8 Puzzle Solver

Team Members:-

Karim Mohamed Moustafa (45)
Kamal Abd El-Aziz Kamal (46)

Problem Statement:-

Given an initial state of the board, the 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 0,1,2,3,4,5,6,7,8.

The Search Algorithms:-

1) BFS:-

BFS search

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

2)DFS:-

DFS search

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.push(neighbor)

    return FAILURE
```

Taken from the edX course ColumbiaX: CSMM.101x Artificial Intelligence (AI)

3)A*:-

A* search

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) + h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

In the A* Search we used 2 heuristics:-

1. Manhattan Distance:

$$h = abs(current\_cell.x - goal.x) + abs(current\_cell.y - goal.y)$$

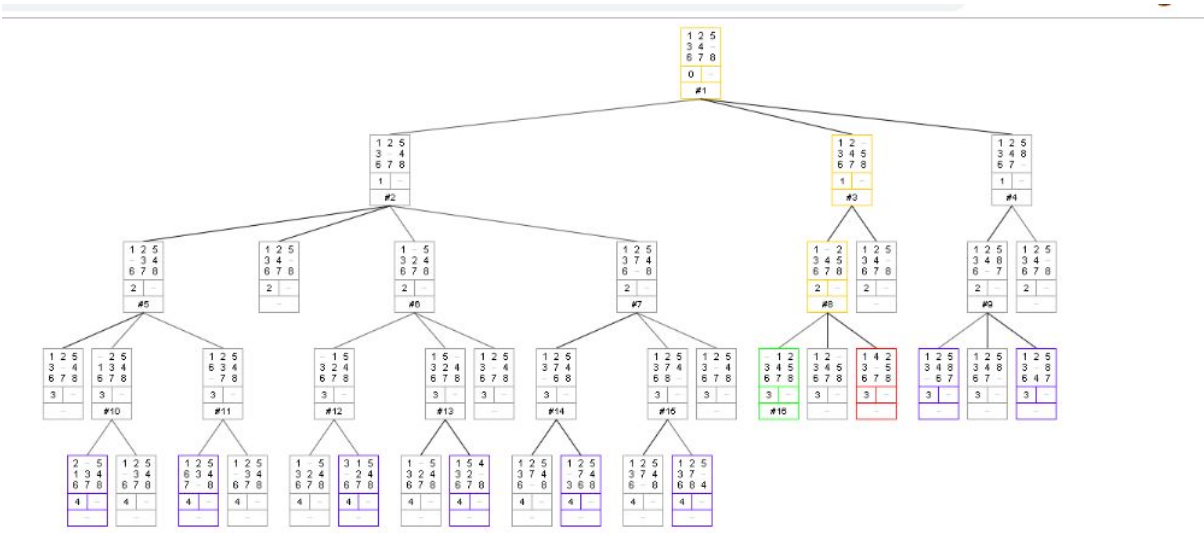
2. Euclidean Distance:

$$h = sqrt((current\_cell.x - goal.x)^2 + (current\_cell.y - goal.y)^2)$$

8 Puzzle problem sample:-

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

1. Applying BFS:-
    a. Path to goal And Nodes Expanded:-



    b. Cost of Path:-

    Cost = 3 : number of changed configurations (depth).

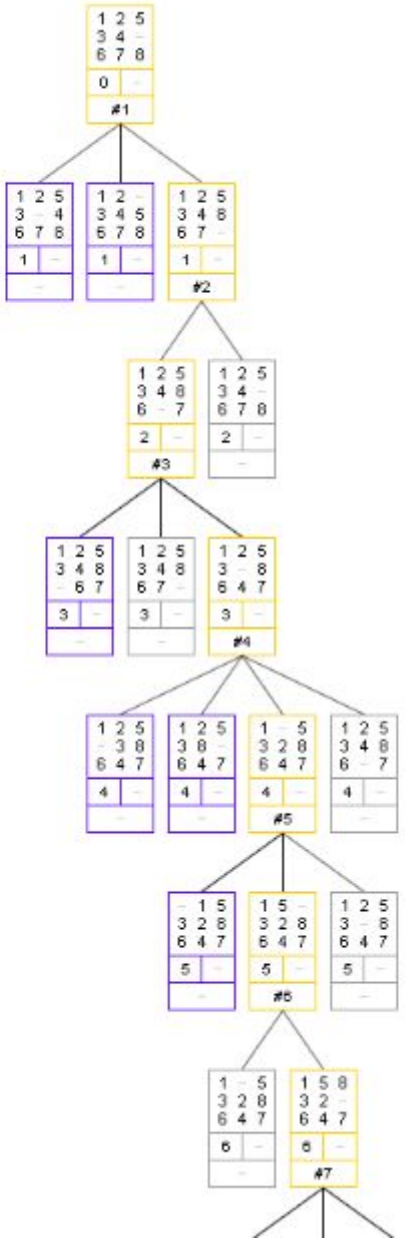
    c. Search Depth:-

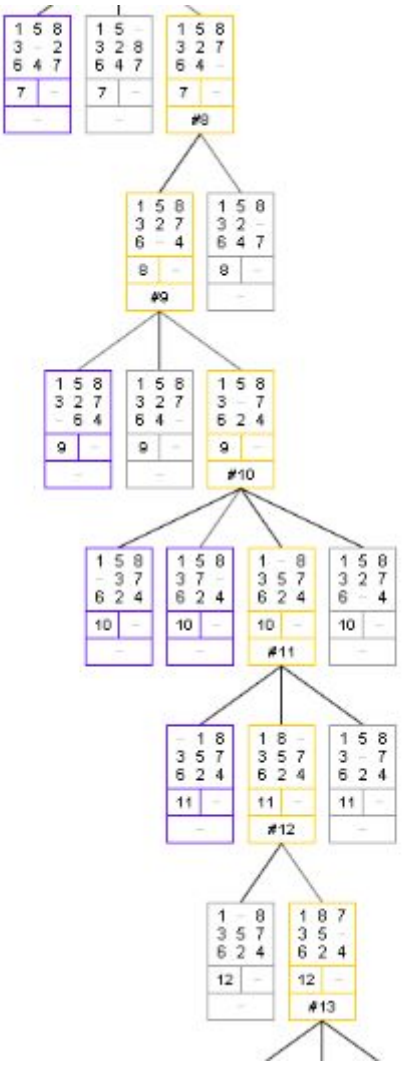
    Depth = 3 (root depth = 0)

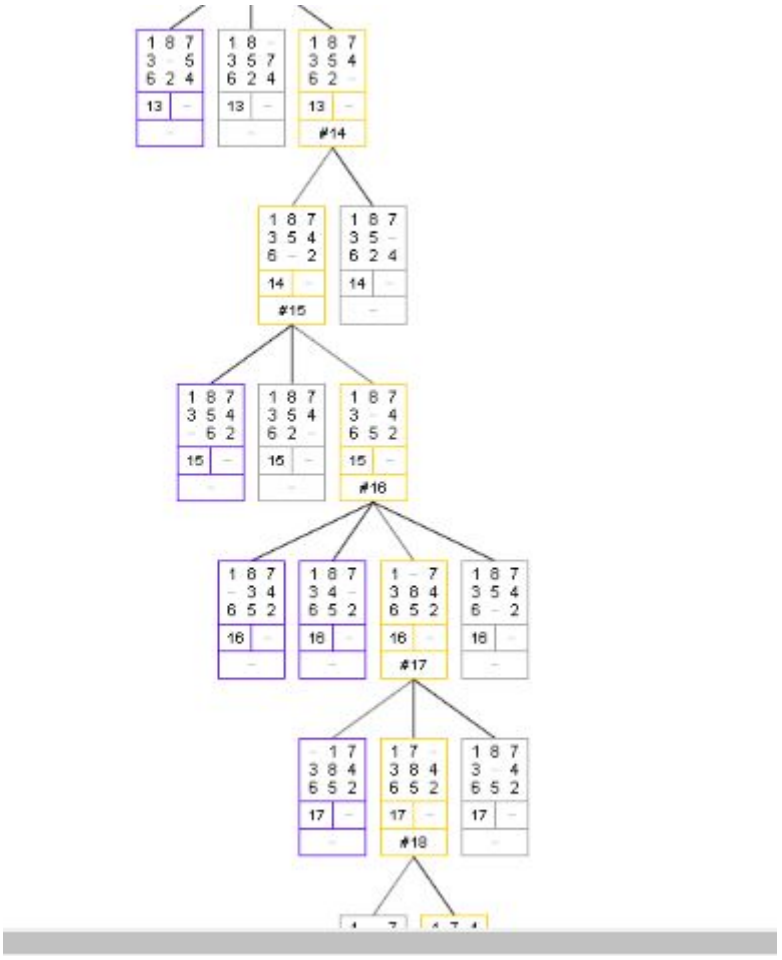
    d. Running Time:-

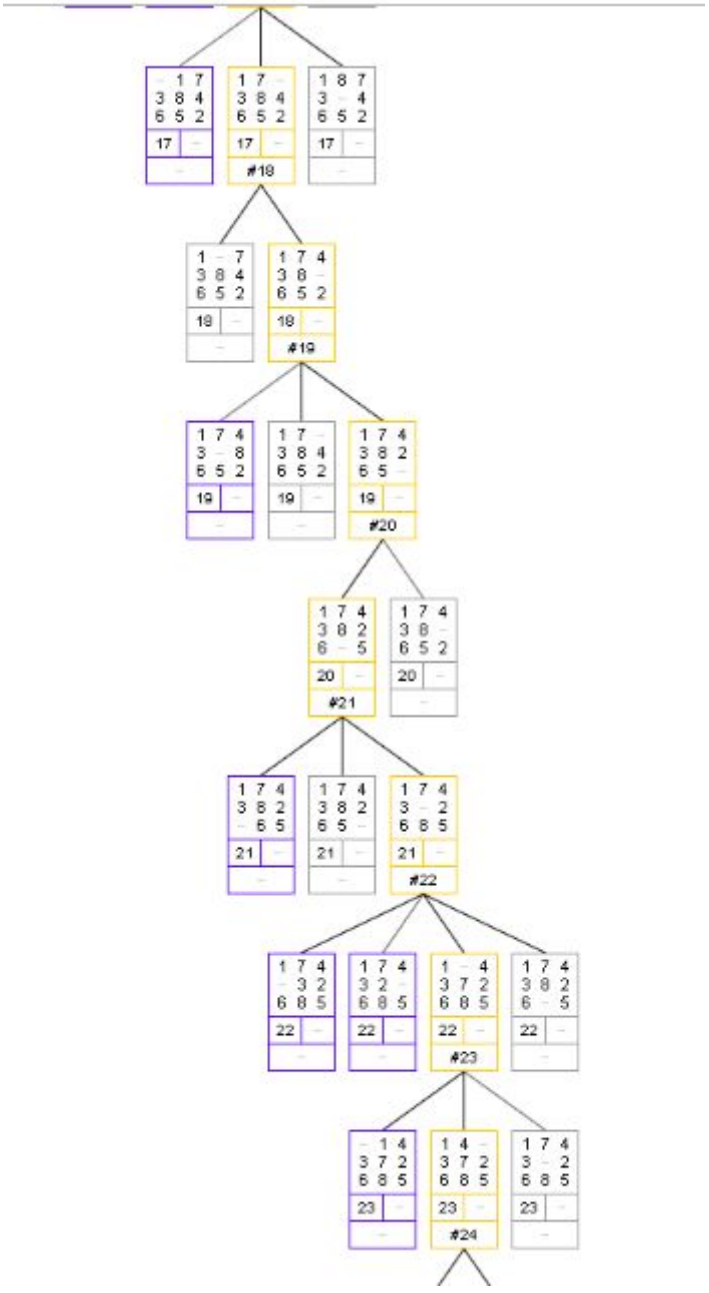
    BFS time 2.0 milliseconds

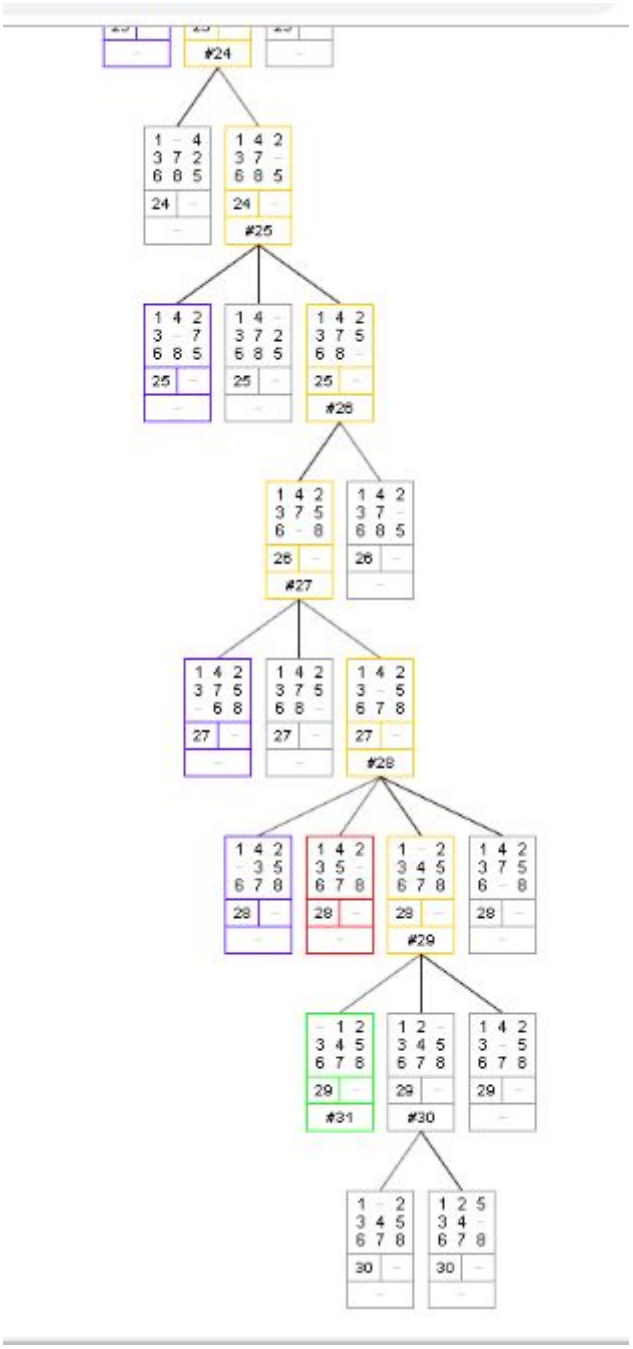
2. Applying DFS:-
    a. Path to goal And Nodes Expanded:-




#8
#9
#10
#11
#12
#13


1 8 7
3 - 5
6 2 4
13
1 8 -
3 5 7
6 2 4
13
1 8 7
3 5 4
6 2 -
13
#14
1 8 7
3 5 4
6 - 2
14
#15
1 8 7
3 5 -
6 2 4
14
1 8 7
3 5 4
- 6 2
15
1 8 7
3 5 4
6 2 -
15
1 8 7
3 - 4
6 5 2
15
#16
1 8 7
- 3 4
6 5 2
16
1 8 7
3 4 -
6 5 2
16
1 - 7
3 8 4
6 5 2
16
#17
1 8 7
3 5 4
6 - 2
16
- 1 7
3 8 4
6 5 2
17
1 7 -
3 8 4
6 5 2
17
#18
1 8 7
3 - 4
6 5 2
17


17
#18
18
#19
19
#20
20
#21
21
#22
22
#23
23
#24



b. Cost of Path:-

Cost = 30 : number of changed configurations (depth).

After setting maxDepth:-

Cost = 3 (depth)

c. Search Depth:-

Depth = 30 (root depth = 0)

After setting maxDepth:-

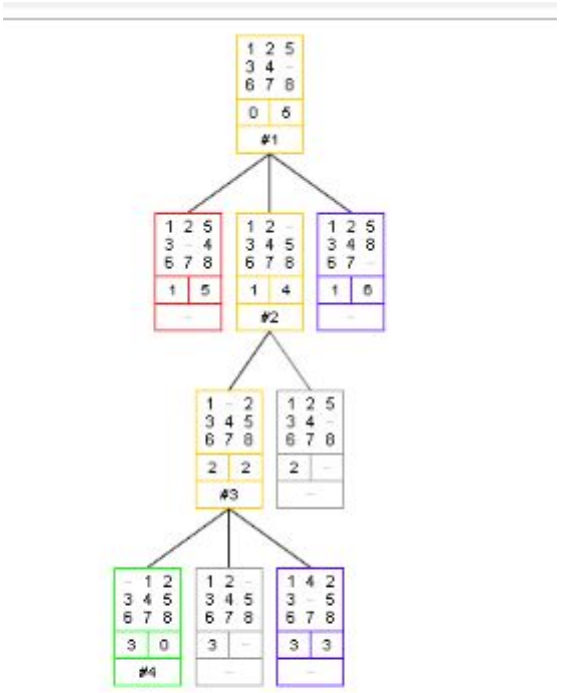
Depth = 3 (depth)

d. Running Time:-

DFS time 102.0 milliseconds

3. Applying A* with Euclidean Search:-

a. Path to goal And Nodes Expanded:-



b. Cost of Path:-

Cost = 17 : number of changed configurations.

c. Search Depth:-

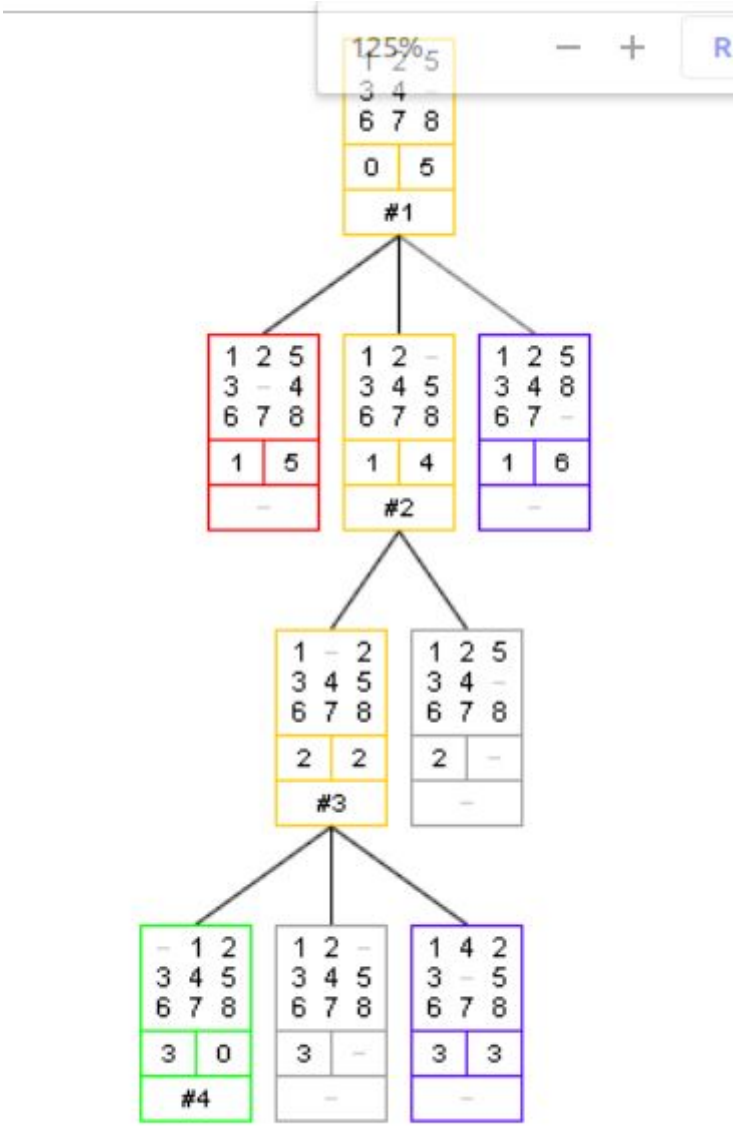
Depth = 3 (root depth = 0)

d. Running Time:-

Euclidean time 3.0 milliseconds

4. Applying A* with Manhattan Search:-

a. Path to goal And Nodes Expanded:-



b. Cost of Path:-

Cost = 18 : number of changed configurations (depth).

c. Search Depth:-

Depth = 3 (root depth = 0)

d. Running Time:-

Manhattan time 3.0 milliseconds