CS561 - ARTIFICIAL INTELLIGENCE LAB
ASSIGNMENT-1:DFS,BFS

Intaj Choudhury – 2211MC09

Ankit Anand – 2311MC04

Khushbu Bharti – 2311MC21

Question

The task is to check if we can reach from any random start grid to the mentioned target grid by moving the Blank space('B'). In one step, the Blank space can move either top or down or left or right.

1. Compare Breadth First Search (BFS) and Depth First Search (DFS) with respect to the number of steps required to reach the solution if they are reachable.

Solution

1. Algorithm:

Step 1: Obtain the initial state from the user while keeping the target state constant.

Step 2: Compute the inversion count for all elements and aggregate them. If the sum is odd, this signifies an unsolvable problem – meaning the goal state cannot be attained starting from the initial state. In this case, the program prints "problem cannot be solved" and terminates.

Step 3: Invoke the BFS function, employing a queue data structure to store the visited states. The number of steps needed for BFS to reach the goal state is counted.

Step 4: Call the DFS function, utilizing a stack (LIFO) data structure to retain the visited states. Calculate the number of steps taken by DFS to reach the goal state.

Step 5: Compare the number of steps taken by both functions and determine which one performed more effectively.

It's important to note that both BFS and DFS follow the order: Right, Left, Up, Down.

Target :
[1, 2, 3]
[4, 5, 6]
[7, 8, -1]

Sample Input :
[3, 2, 1]
[4, 5, 6]
[8, 7, -1]

Case 1 : When BFS performs better than DFS.

```
Enter Initial State:
Enter 1 for the sample grid.
Enter 2 for user input.
Enter your choice: 2
Enter row 1 : 1 3 2
Enter row 2 : 4 6 5
Enter row 3 : 7 8 -1

-------Initial State-------

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

-------Target State-------

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

Problem solved with BFS in 24663 steps

Problem solved with DFS in 117724 steps

BFS took fewer steps than DFS
```

Case 2 : When DFS performs better than BFS.

```
Enter Initial State:
Enter 1 for the sample grid.
Enter 2 for user input.
Enter your choice: 2
Enter row 1 : 1 2 -1
Enter row 2 : 4 5 3
Enter row 3 : 7 8 6

-------Initial State-------

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

-------Target State-------

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

Problem solved with BFS in 6 steps

Problem solved with DFS in 2 steps

DFS took fewer steps than BFS
```

Case 3 : When puzzle is not solvable.

```
Enter Initial State:
Enter 1 for the sample grid.
Enter 2 for user input.
Enter your choice: 2
Enter row 1 : 1 2 3
Enter row 2 : 5 4 6
Enter row 3 : 7 8 -1

-------Initial State-------

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

-------Target State-------

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


Problem cannot be solved
```

Output of sample input given in assignment:

```
Enter Initial State:
Enter 1 for the sample grid.
Enter 2 for user input.
Enter your choice: 2
Enter 2 for user input.
Enter your choice: 1

-------Initial State-------

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

-------Target State-------

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

Problem solved with BFS in 131207 steps

Problem solved with DFS in 152813 steps

BFS took fewer steps than DFS
```

2. Comment on which algorithm will be faster and when, by mentioning proper intuition and examples

→ Initial state which is given as Sample state:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & -1 \\ 7 & 8 & 6 \end{bmatrix}$$

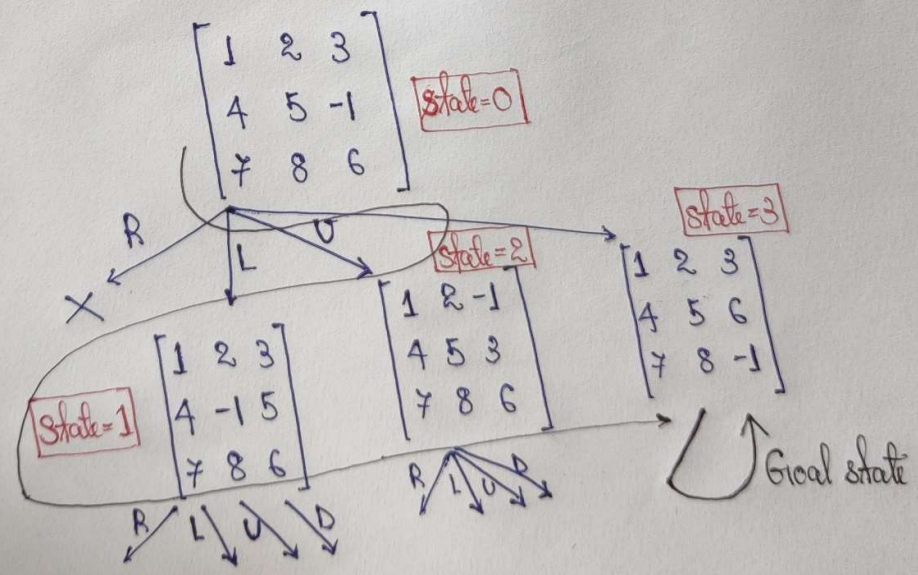
→ Target state which is termed as Goal state:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & -1 \end{bmatrix}$$

* Here, -1 is termed as Blank which is given in question.

→ By using BFS:

Here, we are using Queue Data structure.

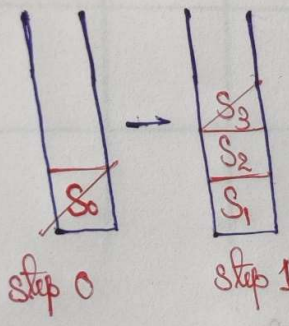


→ By Using DFS :

It is done using Stack data structure.

Steps

1. Push S_0 in stack.
2. Pop S_0 and push its childrens in stack in R, L, U, D order.
3. Again repeat the above process till we reach Goal state.



S_3 = Goal

No. of steps = 1

Breadth First Search(BFS)

Breadth First Search (BFS) stands as one of the primary strategies for navigating trees or graphs. This algorithm traverses breadth-wise, leading to its name. It initiates its search from the root node, progressively expanding all successor nodes on the current level before proceeding to the subsequent level.
Noteworthy is the fact that the breadth-first search algorithm exemplifies a general-graph search approach. It is effectively implemented through the utilization of a First-In-First-Out (FIFO) queue data structure.

Complexity Analysis

Time Complexity: The time complexity of BFS is derived from the number of nodes traversed until the shallowest Node is reached. If 'd' signifies the depth of the shallowest solution and 'b' represents nodes at each state, then the time complexity can be expressed as:

$T(b) = 1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$

Space Complexity: The space complexity of the BFS algorithm is determined by the memory size of the frontier, which is proportional to **$O(b^d)$.**

In summary, BFS offers a systematic approach for exploring a tree or graph, ensuring that the breadth-wise search leads to an efficient coverage of levels. This traversal strategy is underscored by its time and space complexities, which highlight its suitability for various scenarios.

Depth First Search (DFS)

Depth First Search (DFS) stands as a recursive algorithm designed for the traversal of tree or graph data structures. It earns its name from its methodical progression – originating from the root node, it plunges into each path to explore the deepest node before proceeding to the next path.
DFS employs a stack data(LIFO) structure for its seamless execution, a characteristic that contributes to its structured traversal.
An intriguing observation is that the process of the DFS algorithm aligns with that of the Breadth-First Search (BFS) algorithm, underpinning their shared approach in exploring data structures.

Complexity Analysis

Time Complexity: The time complexity of DFS corresponds to the number of nodes traversed by the algorithm. This can be expressed as:

$T(n) = 1 + n + n^2 + n^3 + \ldots + n^m = O(n^m)$

Here, 'm' represents the maximum depth of any node, which can potentially surpass 'd' – the depth of the shallowest solution.

Space Complexity: DFS necessitates the storage of a solitary path stemming from the root node. Consequently, its space complexity aligns with the size of the fringe set and is denoted as **$O(b^m)$.**

In essence, the choice between BFS and DFS hinges on the relationship between the depth of the state space and the branching factor. If the depth of the state space significantly surpasses the branching factor, BFS generally proves more efficient. Conversely, if the depth is considerably less than the branching factor, DFS tends to exhibit superior performance.