```
# Breadth First Search Algorithm for 8-puzzle problem
import copy


""" Transpose function """
def transpose(l1, l2):
   l2 = [[row[i] for row in l1] for i in range(len(l1[0]))]
   return l2


""" Starting Node """
node_state_i_input = [[1, 6, 7], [2, 0, 5], [4, 3, 8]]
node_state_i = []
print("Initial State Input:\n",node_state_i_input)
node_state_i = transpose(node_state_i_input,node_state_i)
# print("Initial State :\n",node_state_i)

""" Goal Node """
goal_state_input = [[1, 4, 7], [2, 5, 8], [3, 0, 6]]
goal_state = []
print("Goal Node Input:\n",goal_state_input)
goal_state = transpose(goal_state_input,goal_state)
# print("Goal Node :\n",goal_state)

res = node_state_i

location = None
closed_list = [node_state_i]
visited_nodes = []
backtrack = {}
back = []
indexes = []

""" Backtracking function to generate path taken to goal node """
def backtracking(child):
   back.append(child)
   parent = backtrack[str(child)]
   back.append(parent)
   while parent != node_state_i:
      parent = backtrack[str(parent)]
      back.append(parent)
   path = back[::-1]
   return path


""" Get '0' position """
def position(state):
   for i in range(0,3):
        for j in range(0,3):
            if state[i][j] == 0:
                pos = i,j
                return pos


""" Shift '0' to Right """
def ActionMoveRight(state,loc):
    i, j = loc
    temp = state[i][j]
    state[i][j] = state[i][j+1]
    state[i][j + 1] = temp
    return state


""" Shift '0' to Left """
def ActionMoveLeft(state,loc):
    i, j = loc
    temp = state[i][j]
    state[i][j] = state[i][j-1]
    state[i][j-1] = temp
    return state


""" Shift '0' to Up """
def ActionMoveUp(state,loc):
    i, j = loc
    temp = state[i][j]
    state[i][j] = state[i-1][j]
    state[i-1][j] = temp
    return state


""" Shift '0' to Down """
def ActionMoveDown(state,loc):
    i, j = loc
    temp = state[i][j]
    state[i][j] = state[i+1][j]
    state[i+1][j] = temp
    return state


val = 0
while True:
    popped_list = closed_list.pop(0)
    location = position(popped_list)
    i, j = location
    it = val
```

```
if popped_list != goal_state:
    if i+1 < 3:
        copied = copy.deepcopy(popped_list)
        moved = ActionMoveDown(copied, location)
        if moved not in visited_nodes:
            val += 1
            visited_nodes.append(moved)
            closed_list.append(moved)
            backtrack[str(moved)] = popped_list

            temp = []
            value = (val, it, transpose(moved,temp))
            indexes.append(value)

    if i-1 >= 0:
        copied = copy.deepcopy(popped_list)
        moved = ActionMoveUp(copied, location)
        if moved not in visited_nodes:
            val += 1
            visited_nodes.append(moved)
            closed_list.append(moved)
            backtrack[str(moved)] = popped_list

            temp = []
            value = (val, it, transpose(moved, temp))
            indexes.append(value)

    if j+1 < 3:
        copied = copy.deepcopy(popped_list)
        moved = ActionMoveRight(copied, location)
        if moved not in visited_nodes:
            val += 1
            visited_nodes.append(moved)
            closed_list.append(moved)
            backtrack[str(moved)] = popped_list

            temp = []
            value = (val, it, transpose(moved, temp))
            indexes.append(value)

    if j-1 >= 0:
        copied = copy.deepcopy(popped_list)
        moved = ActionMoveLeft(copied, location)
        if moved not in visited_nodes:
            val += 1
            visited_nodes.append(moved)
            closed_list.append(moved)
```

```
                backtrack[str(moved)] = popped_list

                temp = []
                value = (val, it, transpose(moved, temp))
                indexes.append(value)

    else:
        goal = []
        print("Goal State Reached:")
        goal = transpose(popped_list,goal)
        print(goal)

        path = backtracking(popped_list)
        break


""" Editing nodePath text file """
generate_path = []
with open("nodePath.txt",'r+') as file:
    file.truncate(0)
    for i in range(len(path)):
        generate_path = transpose(path[i], generate_path)
        for j in range(0,3):
            for k in range(0,3):
                file.write(str(generate_path[j][k]))
                file.write(str(' '))
        file.write('\n')
    file.close()
    pass


""" Editing Nodes text file """
all_visited_nodes = []
with open("Nodes.txt",'r+') as file:
    file.truncate(0)
    for i in range(len(visited_nodes)):
        all_visited_nodes = transpose(visited_nodes[i], all_visited_nodes)
        for j in range(0,3):
            for k in range(0,3):
                file.write(str(all_visited_nodes[j][k]))
                file.write(str(' '))
        file.write('\n')
    file.close()
    pass


""" Creating NodesInfo text file """
with open("NodesInfo.txt",'w') as file:
```

```
file.truncate(0)
file.write("NODE INDEX \t\t PARENT NODE INDEX \t\t\t\t NODE\n")
for i in range(len(indexes)):
   file.write(str(indexes[i][0]))
   file.write("\t\t|\t\t")
   file.write(str(indexes[i][1]))
   file.write("\t\t|\t")
   file.write(str(indexes[i][2]))
   file.write("\n")

file.close()
pass
```