# DEPARTMENT OF INFORMATION TECHNOLOGY

**COURSE CODE: DJ19ITL504 DATE:22/10/22**

# COURSE NAME: Artificial Intelligence Laboratory CLASS: TY-IT

**EXPERIMENT NO.03**

**CO/LO:** Formulate the problem as a state space and select appropriate technique from blind, heuristic or adversarial search to generate the solution.

**AIM / OBJECTIVE:** To Implement iterative deepening and depth limited search on 8-Puzzle Problem

# DESCRIPTION OF EXPERIMENT:

* We should generate the state space for above mentioned problem
* The traversal path for iterative deepening and depth limited search should be displayed
* compare iterative deepening and depth limited search wrt time, space complexities and completeness and optimality.

**Depth-Limited Search Algorithm:**

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

* Standard failure value: It indicates that problem does not have any solution.
* Cutoff failure value: It defines no solution for the problem within a **given depth limit.**

**Iterative deepening depth-first Search:**

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

# Explanation/Solutions(Design):

**Code:**

**statespace formulation:**

import copy

from collections import defaultdict

from treelib import Node, Tree

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

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

tree=defaultdict(list)

graph=defaultdict(list)

def locate(matrix):

    for i,j  in enumerate(matrix):

        if -1 in j:

            index=(i,j.index(-1))

            return index

def checkmoves(matrix):

    indx=locate(matrix)

    moves=["u","d","l","r"]

    if(indx[1]==0):

        moves.remove("l")

    if(indx[1]==2):

        moves.remove("r")

    if(indx[0]==0):

        moves.remove("u")

    if(indx[0]==2):

        moves.remove("d")

    return moves

def swap(matrix,i,f):

    temp=matrix[i[0]][i[1]]

    matrix[i[0]][i[1]]=matrix[f[0]][f[1]]

    matrix[f[0]][f[1]]=temp

def move(matrix,final):

    nextlvl=[]

    nextlvl.append(matrix)

    for m in nextlvl:

        p=0

        moves=checkmoves(m)

        indx=locate(m)

        nl=[]

        #print(m)

        for i in moves:

            current=copy.deepcopy(m)

            if i=="u":

                k=indx[0]-1

                j=indx[1]

            elif i=="d":

                k=indx[0]+1

                j=indx[1]

            elif i=="l":

                k=indx[0]

                j=indx[1]-1

            elif i=="r":

                k=indx[0]

                j=indx[1]+1

            swap(current,indx,(k,j))

            nl.append(current)

            if current not in nextlvl:

                nextlvl.append(current)

                #print(current)

                tree[nextlvl.index(m)].append(current)

                graph[nextlvl.index(m)].append(nextlvl.index(current))

        if final in nextlvl:

            return graph

def printstatespace(graph):

    #print(graph)

    print("\n\n",tree,"\n\n")

    #print(nextlvl)

    t=Tree()

    t.create\_node(0,0)

    for k in graph:

        for v in graph[k]:

            t.create\_node(v,v,parent=k)

    t.show()

def getdepth(graph):

    deapth={}

    deapth[0]=0

    for k in graph:

        for v in graph[k]:

            deapth[v]=deapth[k]+1

    return deapth

'''

    t=Tree()

    t.create\_node(0,nextlvl[0])

    for k in graph:

        for v in graph[k]:

            t.create\_node(v,nextlvl[v],parent=nextlvl[k])

    t.show()

'''

mxtr=move(initial,final)

printstatespace(mxtr)

**dls:**

import statespace

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

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

#print(s\_space)

visited = set()

graph=statespace.move(initial,final)

deapth=statespace.getdepth(graph)

#print(deapth)

def dfs(visited, graph, node,l):  #function for dfs

    if deapth[node]>l:

        return

    else:

        if node not in visited:

            print (node)

            visited.add(node)

            for neighbour in graph[node]:

                dfs(visited, graph, neighbour,l)

limit=2

print("traversal : ")

dfs(visited,graph,0,limit)

**ids:**

import dls

import sys

import statespace

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

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

graph=statespace.move(initial,final)

deapth=statespace.getdepth(graph)

limit=1

while(True):

    visited=set()

    print("traversal for limit : ",limit)

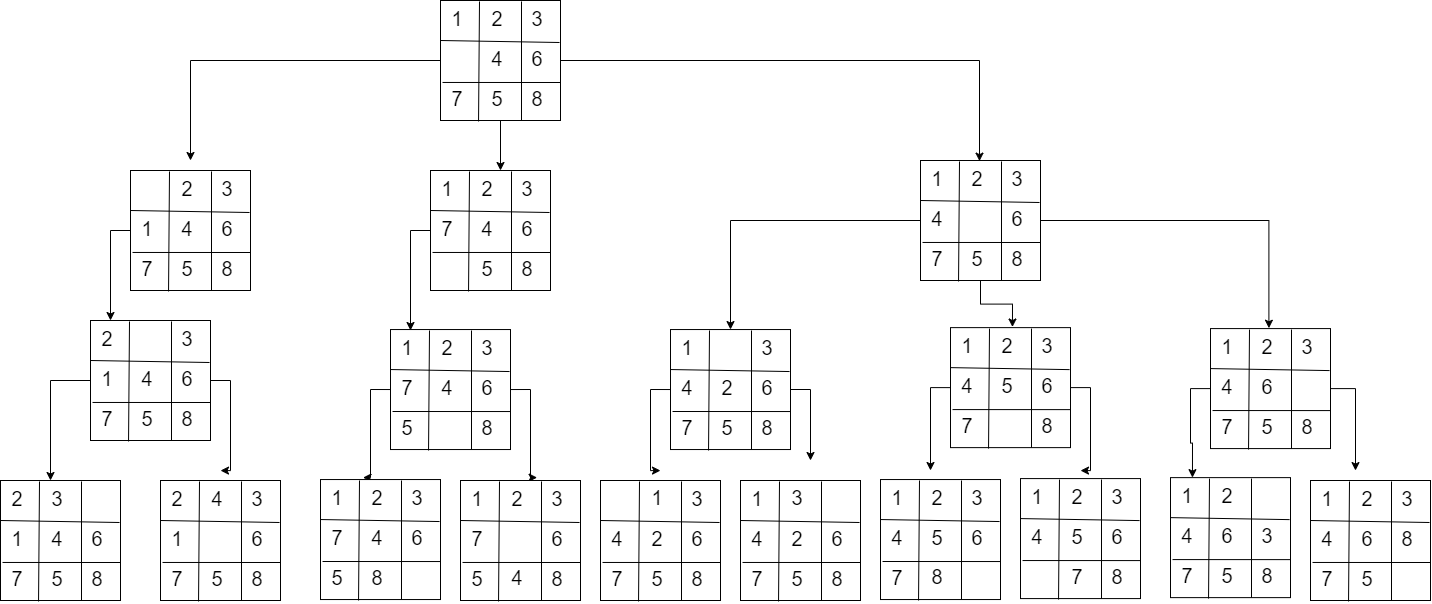
    dls.dfs(visited,graph,0,limit)

    limit=limit+1

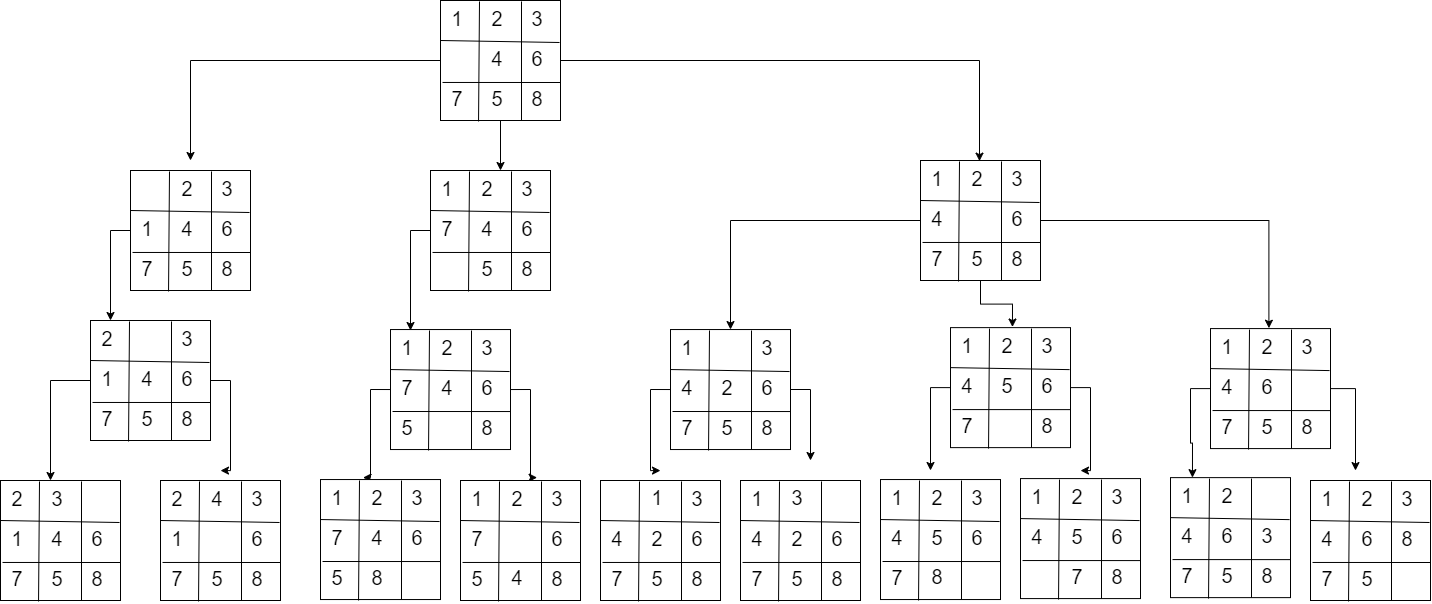
    if 16 in visited:

        sys.exit()

**example state space:**

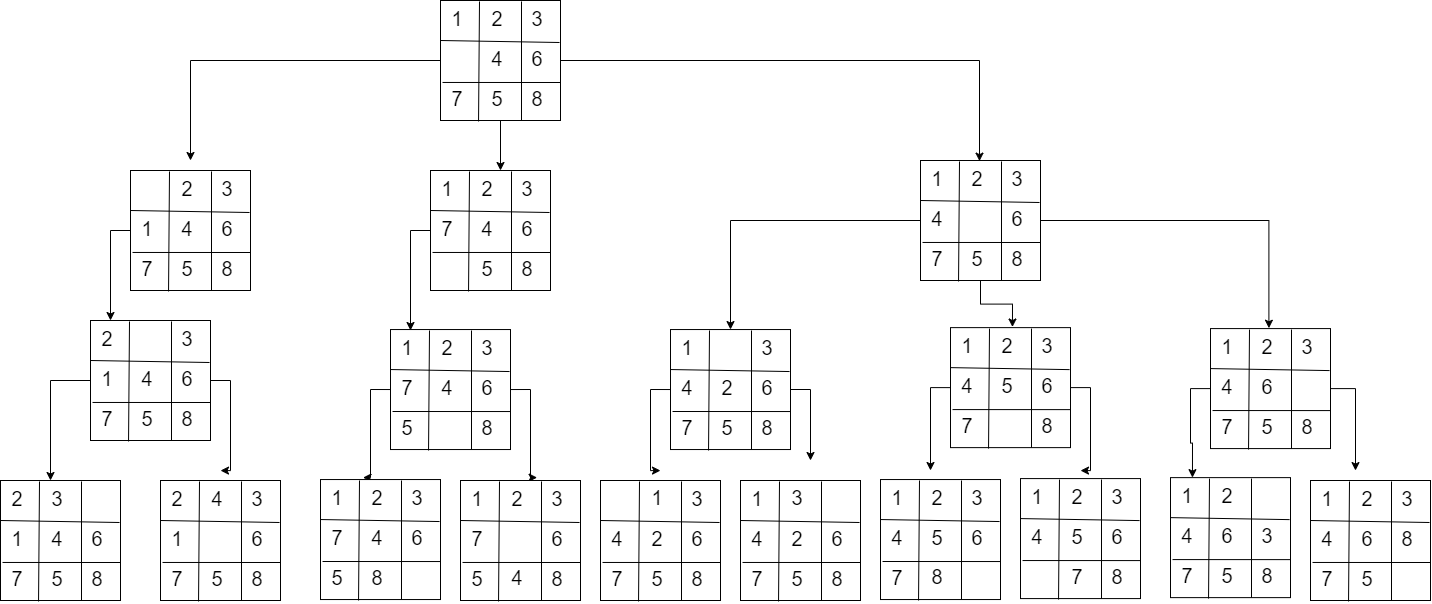
****

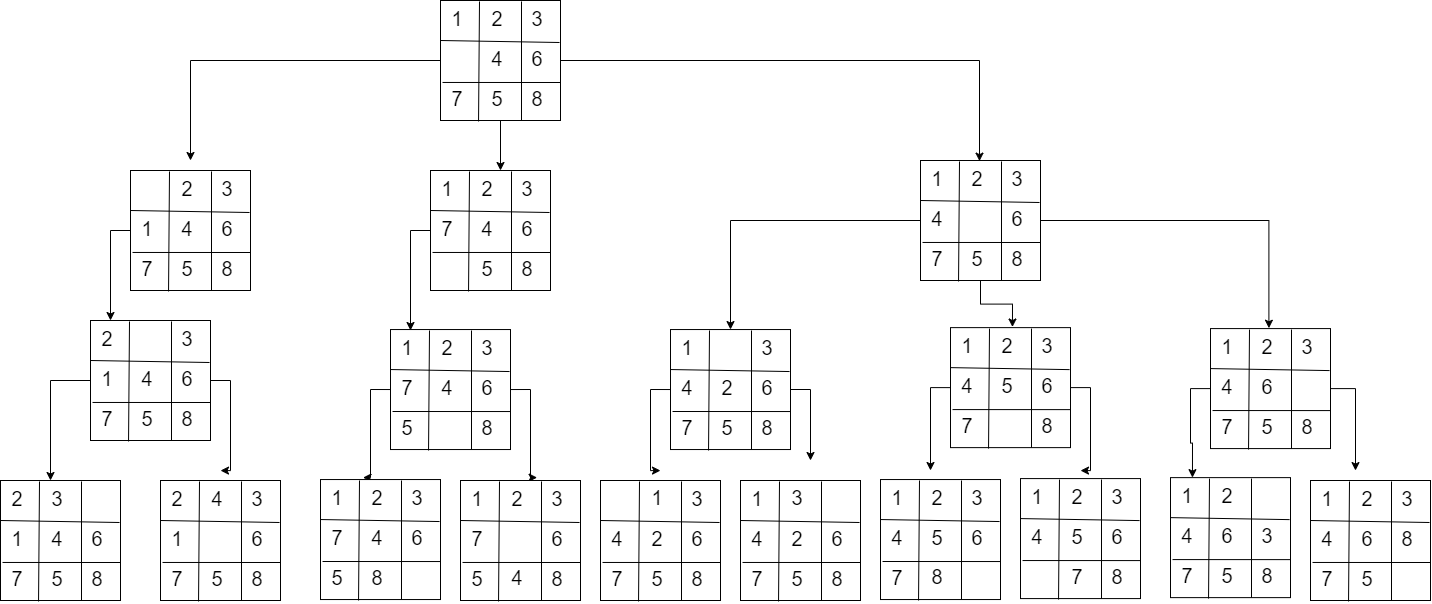
**Dls(limit =2)**

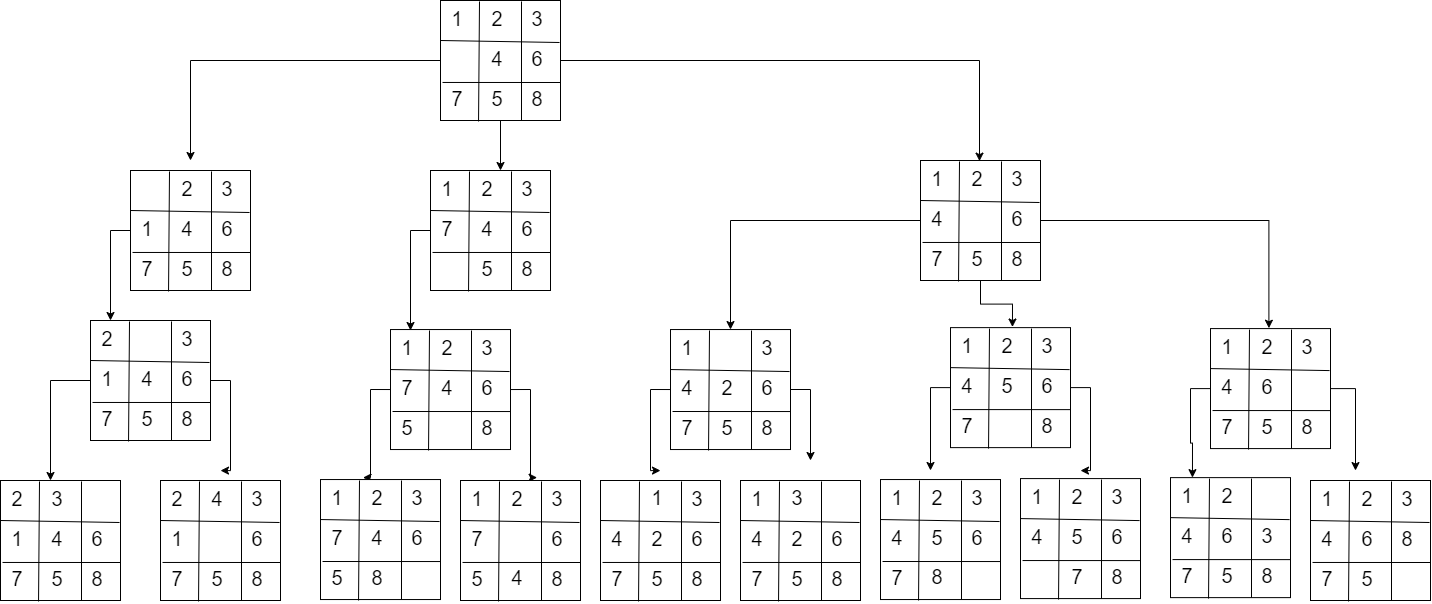
****

**Iddls:**

**Limit =1**

****

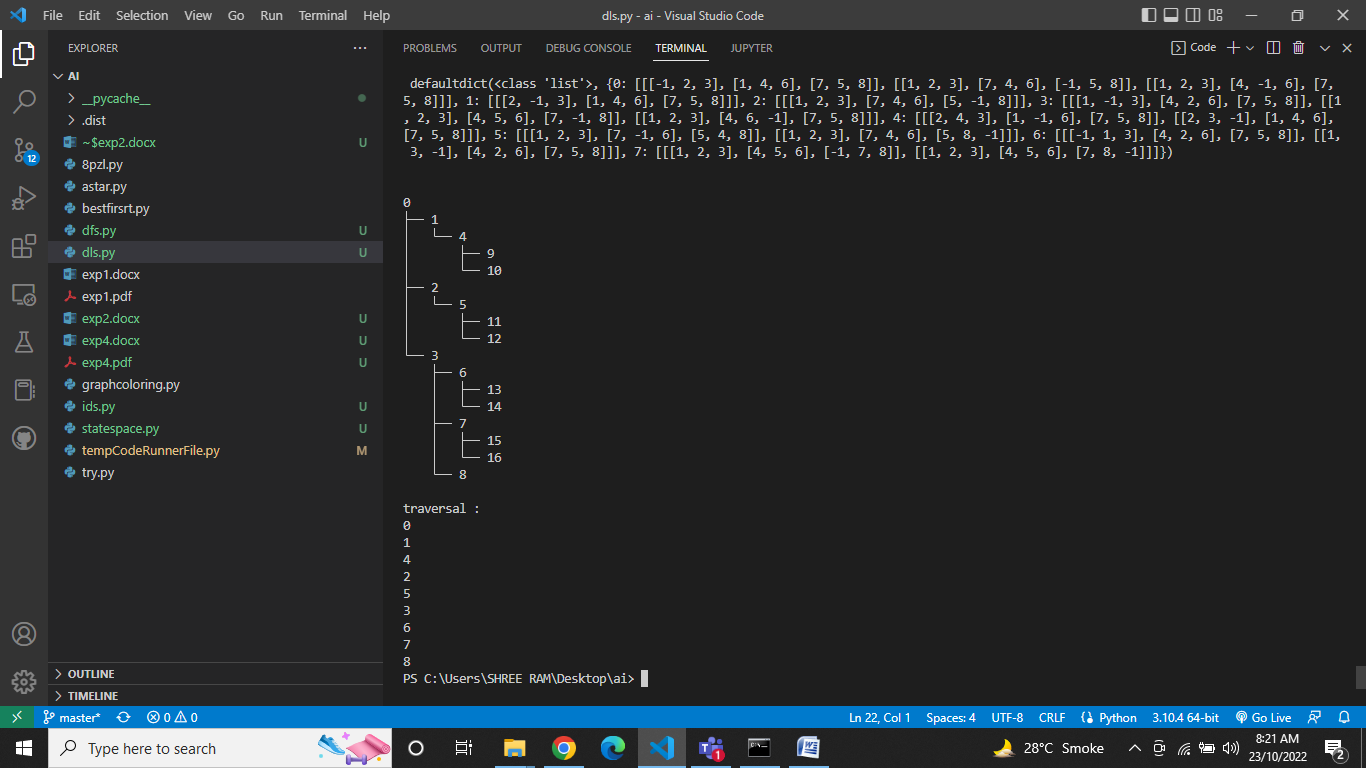
**Limit =2**

**Limit =3**

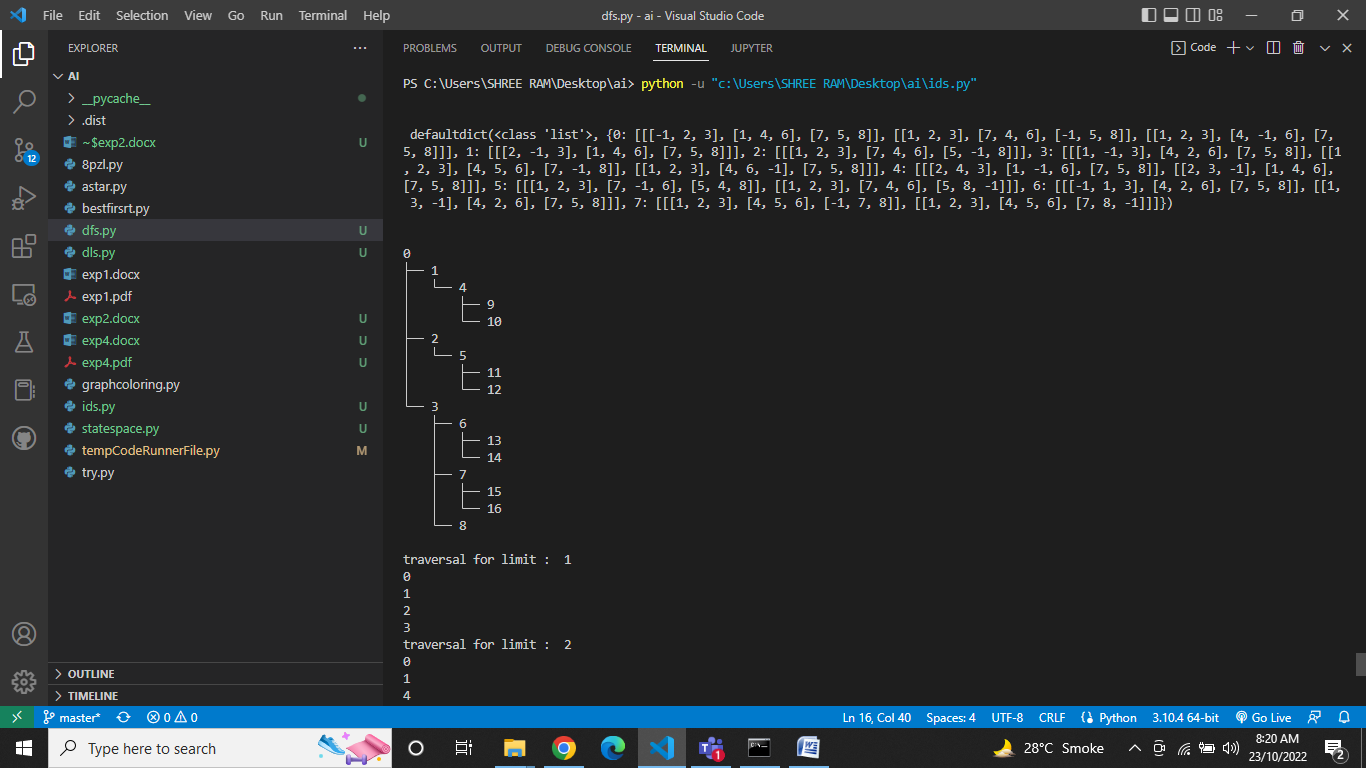
**Output:**

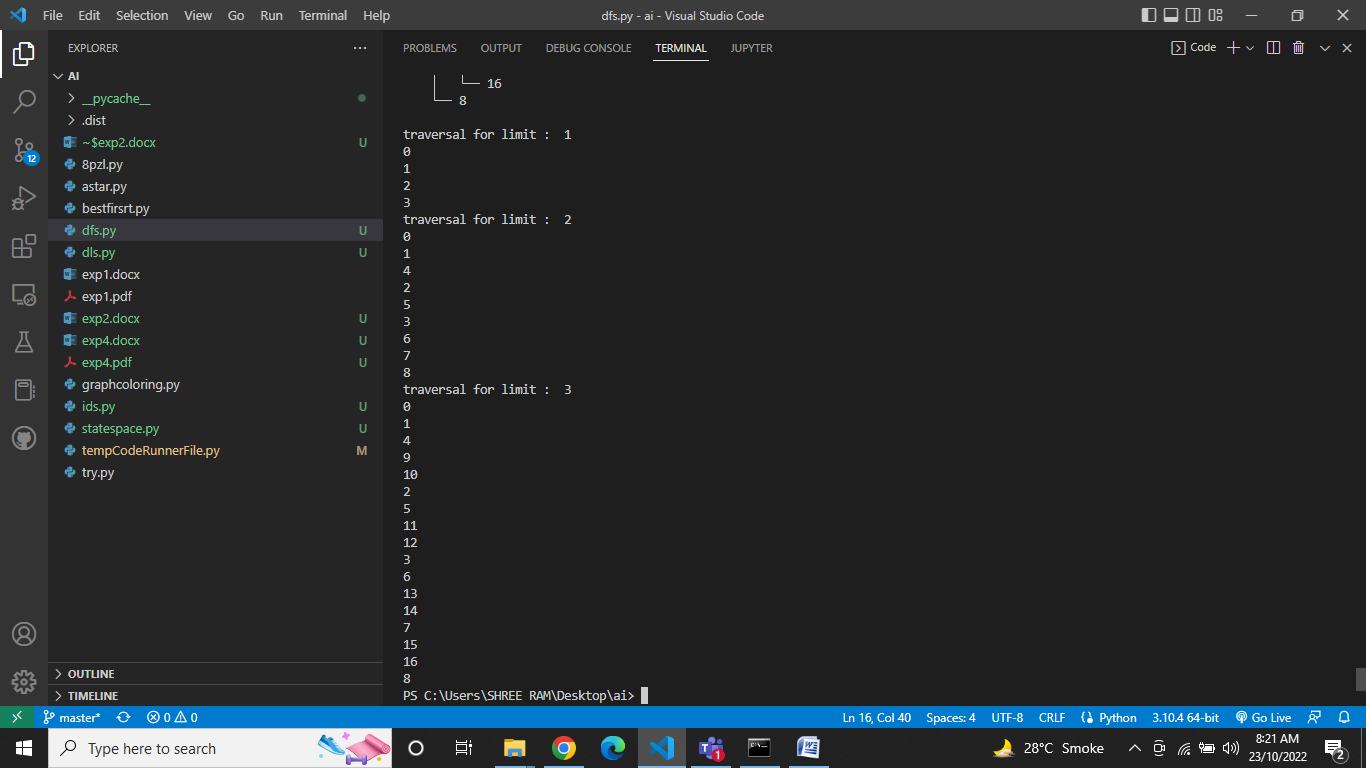
Best first search

**dls:**

****

**iddls:**

****

****

**comparison for dls and iddls:**

|  |  |  |
| --- | --- | --- |
| **Parameters** | **dls** | **iddls** |
| **completeness** | Yes only if goal exists within depth l | yes |
| **Time complexity** | O(b^l) | O(b^d) |
| **Space complexity** | O(bl) | O(bd) |
| **optimality** | Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d. | yes |

# CONCLUSION:

# Hence, we have successfully implemented dls and iddls on 8 puzzle problem