final

February 25, 2023

[16]:
```
import numpy as np
from collections import deque
import copy
import time
from collections import defaultdict
from itertools import chain

'''
get_action is used to know the current location of file
'''
def get_action(current_node):
    blank_tile_loc  = np.array(current_node)
    d = np.where(blank_tile_loc == 0)
    loc_x,loc_y = d[0][0],d[1][0]
    action = action_set[(loc_x,loc_y)]
    return action,(loc_x,loc_y)
'''
Below are the short descriptions of actions
ActionMoveUp - The Blank Tile Moves Up from its current position by swapping.
ActionMoveDown - The Blank Tile Moves Down from its current position by␣
 ↪swapping.
ActionMoveLeft - The Blank Tile Moves Left from its current position by␣
 ↪swapping.
ActionMoveRight -The Blank Tile Moves Right from its current position by␣
 ↪swapping.
'''
def ActionMoveUp(current_node,loc,blank_tile_loc):
    node = copy.deepcopy(current_node)
    temp = node[loc[0]][loc[1]]
    node[loc[0]][loc[1]] = node[blank_tile_loc[0]][blank_tile_loc[1]]
    node[blank_tile_loc[0]][blank_tile_loc[1]] = temp
    return node
def ActionMoveDown(current_node,loc,blank_tile_loc):
    node = copy.deepcopy(current_node)
    temp = node[loc[0]][loc[1]]
    node[loc[0]][loc[1]] = node[blank_tile_loc[0]][blank_tile_loc[1]]
    node[blank_tile_loc[0]][blank_tile_loc[1]] = temp
```

```python
    return node
def ActionMoveLeft(current_node,loc,blank_tile_loc):
    node = copy.deepcopy(current_node)
    temp = node[loc[0]][loc[1]]
    node[loc[0]][loc[1]] = node[blank_tile_loc[0]][blank_tile_loc[1]]
    node[blank_tile_loc[0]][blank_tile_loc[1]] = temp
    return node
def ActionMoveRight(current_node,loc,blank_tile_loc):
    node = copy.deepcopy(current_node)
    temp = node[loc[0]][loc[1]]
    node[loc[0]][loc[1]] = node[blank_tile_loc[0]][blank_tile_loc[1]]
    node[blank_tile_loc[0]][blank_tile_loc[1]] = temp
    return node
'''
Get the Indices for the optimal path
'''
def get_back_track_indices(goal_index,goal_indices):
    goal_indices_list = []
    goal_indices_list.append(goal_index)
    check = True
    while(check):
        if(0 == goal_index):
            check = False
        for key, list_of_values in goal_indices.items():
             if goal_index in list_of_values :
                    goal_indices_list.append(key)
                    goal_index = key
    goal_indices_list = goal_indices_list[::-1]
    return goal_indices_list
'''
The Below Function is used to generate two files Nodes.txt and nodePath.txt
'''
def generate_nodes_file(filename,visited_list):
    f = open(filename,'w+')
    for i in visited_list:
        ab = np.array(i).T
        flatten_list = np.ravel(ab)
        string_write = " ".join(map(str,flatten_list))
        f.write(string_write+"\n")
'''
The Below Function is used to generate the NodesInfo.txt
'''
def generate_nodeinfo_file(filename,node_path):
    f = open(filename,'w+')
    f.write("ParentNode"+'    '+"ChildNode"+'    '+ "NodeValue"+'\n')
    for key,values in node_path.items():
        for i in values:
```

```python
            ab = np.array(i[1]).T
            flatten_list = np.ravel(ab)
            string_write = " ".join(map(str,flatten_list))
            f.write(str(key)+'    '+str(i[0])+'    '+ string_write+'\n')
'''
The Below Function is used to generate the optimal path using the back track␣
↪indices and the whole node path
'''
def generate_path(node_path,back_track_indices):
    optimal_path = []
    optimal_path.append(start)
    result = False
    for i in range(len(back_track_indices)):
        a = node_path[back_track_indices[i]]
        for j in a:
            if(i == len(back_track_indices)-1):
                    if(j[0] == back_track_indices[-1]):
                        print("yes")
                        optimal_path.append(j[1])
            elif (j[0] == back_track_indices[i+1]):
                    optimal_path.append(j[1])
    return optimal_path
def main(start,goal):

    visited=[]
    q = []
    parent_node_index = 0
    q.append(start)
    i = 0
    count_start = 0
    count_end = 1
    node_indices = defaultdict(list)
    node_path =defaultdict(list)
    node_path[parent_node_index].append((0,start))
    visited.append(start)
    goal_reached = False
    while q:
        current_node = q.pop(0)
        action_dict,blank_tile_loc = get_action(current_node)
        count_for_loop=0
        if(goal_reached == True):
                break
        for action,loc in action_dict.items():
            if(action == 'l'):
                NewNode = ActionMoveLeft(current_node,loc,blank_tile_loc)
                if (NewNode not in visited):
                    visited.append(NewNode)
```

```
                    q.append(NewNode)
                    count_for_loop+=1
                    if (goal == NewNode):
                        goal_reached = True
                        print("Goal Reached!")
                        break
            elif(action == 'r'):
                NewNode = ActionMoveRight(current_node,loc,blank_tile_loc)
                if (NewNode not in visited):
                    visited.append(NewNode)
                    q.append(NewNode)
                    count_for_loop+=1
                    if (goal == NewNode):
                        goal_reached = True
                        print("Goal Reached!")
                        break
            elif(action == 'u'):
                NewNode = ActionMoveUp(current_node,loc,blank_tile_loc)
                if (NewNode not in visited):
                    visited.append(NewNode)
                    q.append(NewNode)
                    count_for_loop+=1
                    if (goal == NewNode):
                        goal_reached = True
                        print("Goal Reached!")
                        break
            elif(action == 'd'):
                NewNode = ActionMoveDown(current_node,loc,blank_tile_loc)
                if (NewNode not in visited):
                    visited.append(NewNode)
                    q.append(NewNode)
                    count_for_loop+=1
                    if (goal == NewNode):
                        goal_reached = True
                        print("Goal Reached!")
                        break
        count_start = count_end
        count_end+=count_for_loop
        for j in range(count_start,count_end):
            node_path[parent_node_index].append((j,visited[j]))
            node_indices[parent_node_index].append(j)
        parent_node_index+=1
        i+=1
    generate_nodes_file("Nodes.txt",visited)
    return node_indices,node_path
```

```
'''
Main Start Of the Program
Test Cases are taken in the same format as provided and later converted to␣
 ↪row-wise to suffice for the correct implementation of the
program.
'''

start_time = time.time()
#GivenTestCase1
arg1 = [[1,6,7],[2,0,5],[4,3,8]]
arg2 = [[1,4,7],[2,5,8],[3,0,6]]
#Given TestCase2
# arg1 = [[4,7,8],[2,1,5],[3,6,0]]
# arg2 = [[1,4,7],[2,5,8],[3,6,0]]
#Random Test Case
# arg1 = [[8,3,2],[7,4,6],[0,5,1]]
# arg2 = [[5,2,0],[8,3,4],[7,1,6]]
#Random Test Case 2
# arg1 = [[1,6,7],[2,0,5],[4,3,8]]
# arg2 = [[1,6,7],[2,3,5],[0,4,8]]
# arg1 = [[1,6,7],[2,0,5],[4,3,8]]
# arg2 = [[1,6,7],[2,3,5],[4,8,0]]

#The Below is used to transpose the lists given
start = [list(x) for x in zip(*arg1)]
goal = [list(x) for x in zip(*arg2)]
print("BFS Started")
print("The start node : ",np.ravel(start))
print("The Goal Node :",np.ravel(goal))

#Action set to perform any of the four movements based on the blank tile␣
 ↪position in the array
# u : Up , d : down , r : right , l : left
action_set = {(0,0):{'r' : (0,1),'d':(1,0)}, (0,1) : {'l' : (0,0) , 'r' :(0,2)␣
 ↪, 'd' :(1,1)} , (0,2) : {'l' : (0,1) ,'d' : (1,2)},
              (1,0) : { 'r' : (1,1) ,'u' : (0,0) ,'d' :(2,0)} , (1,1) : {'l' :␣
 ↪(1,0), 'r' : (1,2),'u' : (0,1),'d' : (2,1)},
              (1,2) : { 'l' : (1,1) , 'u'  : (0,2),'d' : (2,2)} , (2,0) : {'r' :
 ↪ (2,1),'u' : (1,0)},
              (2,1) : {'l' : (2,0),'r' : (2,2) ,'u' : (1,1)}, (2,2) : {'l' :␣
 ↪(2,1),'u' :(1,2)}}

node_indices,node_path = main(start,goal)
path_to_list= list(node_path)
last_node_index = path_to_list[-1]
```

```python
last_node = node_path[last_node_index]
for i in range(len(last_node)):
    if(last_node[i][1] == goal):
        goal_index = last_node[i][0]
back_track_indices = get_back_track_indices(goal_index,node_indices)
optimal_path = generate_path(node_path,back_track_indices)
print("The Back Track Indices of optimal path",back_track_indices)
generate_nodes_file("nodePath.txt",optimal_path)
generate_nodeinfo_file("NodesInfo.txt",node_path)
end_time = time.time()
total_time = end_time-start_time
print("Time Taken By the Algorithm:",total_time)
```

```
BFS Started
The start node :  [1 2 4 6 0 3 7 5 8]
The Goal Node : [1 2 3 4 5 0 7 8 6]
Goal Reached!
The Back Track Indices of optimal path [0, 1, 5, 13, 21, 37, 69, 130, 203, 339,
541, 923, 1444, 2410]
Time Taken By the Algorithm: 0.5792896747589111
```

[ ]:

```python

```