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.02

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 Breadth first search and depth first search on 8-Puzzle Problem

DESCRIPTION OF EXPERIMENT:

- We should generate the state space for above mentioned problem
- The traversal path for Depth first search and Breadth first search should be displayed
- compare DFS & BFS wrt time, space complexities and completeness and optimality.

Breadth First Search (BFS)

There are many ways to traverse graphs. BFS is the most commonly used approach.
BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.
As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

Depth First Search (DFS)

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.
Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.
This recursive nature of DFS can be implemented using stacks. The basic idea is as follows:
Pick a starting node and push all its adjacent nodes into a stack.
Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

Explanation/Solutions(Design):

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)

Code:

Bfs /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
```

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)

```
            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:
            print("\n*** final configuration achieved at :",nextlvl.index(final)," ***\n")
            return nextlvl

def printstatespace(nextlvl):
    print(graph)
    print(tree)
    #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()
'''
    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)
```

dfs:

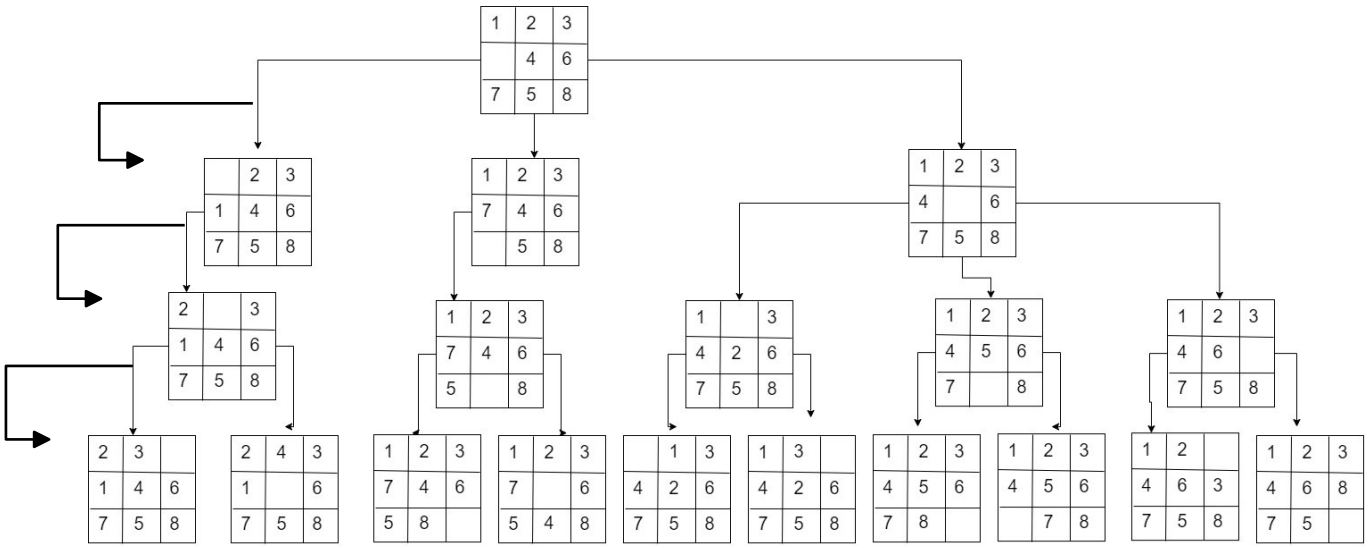
```
import statespace


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()

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
```

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)

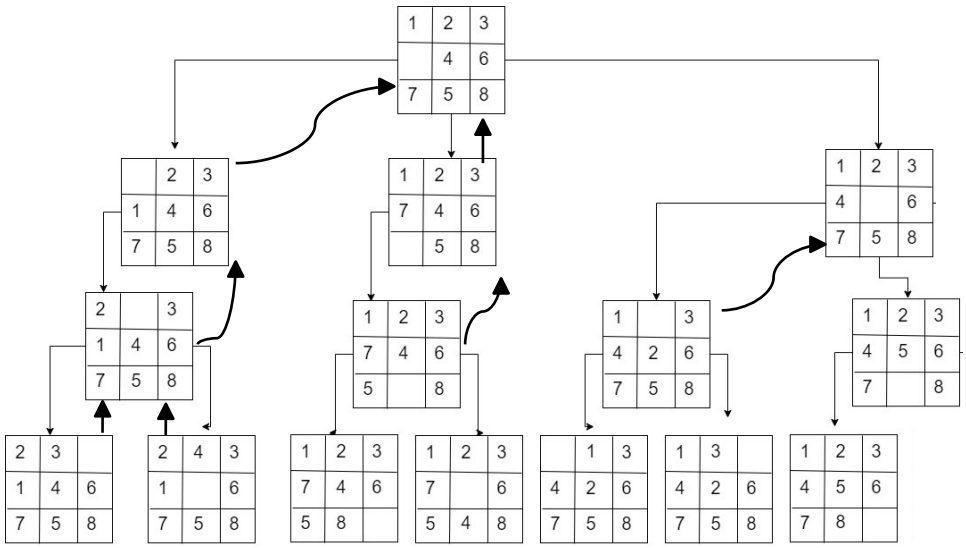
```
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

graph=statespace.move(initial,final)

print("traversal : ")
dfs(visited,graph,0)
```

traversal paths:
example

bfs:



Dfs:



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)

Output:

bfs:

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

*** final configuration achieved at : 16  ***

defaultdict(<class 'list'>, {0: [1, 2, 3], 1: [4], 2: [5], 3: [6, 7, 8], 4: [9, 10], 5: [11, 12], 6: [13, 14], 7: [15, 16]})
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

PS C:\Users\SHREE RAM\Desktop\ai>
```

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)

Dfs:

```
PS C:\Users\SHREE RAM\Desktop\ai> python -u "c:\Users\SHREE RAM\Desktop\ai\dfs.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 :
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 bfs and dfs:

Parameters	**bfs**	**dfs**
completeness	yes	No if the nodes in a given depth tend to infinity
Time complexity	O(b^d)	O(n^m)
Space complexity	O(b^d)	O(bm)
optimality	yes	no

CONCLUSION:

Hence, we have successfully implemented bfs and dfs on 8 puzzle problem