**Team Members:**

**2403res81 , 2303res132, 2303res204**

**Summary of the assignment.py file**

1. How to run the filename.py

python filename.py

1. Functions used and there functionality
   1. def create\_initial\_state(state\_values) : This function receives state\_values which is a list containing values from 1 to 8 and letter ‘B’ which signifies a blank tile in the grid . This list is used to generate a 3\*3 grid which random placements of state\_values. To generate random positions of state\_values , it is shuffled using the python package “random” function “shuffle”.

For eg : state\_values = [1,2,3,4,5,6,7,8,’B’]

After applying a random package function shuffle , it will change it to ['B', 3, 5, 4, 7, 8, 2, 6, 1] .

This output will be different when called again.

* 1. def find\_blank\_tile(state): This function receives a state matrix and returns a list of rows and columns where value ‘B’ is found.
  2. def generate\_new\_states(state): This function receives a state matrix and returns a list that contains matrices with valid moves. In this function first we find a blank tile by calling function **find\_blank\_tile** and based on the returned row and col of blank tile ,new matrices are generated by iterating on valid moves.
  3. def convert\_arr\_tuple(matrix) : This function converts a matrix which is list of list to tuple of tuple. This function is needed so that we can easily compare arrays after pushing them into set().
  4. def dfs(start,target): This function performs depth first search .
     1. Input to this function is a state matrix and target matrix which needs to be reached.
     2. A stack is created using a list and a visited set is created using set. Stack is used to insert the states and visited is used to store all the states that have been visited while performing dfs. It is needed in order to avoid visiting a state that has been visited again.
     3. Initially in the stack , state matrix and steps to reach that state matrix is inserted as a tuple. For initial state steps will be 0.
     4. Iterate till the stack is empty
     5. If stack is not empty pop the last element which is state and steps
     6. If the state is equal to the target return True and print the steps that was needed to reach the target
     7. If the state is not equal to target add the state to the visited set after converting it to the tuple of tuple using the function **convert\_arr\_tuple** .
     8. Call the function **generate\_new\_states** and store the returned list into variable new\_states.
     9. Iterate over new states and push them into stack if is not present in the visited.
     10. While pushing the elements into the stack, increase the steps by 1.
     11. Go to step iv and if the stack is empty. Return False
  5. def bfs(start,target): This function performs breadth first search .
     1. Input to this function is a state matrix and target matrix which needs to be reached.
     2. A queue is created using a list and a visited set is created using set. Stack is used to insert the states and visited is used to store all the states that have been visited while performing dfs. It is needed in order to avoid visiting a state that has been visited again.
     3. Initially in the queue , the state matrix and steps to reach that state matrix is inserted as a tuple. For initial state steps will be 0.
     4. Iterate till the queue is empty
     5. If queue is not empty pop the first element which is state and steps
     6. If the state is equal to the target return True and print the steps that was needed to reach the target.
     7. If the state is not equal to target add the state to the visited set after converting it to the tuple of tuple using the function **convert\_arr\_tuple** .
     8. Call the function **generate\_new\_states** and store the returned list into variable new\_states.
     9. Iterate over new states and push them into queue if they are not present in the visited.
     10. While pushing the elements into the queue, increase the steps by 1.
     11. Go to step iv and if the queue is empty. Return False
  6. def main(): This is the main function which is called every time when the .py file run.
     1. In this function the state matrix given in assignment is initialized as state.
     2. The target matrix is stored in the target variable.
     3. In case a given state is not reached by the target, random state matrices have to be created. To do that, state\_values is initialized as a list containing numbers from 1 to 8 and ‘B’.
     4. First bfs and dfs are called on the initial given state matrix and target matrix. If the output from both functions is true then we exit the main .
     5. If the output is False from both of them perform below steps :
        1. Initialize a count as 0 which ensures that functions will be for a finite duration. In this assignment 362880 times new random states are generated
        2. Initialize the flag as False which gives the information whether any solution was found or not till the end. If it stays False for 362880 iterations print statement as **Solution not found after 362880 iterations**
        3. Iterate using while loop till count is less than 362880.
        4. Generate a new initial matrix and call bfs and dfs . If the output from both bfs and dfs is True, break and exit the main function.
        5. If it is false then increase the count and repeat 4.

Note : **362880** this number is chosen because for 3\*3 grid having unique values factorial(9) unique possible combinations exist.

**Sample output for given initial state and target matrix**

Initial State matrix is :

[[3, 2, 1], [4, 5, 6], [8, 7, 'B']]

Target matrix is :

[[1, 2, 3], [4, 5, 6], [7, 8, 'B']]

BFS : State found with steps 24

DFS : State found with steps 18392

Q1. Compare the Breadth First Search(BFS) and Depth First Search(DFS) with respect to 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.

Answer : Using the given initial matrix, we can each the target :

By using BFS : 24 steps

By using DFS : 18392 steps

Q2. Comment on which algorithm will be faster and when by mentioning proper intuition and examples .

Answer : For the given state bfs is much faster by dfs . Reason for this bfs finds the solution breadth wise i.e. for every state it first checks solution on every level and then goes to the next whereas in depth first search we explore the complete depth until we find solution or depth limit is reached . So in any case dfs cannot be faster than bfs. But it could be equal to dfs .

Note :

While developing the assignment it was found that we can have faster steps if constraint on dfs by setting some depth limit of 50 . In that scenario below given solution was observed

Initial State matrix is :

[[3, 2, 1], [4, 5, 6], [8, 7, 'B']]

Target matrix is :

[[1, 2, 3], [4, 5, 6], [7, 8, 'B']]

BFS : State found with cost 24

DFS : State found with cost 48