Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

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:

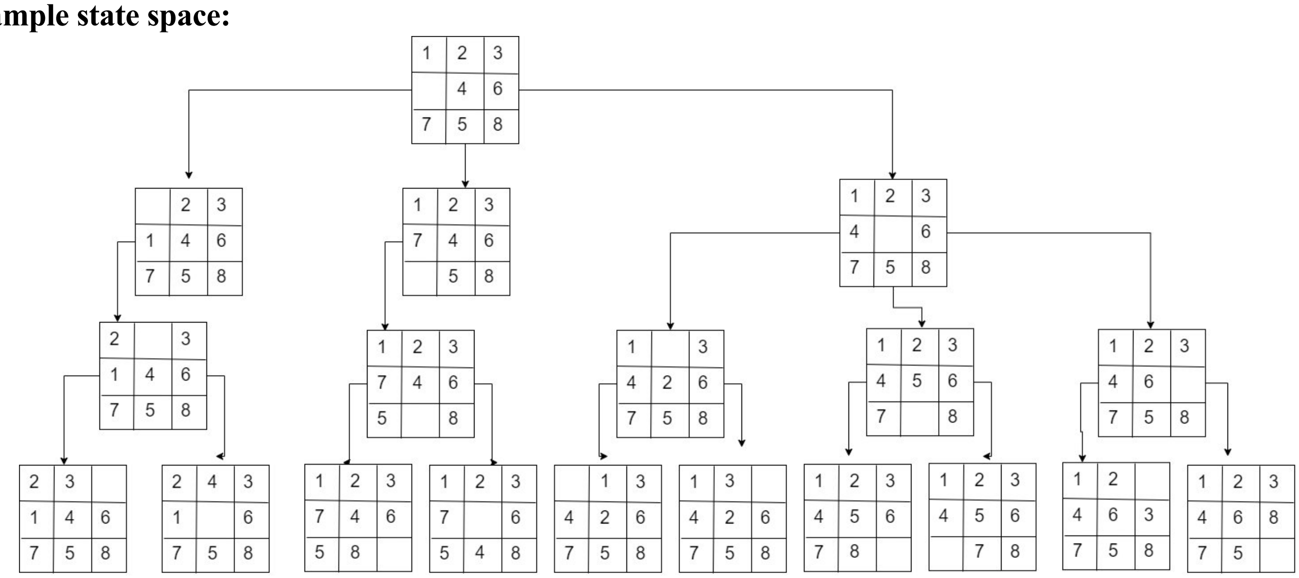
```
import statespace


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)
```

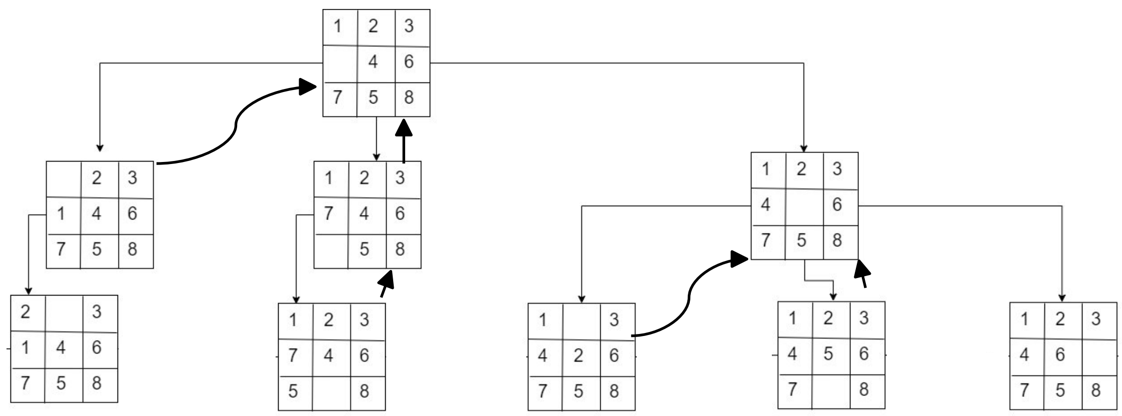
ids:

```
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()
```

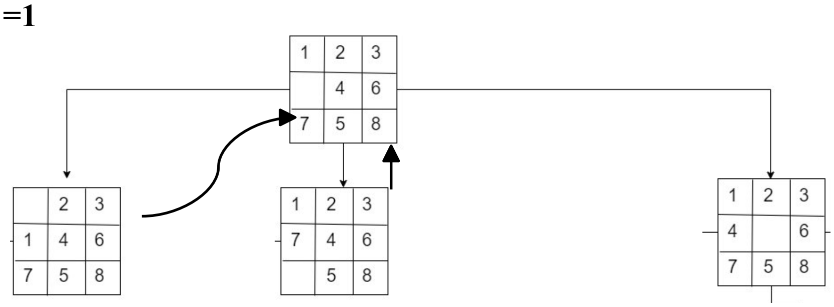
example state space:



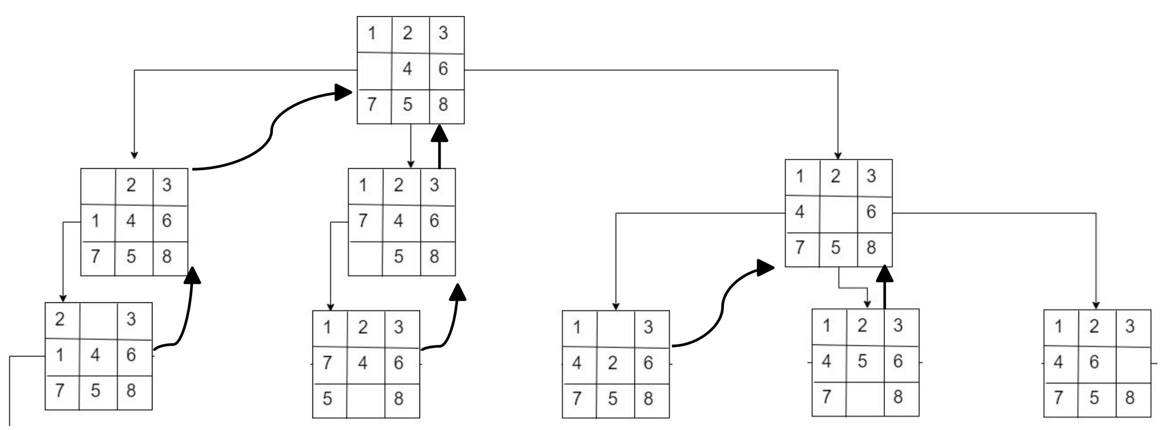
Dls(limit =2)



Iddls:
Limit =1

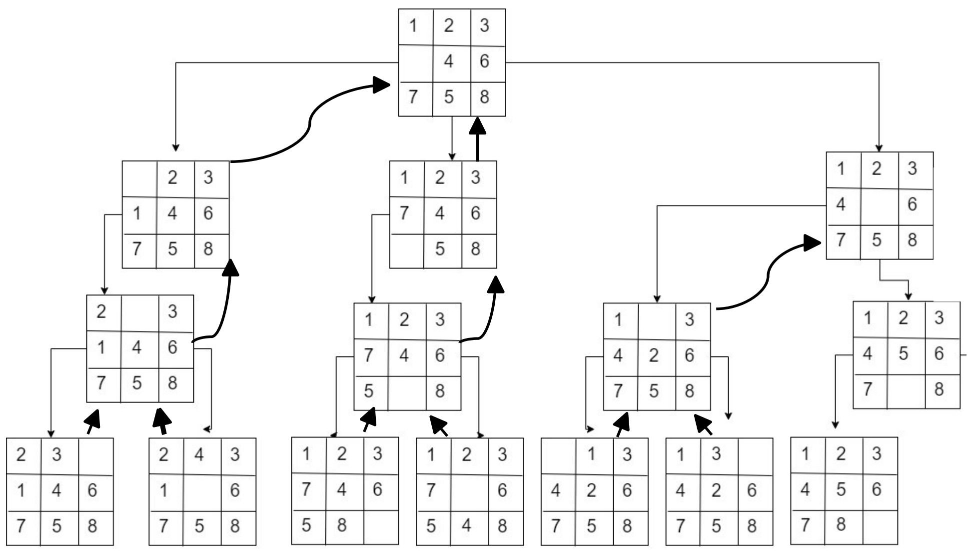


Limit =2



Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Limit =3



Output:

dls:

```
 defaultdict(<class 'list'>, {0: [[[-1, 2, 3], [1, 4, 6], [7, 5, 8]], [[1, 2, 3], [7, 4, 6], [-1, 5, 8]], [[1, 2, 3], [4, -1, 6], [7,
5, 8]]], 1: [[[2, -1, 3], [1, 4, 6], [7, 5, 8]]], 2: [[[1, 2, 3], [7, 4, 6], [5, -1, 8]]], 3: [[[1, -1, 3], [4, 2, 6], [7, 5, 8]], [[1
, 2, 3], [4, 5, 6], [7, -1, 8]], [[1, 2, 3], [4, 6, -1], [7, 5, 8]]], 4: [[[2, 4, 3], [1, -1, 6], [7, 5, 8]], [[2, 3, -1], [1, 4, 6],
[7, 5, 8]]], 5: [[[1, 2, 3], [7, -1, 6], [5, 4, 8]], [[1, 2, 3], [7, 4, 6], [5, 8, -1]]], 6: [[[-1, 1, 3], [4, 2, 6], [7, 5, 8]], [[1,
 3, -1], [4, 2, 6], [7, 5, 8]]], 7: [[[1, 2, 3], [4, 5, 6], [-1, 7, 8]], [[1, 2, 3], [4, 5, 6], [7, 8, -1]]]})


0
├── 1
│   └── 4
│       ├── 9
│       └── 10
├── 2
│   └── 5
│       ├── 11
│       └── 12
└── 3
    ├── 6
    │   ├── 13
    │   └── 14
    ├── 7
    │   ├── 15
    │   └── 16
    └── 8

traversal :
0
1
4
2
5
3
6
7
8
PS C:\Users\SHREE RAM\Desktop\ai>
```

iddls:

```
PS C:\Users\SHREE RAM\Desktop\ai> python -u "c:\Users\SHREE RAM\Desktop\ai\ids.py"


 defaultdict(<class 'list'>, {0: [[[-1, 2, 3], [1, 4, 6], [7, 5, 8]], [[1, 2, 3], [7, 4, 6], [-1, 5, 8]], [[1, 2, 3], [4, -1, 6], [7,
5, 8]]], 1: [[[2, -1, 3], [1, 4, 6], [7, 5, 8]]], 2: [[[1, 2, 3], [7, 4, 6], [5, -1, 8]]], 3: [[[1, -1, 3], [4, 2, 6], [7, 5, 8]], [[1
, 2, 3], [4, 5, 6], [7, -1, 8]], [[1, 2, 3], [4, 6, -1], [7, 5, 8]]], 4: [[[2, 4, 3], [1, -1, 6], [7, 5, 8]], [[2, 3, -1], [1, 4, 6],
[7, 5, 8]]], 5: [[[1, 2, 3], [7, -1, 6], [5, 4, 8]], [[1, 2, 3], [7, 4, 6], [5, 8, -1]]], 6: [[[-1, 1, 3], [4, 2, 6], [7, 5, 8]], [[1,
 3, -1], [4, 2, 6], [7, 5, 8]]], 7: [[[1, 2, 3], [4, 5, 6], [-1, 7, 8]], [[1, 2, 3], [4, 5, 6], [7, 8, -1]]]})


0
├── 1
│   └── 4
│       ├── 9
│       └── 10
├── 2
│   └── 5
│       ├── 11
│       └── 12
└── 3
    ├── 6
    │   ├── 13
    │   └── 14
    ├── 7
    │   ├── 15
    │   └── 16
    └── 8

traversal for limit :  1
0
1
2
3
traversal for limit :  2
0
1
4
2
5
3
6
7
8
traversal for limit :  3
0
1
4
9
10
2
5
11
12
3
6
13
14
7
15
16
8
PS C:\Users\SHREE RAM\Desktop\ai>
```

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