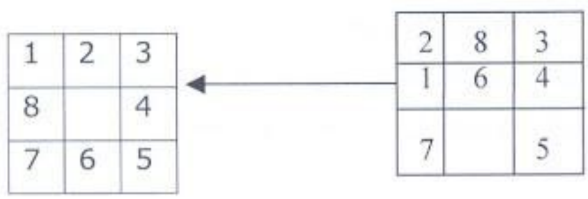
6. 8-Puzzle Game

29/08/2019

INTRODUCTION

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. On each grid square is a tile, expect for one square which remains empty. Thus, there are eight tiles in the 8-puzzle and 15 tiles in the 15-puzzle. A tile that is next to the empty grid square can be moved into the empty space, leaving its previous position empty in turn. Tiles are numbered, 1 through 8 for the 8-puzzle, so that each tile can be uniquely identified.



Goal Initial

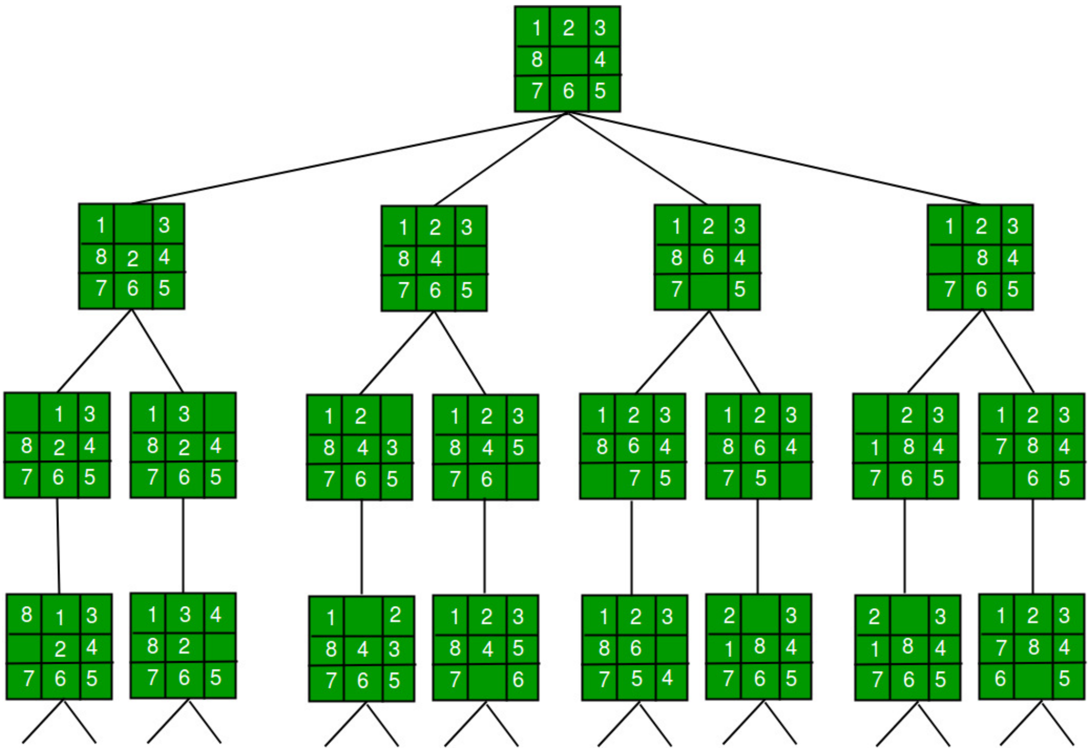
The aim of the puzzle is to achieve a given configuration of tiles from a given (different) configuration by sliding the individual tiles around the grid as described above.

ALGORITHM:

Step 1: Evaluate the initial state. If it is goal state then exit else make the current state as initial state

Step 2: Repeat these steps until a solution is found or current state does not change
1. Let ‘target’ be a state such that any successor of the current state will be better than it.
2. For each operator that applies to the current state
    a. apply the new operator and create a new state
    b. evaluate the new state
    c. if this state is goal state then quit else compare with ‘target’
    d. if this state is better than ‘target’, set this state as ‘target’
    e. if target is better than current state set current state to Target

Step 3: Exit



PYTHON IMPLEMENTATION

```
def find_pos(board, elem):
    '''Finds the position of 'elem' in the board'''
    for i, row in enumerate(board):
        try:
            return i, row.index(elem)
        except ValueError:
            pass


def manhattan_distance(start_state, goal_state):
    '''Returns the Manhattan Distance between to states of the board'''
    distance = 0
    for elem in range(1, 9):
        i, j = find_pos(start_state, elem)
        k, l = find_pos(goal_state, elem)
        distance += abs(i-k) + abs(j-l)
    return distance


def make_move(current_state, move, i, j):
    '''Returns the new board position after implementing the passed move'''
    temp_state = copy.deepcopy(current_state)

    if move == 1:  # Move up
        temp_state[i][j], temp_state[i-1][j] = \
```

```python
            temp_state[i-1][j], temp_state[i][j]
    elif move == 2:  # Move down
        temp_state[i][j], temp_state[i+1][j] = \
            temp_state[i+1][j], temp_state[i][j]
    elif move == 3:  # Move left
        temp_state[i][j], temp_state[i][j-1] = \
            temp_state[i][j-1], temp_state[i][j]
    elif move == 4:  # Move right
        temp_state[i][j], temp_state[i][j+1] = \
            temp_state[i][j+1], temp_state[i][j]

    return temp_state


def tile_ordering(current_state, goal_state, move, x, y):
    '''Returns the Manhattan Distance between the current state and the
    goal state after the proposed move is implemented'''
    temp_state = make_move(current_state, move, x, y)
    distance = manhattan_distance(temp_state, goal_state)
    return distance


def steepest_ascent_hill_climb(start_state, goal_state, former_move):
    '''Solves the 8 puzzle using Steepest Ascent Hill Climbing algorithm'''

    # Store the value of the evaluation function for each move
    heuristic_values = [100, 100, 100, 100]

    moves = {1: 'up', 2: 'down', 3: 'left', 4: 'right'}
    i, j = find_pos(start_state, 0)

    print("\nEvaluating all possibilities...")
    if i > 0 and former_move != 2:
        heuristic_values[0] = tile_ordering(start_state, goal_state, 1, i, j)
        print("Checking child (moving 0 up): %d" % (heuristic_values[0]))
    if i < 2 and former_move != 1:
        heuristic_values[1] = tile_ordering(start_state, goal_state, 2, i, j)
        print("Checking child (moving 0 down): %d" % (heuristic_values[1]))
    if j > 0 and former_move != 4:
        heuristic_values[2] = tile_ordering(start_state, goal_state, 3, i, j)
        print("Checking child (moving 0 left): %d" % (heuristic_values[2]))
    if j < 2 and former_move != 3:
        heuristic_values[3] = tile_ordering(start_state, goal_state, 4, i, j)
        print("Checking child (moving 0 right): %d" % (heuristic_values[3]))

    min_val, min_idx = heuristic_values[0], 0
    for idx, val in enumerate(heuristic_values):
        if val < min_val:
```

```python
                min_val, min_idx = val, idx

        next_state = make_move(start_state, min_idx+1, i, j)
        print("\nNext state (moved %s):" % (moves[min_idx+1]))
        print_board(next_state)
        if min_val == 0:
            print("Reached goal state. Quitting...")
        else:
            steepest_ascent_hill_climb(next_state, goal_state, min_idx+1)


def print_board(board):
    for row in board:
        print(' '.join(map(lambda x: str(x) if x != 0 else ' ', row)))


def main():
    initial_state, goal_state = [], []

    print("Enter initial board configuration (0 represents empty cell): ")
    print("Example: 1 2 3")
    print("         4 0 5")
    print("         6 7 8")
    for i in range(3):
        initial_state.append(list(map(int, input().split())))

    print("Enter goal board configuration (0 represents empty cell): ")
    print("Example: 1 2 3")
    print("         4 6 5")
    print("         7 8 0")
    for i in range(3):
        goal_state.append(list(map(int, input().split())))

    print("Initial board configuration:")
    print_board(initial_state)

    try:
        steepest_ascent_hill_climb(initial_state, goal_state, 0)
    except RecursionError:
        print("Solution could not be found")


if __name__ == '__main__':
    main()
```

ADVANTAGES/DISADVANTAGES OF STEEPEST ASCENT HILL CLIMBING

Advantages:

- Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management.
- It is also helpful to solve pure optimization problems where the objective is to find the best state according to the objective function.
- It requires much less conditions than other search techniques.

Disadvantages:

- The question that remains on hill climbing search is whether this hill is the highest hill possible. Unfortunately without further extensive exploration, this question cannot be answered. This technique works but as it uses local information that's why it can be fooled.
- It may not reach the goal state in some situations.