CS571 - ARTIFICIAL INTELLIGENCE LAB

Lab - 1
DFS & BFS

Jenish Monpara 1901CS28
Tanishq Malu 1901CS63
Tarusi Mittal 1901CS65

---

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.

Sol:

Drive link for code:
https://drive.google.com/file/d/1kVgUULQGY6ip_QdkjtXTRU322WymoE5K/view?usp=sharing

```
from queue import Queue

# Depth of optimum path in DFS
dfs_depth = 0
total_dfs_iteration = 0

#Depth of optimum path in BFS
bfs_depth = 0
total_bfs_iteration = 0

def state_input(input_prompt) :
    print(f'Enter {input_prompt} : ')
    input_matrix = []                           #    Stores    input
matrix
    def convert_to_int(x) :                 #  converts  the  string
input into int
        try :
```

```
            return int(x)
        except(ValueError,TypeError)  :
            return 0

    for i in range(3) :                    #   Takes   the   matrix
input as list of list

input_matrix.append(list(map(convert_to_int,input().split())))

    return get_hash(input_matrix)          #   find   the   hash   of
matrix

# Converts a given hash into matrix
def get_grid(matrix_string_hash) :
    string_matrix_unhash = []
    for i in range(3) :
        string_matrix_unhash.append([])
        for j in range(3) :
            string_matrix_unhash[-1].append(matrix_string_hash %
10)
            matrix_string_hash //= 10
        string_matrix_unhash[-1].reverse()
    string_matrix_unhash.reverse()
    return string_matrix_unhash

# Direction matrix
dx = [1,-1,0,0]
dy = [0,0,-1,1]

# Checks bounds in the matrix while moving the blank space
def check_bounds(x,y) :
    return x >= 0 and y >= 0 and x <= 2 and y <= 2

# get the position of blank tile in the matrix
def get_blank_indexes(matrix_state) :
    for i in range(3) :
```

```
        for j in range(3) :
            if matrix_state[i][j] == 0 :
                return (i,j)

# Gets hash of the matrix
def get_hash(matrix_state) :
    # print(matrix_state)
    matrix_string_hash = 0
    for i in range(3):
        for j in range(3):
            matrix_string_hash *= 10
            matrix_string_hash += matrix_state[i][j]

    return matrix_string_hash

# BFS into the matrix
def bfs(current_state,target_state_hash):
     global bfs_depth
     global total_bfs_iteration

     # Queue for iterative bfs
     q = Queue(maxsize = 0)

     # Set to store hash value of visited states
     visited_state = {current_state}
     q.put((current_state,0))  #  stores  current  state,  no.  of
steps required to reach there
     while q.empty() == False:
          current_state_hash, current_bfs_count = q.get()
          visited_state.add(current_state_hash)

          # If reached the final state
          if current_state_hash == target_state_hash :
               bfs_depth = current_bfs_count
               return True
```

```
          # Convert the given hash string into matrix
          current_state_matrix = get_grid(current_state_hash)

          # get the position of blank tile
          i,j = get_blank_indexes(current_state_matrix)

          for k in range(4) :
               nxti = i +  dx[k]
               nxtj = j + dy[k]

               # if the next state is valid, travese
               if check_bounds(nxti,nxtj) :
                    current_state_matrix[i][j],
current_state_matrix[nxti][nxtj]                                 =
current_state_matrix[nxti][nxtj], current_state_matrix[i][j]
                    new_state_hash                               =
get_hash(current_state_matrix)
                    current_state_matrix[i][j],
current_state_matrix[nxti][nxtj]                                 =
current_state_matrix[nxti][nxtj], current_state_matrix[i][j]

                    # If not state not reached already
                    if new_state_hash not in visited_state :
                         total_bfs_iteration                     =
total_bfs_iteration + 1

q.put((new_state_hash,current_bfs_count + 1))

     # If we reach here then the state is not reachable
     print("State Not Reachable via Bfs")
     return False

# DFS into the matrix
def dfs(current_state,target_state_hash) :
    global dfs_depth
    global total_dfs_iteration
```

```
    # Stack for iterative dfs
    stack = []
    stack.append((current_state,0))#  Stores  current  state  and
number of steps required to reach there

    # Set to store hash value of visited states
    visited_state = set()

    while(len(stack)):
     current_state_hash = stack[-1][0]
     current_dfs_count = stack[-1][1]

     print(current_state_hash,current_dfs_count)  # debug
     stack.pop()
     visited_state.add(current_state_hash)

     # If reached the final state
     if current_state_hash == target_state_hash :
          dfs_depth = current_dfs_count
          return True

     # Convert the given hash string into matrix
     current_state_matrix = get_grid(current_state_hash)

     # get the position of blank tile
     i,j = get_blank_indexes(current_state_matrix)
     for k in range(4) :
             nxti = i +  dx[k]
             nxtj = j + dy[k]

               # if the next state is valid, traverse
             if check_bounds(nxti,nxtj) :
                                     current_state_matrix[i][j],
current_state_matrix[nxti][nxtj]                               =
current_state_matrix[nxti][nxtj], current_state_matrix[i][j]
```

```
                                                          new_state_hash    =
get_hash(current_state_matrix)
                                             current_state_matrix[i][j],
current_state_matrix[nxti][nxtj]                                           =
current_state_matrix[nxti][nxtj], current_state_matrix[i][j]
                     # if not state not reached already
                     if new_state_hash not in visited_state :
                         total_dfs_iteration = total_dfs_iteration +
1

stack.append((new_state_hash,current_dfs_count + 1))

    # if we reach here then the state is not reachable
    print("State Not Reachable via dfs")
    return False

# DFS & BFS
def solve(start_state_hash,target_state_hash) :
    # Depth of best path
    global dfs_depth
    global bfs_depth
    # Total number of iteration
    global total_dfs_iteration
    global total_bfs_iteration

    # Do DFS
    dfs(start_state_hash,target_state_hash)
    print(f'DFS Depth : {dfs_depth}')
    print(f'Total DFS iterations : {total_dfs_iteration}')

    # DO BFS
    bfs(start_state_hash,target_state_hash)
    print(f'BFS Depth : {bfs_depth}')
    print(f'Total BFS iterations : {total_bfs_iteration}')

# Main function
```

```python
def main():
     start_state = state_input("Start grid")
     target_state = state_input("Target grid")
     solve(start_state,target_state)

# Driver code
if __name__ == '__main__':
     main()
```

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

Sol :

BFS or Breadth First Search solution is faster than DFS or Depth First Search in most cases. The intuition comes naturally from the way both the algorithm's are formulated. At times a DFS search might be faster but in general the BFS search, which has greater branching factor usually tends to find a solution faster than DFS search.

For example :

Start Matrix :

B	1	2
3	4	5
6	7	8

Target Matrix:

3	1	2
6	4	5
7	8	B

Run: AI assign

```
/Users/jenish/PycharmProjects
Enter Start grid :
0 1 2
3 4 5
6 7 8
Enter Target grid :
3 1 2
6 4 5
7 8 0
Steps taken by DFS: 84895
Steps taken by BFS: 17
```

Here you can see that states explored in BFS are much lesser than that of DFS.

However the space complexity of BFS increases exponentially when the average depth of solution increases. So for problems where the solution may be at some greater depth say more than 30, DFS or some combination of BFS + DFS should be used to get better time complexity.

—-------------------------------------END—-----------------------------------