# **­­Indian Institute of Technology Patna**

CS571 AI & ML Laboratory

Assignment #1: BFS & DFS

Submission Date: 23rd March 2024

Group Members (2):

1. Baskar Natarajan - 2403res19(IITP001799)
2. Jyotisman Kar – 2403res35(IITP001751)

**Semester-1**

**M.Tech AI & DSC,** IIT Patna.

# Table of Contents

[**Indian Institute of Technology Patna** 1](#_Toc162245524)

[Table of Contents 2](#_Toc162245525)

[1. Problem Definition: 3](#_Toc162245526)

[Python Libraries used are, 3](#_Toc162245527)

[Common logics used are, 4](#_Toc162245528)

[**1.** Finding the blank Tile: 4](#_Toc162245529)

[**2.** Get Total Inversions and is\_solvable : 4](#_Toc162245530)

[3. Get Adjacent Cells: find all the ways to swap the blank cell. 4](#_Toc162245531)

[4. BFS: 5](#_Toc162245532)

[5. DFS: 5](#_Toc162245533)

[Analysis and Solution: 6](#_Toc162245534)

[Question 1: 7](#_Toc162245535)

[Question 2 8](#_Toc162245536)

[OUTPUT : 9](#_Toc162245537)

[Output 1: 9](#_Toc162245538)

[Output 2: 10](#_Toc162245539)

[Output 3: 11](#_Toc162245540)

[Output 4: 12](#_Toc162245541)

[Output 5: 13](#_Toc162245542)

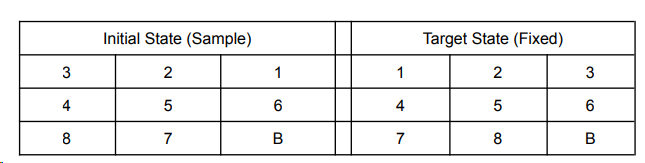
## Problem Definition:

* 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, down, left, or right.

Input:

Generate a random grid of 3×3 shape containing numbers from 1 to 8 and a blank space. The target grid is fixed.

For example,



## Python Libraries used are,

* **Deque – Double ended Queue**
  + **Efficient Insertion and removal:** since deque can be traversed on both ends. O(1) time complexity.
  + **FIFO –**It ensures the order of elements is added.
  + **Memory efficiency –** It uses doubly linked list which ensures optimum memory usage.
  + **Avoid state revisit -** allows us to easily check if a state has already been visited (using a set for example) before adding it to the queue.
  + **Simplicity and Readability:** It has intuitive methods like append and pop left, making the BFS code easier to understand
* **Random** - The random library in Python provides functionalities to generate random numbers, choose random elements from sequences, and shuffle sequences
  + In Our case we are using it to generate random initial state.
* **Datetime** – It provides various datetime functions, we are using now() function to calculate the time taken to attain the goal state.

## Common logics used are,

### Finding the blank Tile:

Using generator function to find out the row and column indices of the Blank (‘B’) cell.

### Get Total Inversions and is\_solvable :

* In the context of solving puzzles like the 8 puzzle or 15 puzzle, the "inversion technique" is used to determine whether a given puzzle state is solvable. This technique relies on the concept of inversions in permutations.

#### Understanding Inversions:

* An inversion occurs when two elements in a sequence (or permutation) are in the reverse order than they appear in the sorted sequence.
* In the context of the 8 puzzle (3x3 grid with 8 tiles and an empty space), we can represent a state as a sequence of numbers, where each number represents a tile, and 0 (or some fixed number) represents the empty space.
* When counting inversions, we look at pairs of numbers (excluding the empty space) and count how many times a larger number appears before a smaller number.
* Using Inversions to Determine Solvability:
* For an 8 puzzle state to be solvable, it must have an even number of inversions.
* This rule holds for 15 puzzle as well; it must have an even number of inversions to be solvable.
* *If the number of inversions is odd, the puzzle state is unsolvable*.
* So, in our program, if the input state is not solvable, then we will generate a new random initial state.

### Get Adjacent Cells: find all the ways to swap the blank cell.

* As explained in our problem definition, we can have the following options for swapping blank space.
  + Up (-1,0)
  + Down (1,0)
  + Left (0,-1)
  + Right(0,1)
  + To get the adjacent cells we need to do above operations, but before that, we need to check
    - Find the indices blank cell ‘B’
    - Boundary conditions like, in 3x3 matrix the index values should be 0<= new\_row/new\_call < 3

### BFS:

* BFS explores all possible moves level by level a graph (or tree), exploring all neighbors of a node before moving on to the next level. ensuring that the shortest path to the solution is found first. and It guarantees finding the shortest path if one exists provided by a condition If the target state is reachable
* It starts at the root (or any arbitrary node) and explores all the neighbor nodes at the current depth prior to moving on to nodes at the next depth level.
* **FIFO**: Uses a queue to maintain the order of exploration. Nodes are added to the back of the queue and removed from the front
* **Properties**:
  + Finds the shortest path in an unweighted graph.
  + Guarantees finding the shortest path (if it exists) in a graph without cycles.
  + Requires more memory than DFS because it needs to store all child pointers at each level.
* In Our Program, we created the following variables
  + Action - (left/right/up/down) to swap the cells adjacent
  + Path – we add each action performed by actions.
  + Neighbours – adjacent cells
  + Visited set – As soon as we get the element from the queue, add it to the visited set, so that we do not need to visit the same state again.
    - To make the state hashable, we convert the list into tuple of tuple
    - Before finding Neighbour first we will check if that is not already visited.
  + Queue – Deque, store the adjacent cells with path into queue for further processing
  + For performance evaluation the following variables were created.
    - Startime, endtime
    - Iteration
  + Finally, check the initial state equal to goal state, if same return the path else continue, and if the queue is empty and the goal state is not reached then we return None as path.

### DFS:

* DFS explores a graph (or tree) by going as deep as possible along each branch before backtracking.
* It starts at the root (or any arbitrary node) and explores as far as possible along each branch before backtracking.
* Uses a stack (or recursion) to maintain the order of exploration. It explores as far as possible along a branch before backtracking to the previous node.
  + In our program we used stack

**Properties:**

* Does not necessarily find the shortest path.
* Can get stuck in cycles if not careful (infinite loops).
* Generally, it uses less memory than BFS because it only needs to store the path from the start node to the current node.
* In Our Program, we created following variables
  + Action - (left/right/up/down) to swap the cells adjacent
  + Path – we add each action performed by actions.
  + Neighbours – adjacent cells
  + Visited set – As soon as we get the element from queue, add it to the visited set, so that we do not need to visit the same state again.
    - To make the state hashable, we convert the list in to tuple of tuple
    - Before finding Neighbour first we will check if that is not already visited.
  + Stack – Deque, store the adjacent cells with path into queue for further processing . Since Deque can be traversed both ways, we can use as stack as well.
  + For performance evaluation the following variables were created.
    - Startime, endtime
    - Iteration
  + Finally, check the initial state equal to goal state, if same return the path else continue, and if the queue is empty and the goal state is not reached then we return None as path.

## Analysis and Solution:

1. We are writing solution using Python Language, attached code is compiled and tested in 3.9
2. Initialize the goal state as mentioned in assignment 1.
3. Print the goal state
4. We have created a function called search which will have the following things.
   1. Generate the initial state using random features.
      1. That checks whether the problem can be solved or not using

Inversion Technique- mentioned in step#2

* 1. Call the BFS function to get the path, if path is None, then goal state is not reached,
     1. Generate the random initial state and call the BFS.
  2. Call the DFS function to get the path, if the path is None, then the goal stat is not reached,
     1. Generate the random initial state and call the BFS and DFS

Performance Evaluation:

To evaluate the performance, we capture the following things.

1. No. of iterations to obtain the goal
   1. How many times have queue operations done
2. No. Of steps to reach the goal
   1. How many adjacent states were created to attain the goal state
3. Total Time taken to reach the goal

## Question 1:

Compare the Breadth First Search(BFS) and Depth First Search(DFS) concerning the number of steps required to reach the solution and whether they are reachable. If unreachable, start with a random state and retry until the Target State (given above) is reached.

* We have compared 4 outputs of BFS &DFS for the same Initial State
* Details are added at the bottom of this document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Ouput No# | Iterations | Steps | Time Taken |
| BFS | 1 | 24121 | 18 | 0.337085 seconds |
| DFS | 1 | 10106 | 9728 | 1.506887 seconds |
| BFS | 2 | 119143 | 24 | 1.570668 seconds |
| DFS | 2 | 13254 | 12716 | 2.201378 seconds (First input DFS not reached goal state in 20000 steps, so retried, second random input generated) |
| BFS | 3 | 132208 | 24 | 1.760911 seconds |
| DFS | 3 | 3572 | 3434 | 0.122322 seconds |
| BFS | 4 | 123279 | 24 | 1.647609 seconds |
| DFS | 4 | 18578 | 17760 | 5.410838 seconds |
| BFS | 5 | 426 | 10 | 0.002669 seconds |
| DFS | 5 | 12595 | 12086 | 1.699696 seconds |

According to the output we can get following info.

1. Average steps taken to attain goal : BFS : 20
2. Average Time taken: BFS: 1.5 seconds
3. Average iterations taken: 99,687
4. Average steps taken to attain goal : DFS : 10909
5. Average Time Taken DFS is: 2.325 seconds
6. Average iterations taken DFS: 11,377

From the above statistics and from the given input BFS wins over the DFS.

## Question 2

Comment on which algorithm will be faster and when by mentioning proper intuition and example

When to Choose:

**Use BFS**:

* When you need to find the shortest path in an unweighted graph.
* When the depth of the tree or graph is large, and you want to minimize the memory usage.
* When you need to explore all neighbors equally.
* If the target state is reachable, BFS will find it with the minimum number of steps required.
* If the target state is unreachable, BFS will continue searching until all possible states are explored.

**Use DFS**:

* When you want to visit as far as possible along a branch before backtracking.
* When the tree or graph is deep and you want to find one solution without exploring all.
* When memory usage is a concern, and the tree or graph is not too deep.
* DFS might be faster in certain scenarios where the target state is closer to the root of the search tree, but it doesn't guarantee finding the shortest path. If the target state is unreachable or deep in the search tree, DFS might take longer to find a solution compared to BFS

BFS is generally used when the shortest path is needed or when the graph is unweighted, while DFS is more suitable for exploring deeply into a graph, detecting cycles, or when memory usage is a concern. Both algorithms have their strengths and are chosen based on the specific requirements of the problem.

**Example**:

* If you observe our outputs, DFS is taking less iterations even though more steps are being taken to reach the goal state.
* In Our example outputs, #2, BFS found the solution very easily but DFS not reached even after 20000 steps, so in this case the initial state has more depth.
* Output 1, BFS has done very fast way to reach the goal. But DFS does not due to the depths of the neighbors.s

## OUTPUT :

Following are the sample outputs.

### Output 1:

Goal State:

1 2 3

4 5 6

7 8 B

Attempt 1 : Random Initial State:

B 3 2

5 1 7

4 8 6

Starting BFS..

BFS Goal Reached!

BFS Total Step Taken is : 18

BFS Total Iterations Taken is : 24121

BFS Total Time Taken to attain the goal state in 0.337085 seconds

Starting DFS..

DFS Goal Reached!

DFS Total Step Taken is : 9728

DFS Total Iterations Taken is : 10106

DFS Total Time Taken to attain the goal state in 1.506887 seconds

Process finished with exit code 0

### Output 2:

C:\Users\MSUSERSL123\PycharmProjects\IITP\_AI\venv\Scripts\python.exe C:\Users\MSUSERSL123\PycharmProjects\IITP\_AI\BFS\_DFS\_8\_PUZZLE\_SOLUTION.py

Goal State:

1 2 3

4 5 6

7 8 B

Attempt 1 : Random Initial State:

6 2 5

3 7 1

4 8 B

Starting BFS..

BFS Goal Reached!

BFS Total Step Taken is : 20

BFS Total Iterations Taken is : 48817

BFS Total Time Taken to attain the goal state in 0.657469 seconds

Starting DFS..

DFS reached 20000 steps,So retrying with new random sample

Attempt 2 : Random Initial State:

2 8 7

4 5 3

6 1 B

Starting BFS..

BFS Goal Reached!

BFS ToTal Step Taken is : 24

BFS Total Iterations Taken is : 119143

BFS Total Time Taken to attain the goal state in 1.570668 seconds

Starting DFS..

DFS Goal Reached!

DFS ToTal Step Taken is : 12716

DFS Total Iterations Taken is : 13254

DFS Total Time Taken to attain the goal state in 2.201378 seconds

Process finished with exit code 0

### Output 3:

Random Initial State:

8 1 7

4 6 3

2 5 B

Goal State:

1 2 3

4 5 6

7 8 B

Attempt 1 : Starting BFS..

BFS Goal Reached!

BFS Total Step Taken is : 24

BFS Total Iterations Taken is : 132208

BFS Total Time Taken to attain the goal state in 1.760911 seconds

Starting DFS..

DFS Goal Reached!

DFS Total Step Taken is : 3434

DFS Total Iterations Taken is : 3572

DFS Total Time Taken to attain the goal state in 0.122322 seconds

### Output 4:

Goal State:

1 2 3

4 5 6

7 8 B

Attempt 1 : Random Initial State:

4 7 5

3 6 2

1 8 B

Starting BFS..

BFS Goal Reached!

BFS ToTal Step Taken is : 24

BFS Total Iterations Taken is : 123279

BFS Total Time Taken to attain the goal state in 1.647609 seconds

Starting DFS..

DFS Goal Reached!

DFS ToTal Step Taken is : 17760

DFS Total Iterations Taken is : 18578

DFS Total Time Taken to attain the goal state in 5.410838 seconds

### Output 5:

Goal State:

1 2 3

4 5 6

7 8 B

Attempt 1 : Random Initial State:

1 3 8

4 2 5

7 6 B

Starting BFS..

BFS Goal Reached!

BFS Total Step Taken is : 10

BFS Total Iterations Taken is : 426

BFS Total Time Taken to attain the goal state in 0.002669 seconds

Starting DFS..

DFS Goal Reached!

DFS Total Step Taken is : 12086

DFS Total Iterations Taken is : 12595

DFS Total Time Taken to attain the goal state in 1.699696 seconds