**ASSIGNMENT-1: DFS, BFS**

1:- final state could be reached from the initial state by both the algos BFS and DFS .

**BFS algo :-**

Number of state visited :- 130041

Depth of the target node :- 24

Time taken with 36 Gb ram and M3 processor :- 520 secs = ~ 9 mins

**Depth limited DFS algo with limiting depth as 27:-**

Number of state visited :- 20063

Depth of the target node :- 26

Time taken with 36 Gb ram and M3 processor :- 22 sec.

BFS algo search all the nodes in the current depth before going to next so it visit more nodes. But in this way it guarantees shortest path.

DFS visit all the depth first before visiting the node in the same depth . That’s why it reached the solution first, if we have a limit over the depth it can go. Without depth limit DFS could go into infinite loop and might never able to get to the solution. Shortest path is not guaranteed by DFS.

We can see BFS took way more time and visits way more node to reach the final state compare to depth limited DFS .

And the path found in DFS is at 26th depth which is not the shortest path as BFS found it on 24th depth which is shortest path to reach the solution.

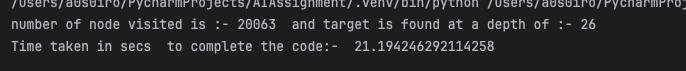
2:- We can clearly see DFS is faster than BFS as DFS doesn’t search all the nodes in the current depth before moving to more deep in the graph.

So number of state visited is less and time taken is less but Shortest path is not guaranteed in DFS .

BFS even though is slow guarantees shortest path as it visit all the node in current node before moving forward.

I have attached the code (.py files) with this assignment with proper comment and output

DFS output:-



BFS output:-



BFS code :-

import copy  
from collections import deque  
import time  
  
start = time.time() # Inital timestamp  
  
initial\_state=[[3,2,1],[4,5,6],[8,7,0]] # Initial state with 0 as blank space  
final\_state=[[1,2,3],[4,5,6],[7,8,0]] # target state with 0 as blank space  
  
visited=[] # List to store all the visited state  
q = deque() # Queue is the data structure to be used for DFS  
  
moves=[[0,-1],[0,1],[1,0],[-1,0]] # possible moves  
  
def checkValidePath(i,j):  
 if i<3 and i>=0 and j<3 and j>=0: # check if we are not going out of the index  
 return True  
 else:  
 return False  
  
def createChild(state,i,j): # returns child nodes for a given state and position of blank space  
 # i an j is the location of blank space("0")  
 childrens=[]  
  
 for move in moves: # iterate over all the possible move in moves list.  
 if checkValidePath(i+move[0],j+move[1]):  
 temp\_state=copy.deepcopy(state)  
 temp\_state[i][j],temp\_state[i+move[0]][j+move[1]]=temp\_state[i+move[0]][j+move[1]],temp\_state[i][j]  
 if temp\_state not in visited:  
 childrens.append(temp\_state)  
 return childrens  
  
  
def BFS(random\_state,final\_state): # DFS function  
 q.append(random\_state) # Initial state to be visited  
 depth =0 # initial depth is 0  
 if random\_state ==final\_state: # check if initial state is the target  
 return depth  
 else:  
 visited.append(random\_state) # mark initial state as visited  
  
 while(len(q) and depth<50):  
 l=len(q)  
 depth+=1 # Increase depth by one as we are visiting the child of the previous state  
  
 for \_ in range(l): # Iterate l times to visit all the nodes from current depth  
 current\_state =q.popleft() # visit the node  
 for i in range(len(current\_state)):  
 for j in range(len(current\_state[0])):  
 if current\_state[i][j] == 0:  
 childerens = createChild(current\_state, i, j) # Find the position of blank space and get the child to be visited .  
 # visit the child node of all the visited states in previous depth level  
 for child in childerens:  
 if child not in visited:  
 print(child,depth) # Print the state/node currently being visited  
 if child==final\_state: # Return if we have reached the final state  
 return depth  
 else:  
 visited.append(child) # Add to the visited list  
 q.append(child) # Add to the queue , Child of these nodes will be visited in the next depth level.  
  
result\_depth=BFS(initial\_state,final\_state)  
  
end = time.time() # timestamp after th run is complete  
  
print("number of node visited is :-",len(visited)," and target is found at a depth of :-" , result\_depth) # print result  
print("Time taken in secs to complete the code:- ",end-start)

DFS CODE:-

import copy  
import time  
  
  
start = time.time() # Inital timestamp  
  
initial\_state=[[3,2,1],[4,5,6],[8,7,0]] # Initial state with 0 as blank space  
final\_state=[[1,2,3],[4,5,6],[7,8,0]] # target state with 0 as blank space  
  
  
visited=[] # List to store all the visited state  
stack=[] # data structure to be used for DFS  
max\_depth=27 # maximum depth allowed for the DFS algo  
moves=[[0,-1],[0,1],[1,0],[-1,0]] # possible moves  
  
def checkValidePath(i,j): # check if we are not going out of the index  
 if i<3 and i>=0 and j<3 and j>=0:  
 return True  
 else:  
 return False  
  
def createChild(state,i,j): # returns child nodes for a given state and position of blank space  
 # i an j is the location of blank space("0")  
 childrens=[]  
  
 for move in moves: # iterate over all the possible move in moves list.  
 if checkValidePath(i+move[0],j+move[1]):  
 temp\_state=copy.deepcopy(state)  
 temp\_state[i][j],temp\_state[i+move[0]][j+move[1]]=temp\_state[i+move[0]][j+move[1]],temp\_state[i][j]  
 if temp\_state not in visited:  
 childrens.append(temp\_state)  
 return childrens  
  
  
def getBlankPos(state): # Returns the location of blank space in the given state  
 for i in range(len(state)):  
 for j in range(len(state[0])):  
 if state[i][j] == 0:  
 return [i,j]  
  
  
  
def DFS(random\_state,final\_state): # DFS function  
  
 stack.append([random\_state,0]) # Insert Initial state with initial depth 0  
  
 while(len(stack)):  
 current\_state,depth=stack.pop() # visited the most recent node  
 if depth<=max\_depth and current\_state not in visited:  
 if final\_state == current\_state: # check if we have reached target state , If yes return depth  
 return depth  
 visited.append(current\_state) # add to the visited after visiting the node  
 blanck\_i,blanck\_j=getBlankPos(current\_state) # get the position of the blank space  
 childerens = createChild(current\_state, blanck\_i, blanck\_j) # get a list of child node from current state which is not visted before  
 for child in childerens:  
 if child not in visited:  
 stack.append([child,depth+1]) # Add all the child node to visit in the stack after checking if we have visited it before  
  
  
result\_depth=DFS(initial\_state,final\_state) # Call the DFS function with initial and final state to get the depth at which we got the target state  
end = time.time() # timestamp after th run is complete  
  
print("number of node visited is :-",len(visited)," and target is found at a depth of :-" , result\_depth) # print result  
  
print("Time taken in secs to complete the code:- ",end-start)