str, Explored[explore])) #Deleting brackets and com
from each of the lists and converting contents to string.
        Doc1.write(TempWrite)
        Doc1.write("\n")
    Doc1.close()

    Doc2 = open('NodesInfo.txt', 'w')
    Doc2.write("Node_Index\tParent_Node_Index\tNode\n")
    for row in range(len(Path)):
        Doc2.write(str(Node_Index[row]))
        Doc2.write("\t\t\t")
        Doc2.write(str(Parent_Index[row]))
        Doc2.write("\t\t\t\t\t")
        for element in Explored[row]:
            StepWrite = ' '.join(map(str, Explored[row])) #Deleting brackets and commas
each of the lists and converting contents to string.
        Doc2.write(StepWrite)
        Doc2.write("\n")
    Doc2.close()

    Doc3 = open('nodePath.txt', 'w')
    for move in range(len(Path)):
        for e in Path[move]:
            ToWrite = ' '.join(map(str,Path[move])) #Deleting brackets and commas from e
of the lists and converting contents to string.
        Doc3.write(ToWrite)
        Doc3.write("\n")
    Doc3.close()




##------------"Main" Program Script-----------------##

Initial_State = GetInitialState() #Pull initial and goal states
Goal_State = GetGoalState()
print("Starting at Initial State:", Initial_State)
Unexplored_Nodes.append(Initial_State) #append the initial state to the queue.
Explored_Nodes.append(Initial_State) #append the first state to explored in order to ens
the initial state is not reexplored. I index next element later.


while(Unexplored_Nodes):
    Node_State_i = Unexplored_Nodes.pop(0) #pop the initial state on first iteration, po
first value onward.

    if np.array_equal(Node_State_i, Goal_State): #Check to see if you've reached the goa
state.
        print("Goal State Reached!")
        results = Node_State_i
        break
```

```
    Discovered_Nodes = BFS_Search(Node_State_i, Explored_Nodes) #Perform BFS search sta
from the current node iteration.


    for nodes in Discovered_Nodes: #Save child and parent nodes to be used in the backt
path. Is for all discovered nodes.
        Child_and_Parent = []
        Child_and_Parent.append(nodes)
        Child_and_Parent.append(Node_State_i)
        Backtrack_Path.append(Child_and_Parent)

    for branch in Discovered_Nodes: #DOUBLE CHECK if nodes discovered have been explore
already or not. Only add to explored nodes if new.
        if branch not in Explored_Nodes: #only if the discovered node has not been expl
already
            Explored_Nodes.append(branch)
            Unexplored_Nodes.append(branch) #loop through the process again.
    print("Currently Searching, Explored Node Count:", len(Explored_Nodes), "Unexplored
Count:", len(Unexplored_Nodes))

    Discovered_Nodes.clear() #Clear the BFS discovered nodes from the list and start lo
again.

print("Searching Complete")

Solved_Path = path_generator(Initial_State, results, Backtrack_Path)

print("Path is:", Solved_Path)


print("Generating Text Files")

FileMaker(Explored_Nodes, Solved_Path, Parent_Node_Index_i, Node_Index_i)



print("Done!")
```