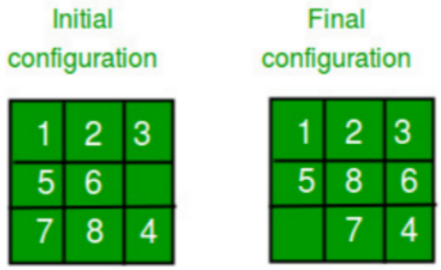
8 Puzzle Single Player Game (Breadth First Search)

Introduction

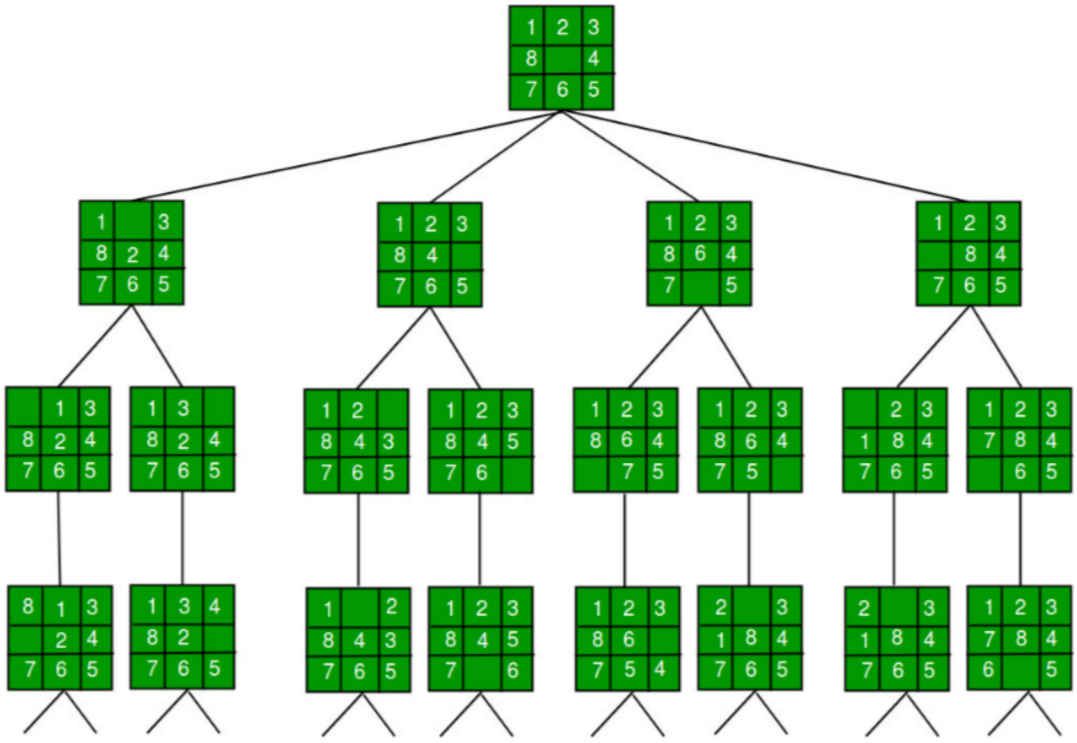
An instance of the n-puzzle game consists of a board holding $n^2 - 1$ distinct movable tiles, plus an empty space. The tiles are numbers from the set $1, . . , n^2 - 1$. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0. Given an initial state of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order $0, 1, . . , n^2 - 1$.

The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions {'Up', 'Down', 'Left', 'Right'}, one move at a time.

In this 8 puzzle problem is discussed. \ Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space. \ For example: \



Breadth First Search (BFS): We can perform a breadth-first search on state space (Set of all configurations of a given problem i.e. all states that can be reached from the initial state) tree. \



Algorithm Review

The searches begin by visiting the root node of the search tree, given by the initial state. Among other book-keeping details, three major things happen in sequence in order to visit a node:

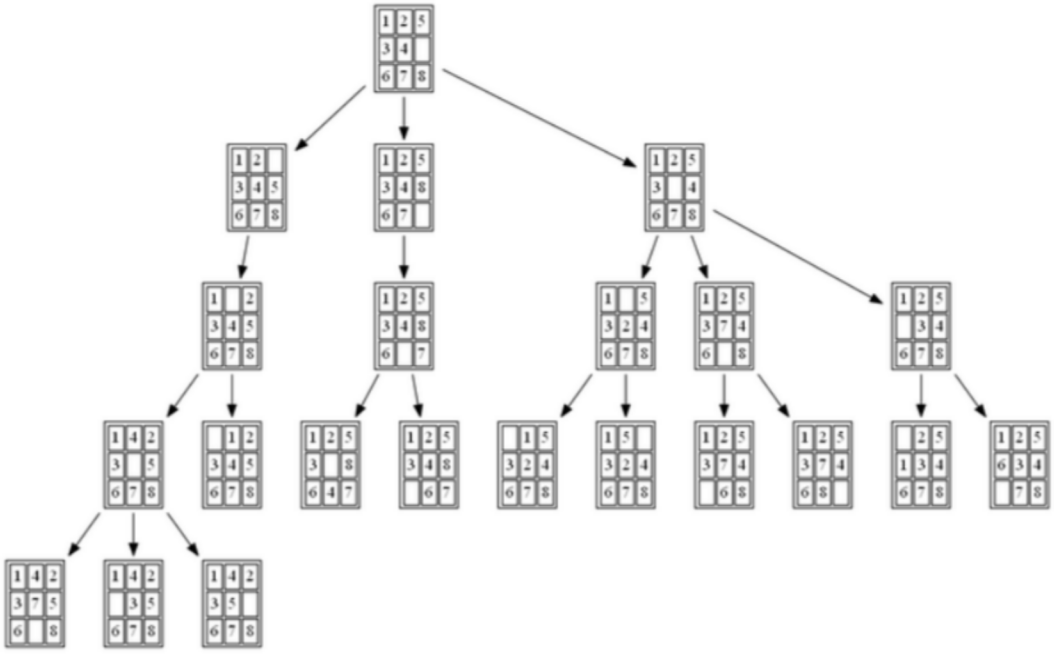
1. First, we remove a node from the frontier set.
2. Second, we check the state against the goal state to determine if a solution has been found.
3. Finally, if the result of the check is negative, we then expand the node. To expand a given node, we generate successor nodes adjacent to the current node, and add them to the frontier set. Note that if these successor nodes are already in the frontier, or have already been visited, then they should not be added to the frontier again.

Example: Breadth-First Search

Initial State: 1,2,5,3,4,0,6,7,8

1	2	5
3	4	
6	7	8

The nodes expanded by BFS (also the nodes that are in the fringe / frontier of the queue) are shown in the following figure:



Dear Students, in the code below there are few *TODO* task that you have to complete in this lab session.

In [1]:
```
#Import the necessary libraries
from time import time
from queue import Queue
```

In [5]:
```
#Creating a class Puzzle
class Puzzle:
    #Setting the goal state of 8-puzzle
    goal_state=[1,2,3,8,0,4,7,6,5]
    num_of_instances=0
    #constructor to initialize the class members
    def __init__(self,state,parent,action):
        self.parent=parent
        self.state=state
        self.action=action

        Puzzle.num_of_instances+= 1

    #function used to display a state of 8-puzzle
    def __str__(self):
        return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.sta

    #method to compare the current state with the goal state
    def goal_test(self):
        if self.state==self.goal_state:
            return True
        return False

    #static method to find the legal action based on the current board position
    @staticmethod
    def find_legal_actions(i,j):
        legal_action = ['U', 'D', 'L', 'R']
        if i == 0:
            # if row is 0 in board then up is disable
            legal_action.remove('U')
        elif i == 2:
            legal_action.remove('D')

        if j == 0:
            legal_action.remove('L')

        elif j == 2:
            legal_action.remove('R')

        return legal_action

    #method to generate the child of the current state of the board
    def generate_child(self):
    children=list()
        x = self.state.index(0)
        i = int(x / 3)
        j = int(x % 3)
        legal_actions=self.find_legal_actions(i,j)
        for action in legal_actions:
            new_state = self.state.copy()
            #if the legal action is UP
            if action == 'U':
                #Swapping between current index of 0 with its up element on the
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif action == 'D':
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif action == 'L':
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif action == 'R':
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]
            children.append(Puzzle(new_state,self,action))
        return children
    #method to find the solution
    def find_solution(self):
        solution = []
        solution.append(self.action)
        path = self
        while path.parent != None:
```

In [6]:
```
#method for breadth first search
def breadth_first_search(initial_state):
    start_node = Puzzle(initial_state, None, None)
    print("Initial state:")
    print(start_node)
    if start_node.goal_test():
        return start_node.find_solution()
    q = Queue()
    q.put(start_node)
    explored=[]
    while not(q.empty()):
        node=q.get()
        explored.append(node.state)
        children=node.generate_child()
        for child in children:
            if child.state not in explored:
                if child.goal_test():
                    return child.find_solution()
                q.put(child)
    return
```

In [7]:
```
#Start executing the 8-puzzle with setting up the initial state
#Here we have considered 3 initial state intitalized using state variable
state=[[1, 3, 4,
        8, 6, 2,
        7, 0, 5],

       [2, 8, 1,
        0, 4, 3,
        7, 6, 5],

       [2, 8, 1,
        4, 6, 3,
        0, 7, 5]]
#Iterate over number of initial_state
for i in range(0,3):
    Puzzle.num_of_instances=0
    #Set t0 to current time
    t0=time()
    bfs=breadth_first_search(state[i])
    #Get the time t1 after executing the breadth_first_search method
    t1=time()-t0
    print('BFS:', bfs)
    print('space:',Puzzle.num_of_instances)
    print('time:',t1)
    print()
print('-----------------------------------------')
```

```
Initial state:
[1, 3, 4]
[8, 6, 2]
[7, 0, 5]
BFS: ['U', 'R', 'U', 'L', 'D']
space: 66
time: 0.0009648799896240234

Initial state:
[2, 8, 1]
[0, 4, 3]
[7, 6, 5]
BFS: ['U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 591
time: 0.00656437873840332

Initial state:
[2, 8, 1]
[4, 6, 3]
[0, 7, 5]
BFS: ['R', 'U', 'L', 'U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 2956
time: 0.048323631286621094

-----------------------------------------
```

BONUS TASK: Now, perform the following task before submitting the assignment:

Show the path in traversing the BFS algorithm of each state from intial_state to goal state.

In [ ]:

In [ ]: