Assignment 1 - Using Informed and Uninformed Search Algorithms to Solve 8-Puzzle

Our code consists of two main classes:

1- State

A class represents each state of the 8-Puzzle array

Main Function:

a. getNeighbors :
   returns a list of neighbors states of this state instance
   uses sample function to get the blank cell position then checks for the availability of the 4 neighbors (Up – Down – Left – Right)
b. equals:

   takes a State Object and returns if the Object is equal to this current state or not

2- Path

A class represents Sequence of States of the 8-Puzzle towards the goal

Contains a (Linked List) of states .

Main Function:

c. addState:

   adds a State to the current path

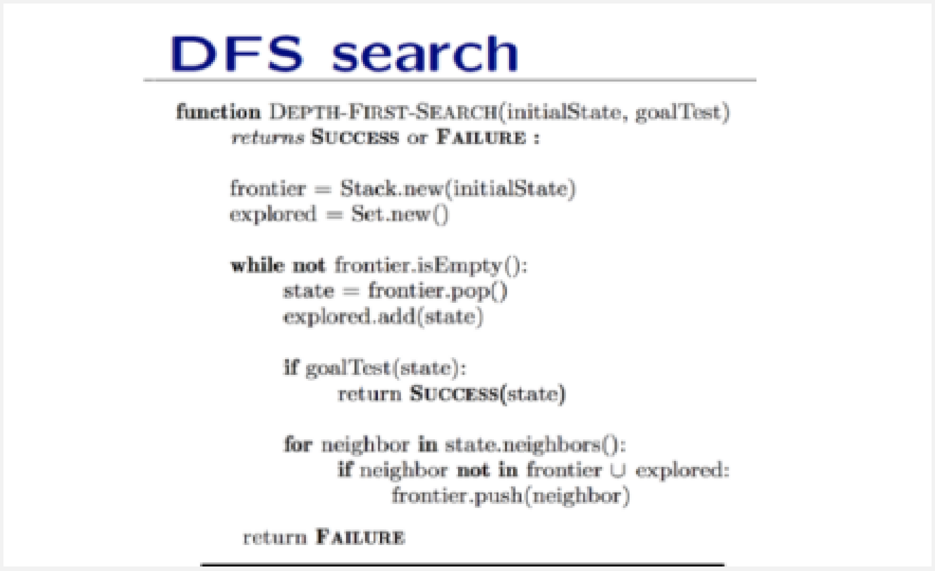
d. contains:

   takes a state as parameter and returns if this path contains that state or not

   to avoid Loops

Then the three main search algorithms

1- DFS

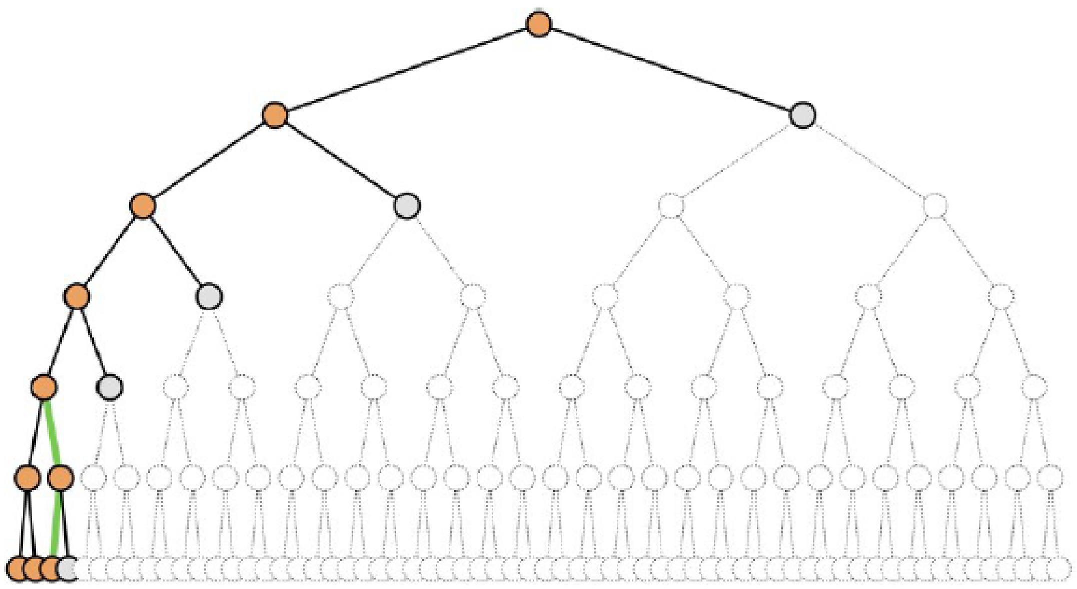


DFS search

```
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
```

Using a Stack of the Path Object as Frontier and a Set for the Explored states

It Searches branch by branch



Searches branch by branch ...

(a) path to goal:

Using the Search function of the DFS Class it returns the Goal Path

(b) cost of path

Cost = the depth of goal path – 1

(c) nodes expanded

Nodes expanded always equals to (#of visited States) + #of States still in the Frontier

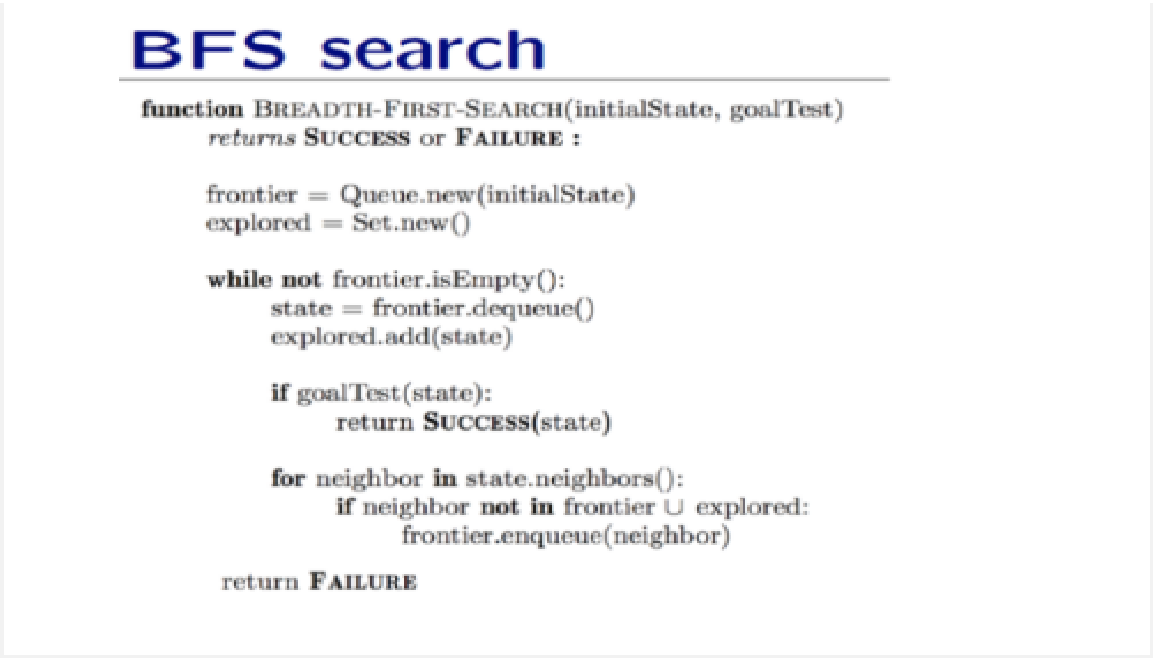
(d) search depth

The largest path depth that was put inside the Stack

(e) running time

Exponential time

2- BFS

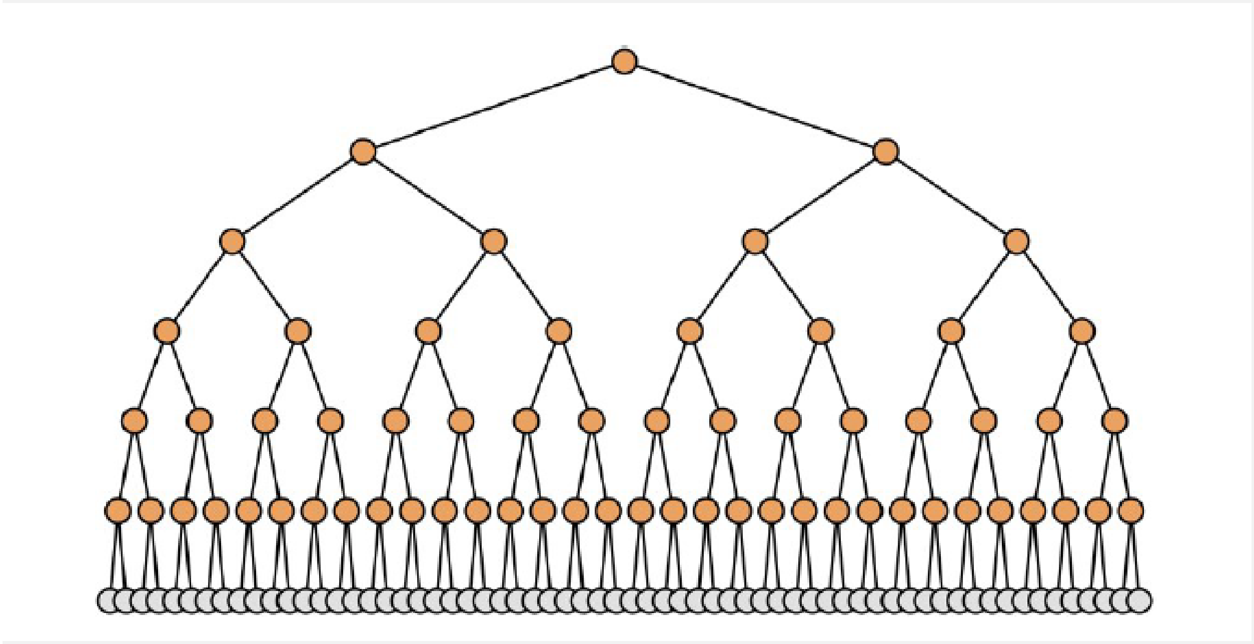


BFS search

```
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
```

Using a Queue of the Path Object as Frontier and a Set for the Explored states

Searches level by level



(a) path to goal:

Using the Search function of the BFS Class it returns the Goal Path

(b) cost of path

Cost = the depth of goal path – 1

(c) nodes expanded

Nodes expanded always equals to (#of visited States) + #of States still in the Frontier

It is a very huge number as the branching factor = 4 so the expanded nodes estimation

would be equal 4 pow the depth

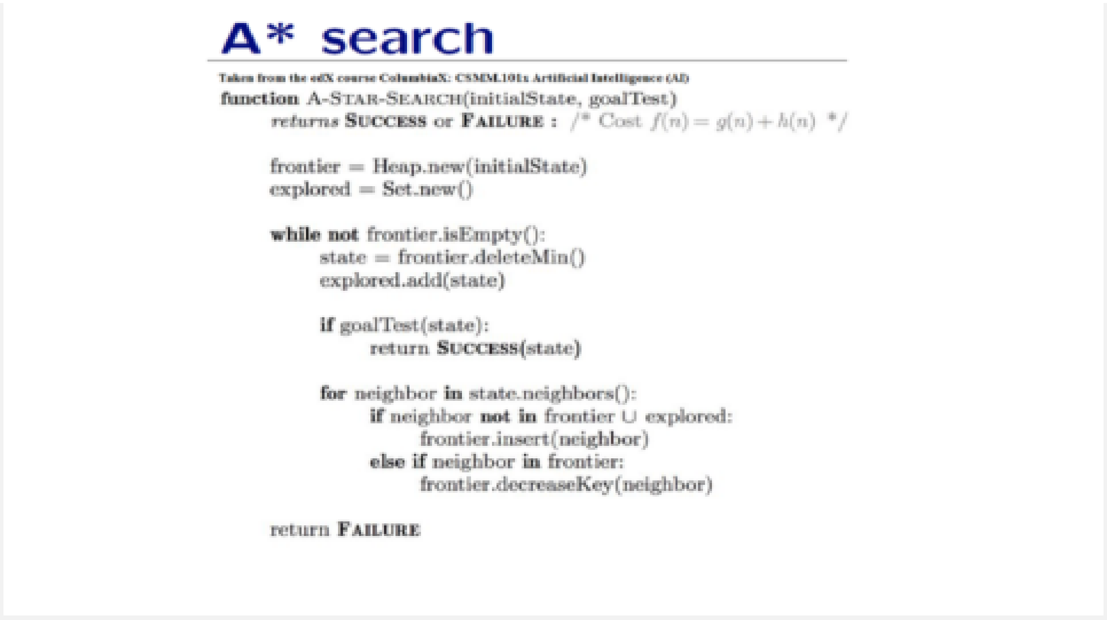
(d) search depth

As it searches level by level so the search depth will always be the depth of the goal path

(e) running time

Exponential time

3- A*

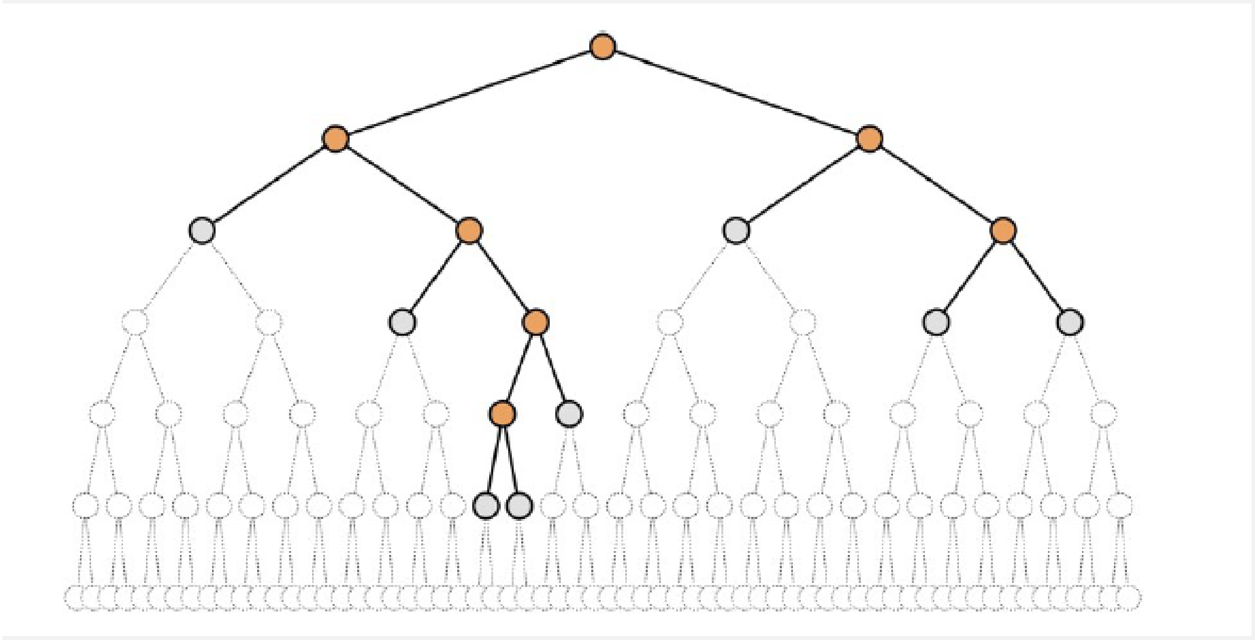

A* search

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

```
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
```

Using a Priority Queue of the Path Object as Frontier and a Set for the Explored states

Searches Layer by Layer but organized by path cost.



(a) path to goal:

Using the Search function of the A* Class it returns the Goal Path

(b) cost of path

Cost = the depth of goal path – 1

(c) nodes expanded

Nodes expanded always equals to (#of visited States) + #of States still in the Frontier

(d) search depth

The largest path depth that was put inside the Priority Queue

(e) running time

depends on the heuristic. In the worst case of an unbounded search space

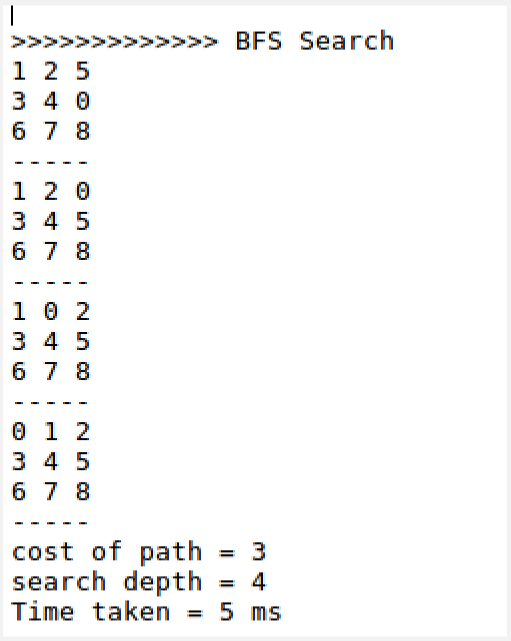
the time is exponential

Screen Shots:

initial state = { 1, 2, 5, 3, 4, 0, 6, 7, 8 }

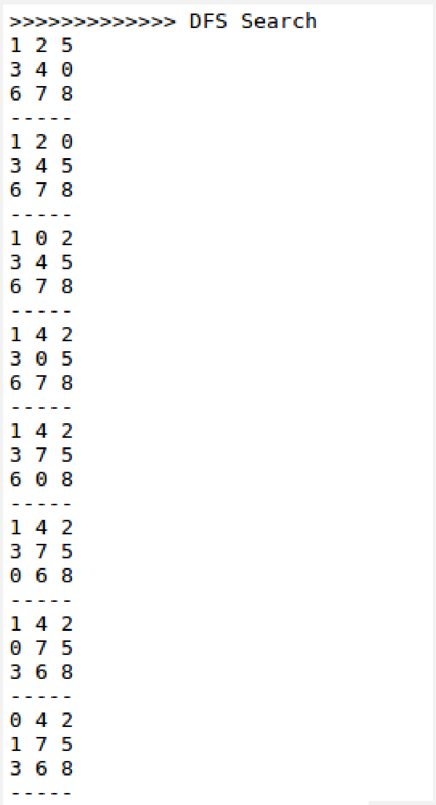
Goal State = { 0, 1, 2, 3, 4, 5, 6, 7, 8 }

BFS output:

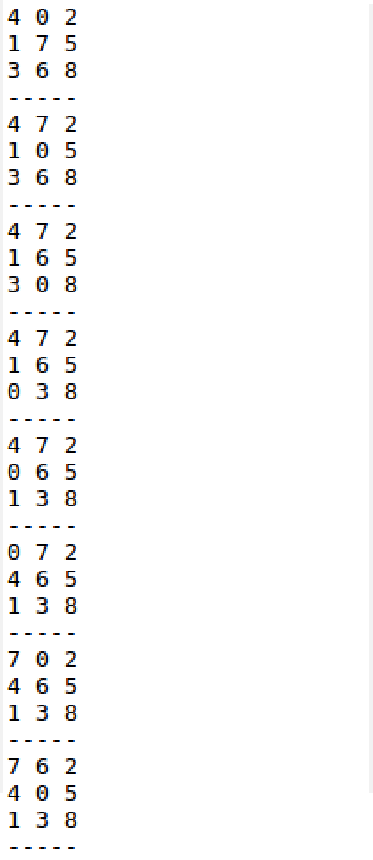


```
>>>>>>>>>>>>> BFS Search
1 2 5
3 4 0
6 7 8
-----
1 2 0
3 4 5
6 7 8
-----
1 0 2
3 4 5
6 7 8
-----
0 1 2
3 4 5
6 7 8
-----
cost of path = 3
search depth = 4
Time taken = 5 ms
```

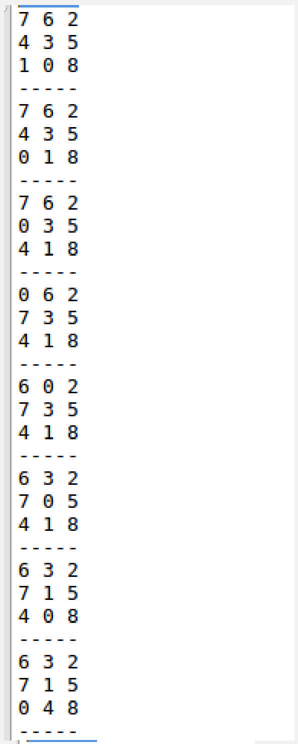
DFS outputs:



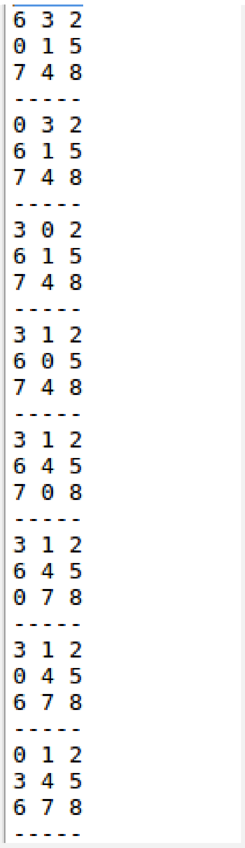
```
>>>>>>>>>>>>>> DFS Search
1 2 5
3 4 0
6 7 8
-----
1 2 0
3 4 5
6 7 8
-----
1 0 2
3 4 5
6 7 8
-----
1 4 2
3 0 5
6 7 8
-----
1 4 2
3 7 5
6 0 8
-----
1 4 2
3 7 5
0 6 8
-----
1 4 2
0 7 5
3 6 8
-----
0 4 2
1 7 5
3 6 8
-----
```



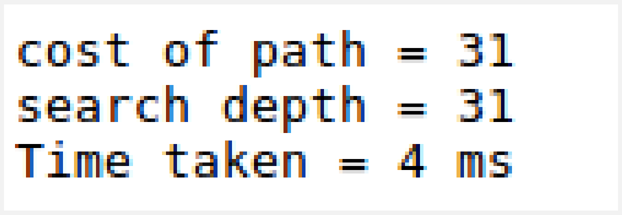
```
4 0 2
1 7 5
3 6 8
-----
4 7 2
1 0 5
3 6 8
-----
4 7 2
1 6 5
3 0 8
-----
4 7 2
1 6 5
0 3 8
-----
4 7 2
0 6 5
1 3 8
-----
0 7 2
4 6 5
1 3 8
-----
7 0 2
4 6 5
1 3 8
-----
7 6 2
4 0 5
1 3 8
-----
```



```
7 6 2
4 3 5
1 0 8
-----
7 6 2
4 3 5
0 1 8
-----
7 6 2
0 3 5
4 1 8
-----
0 6 2
7 3 5
4 1 8
-----
6 0 2
7 3 5
4 1 8
-----
6 3 2
7 0 5
4 1 8
-----
6 3 2
7 1 5
4 0 8
-----
6 3 2
7 1 5
0 4 8
-----
```

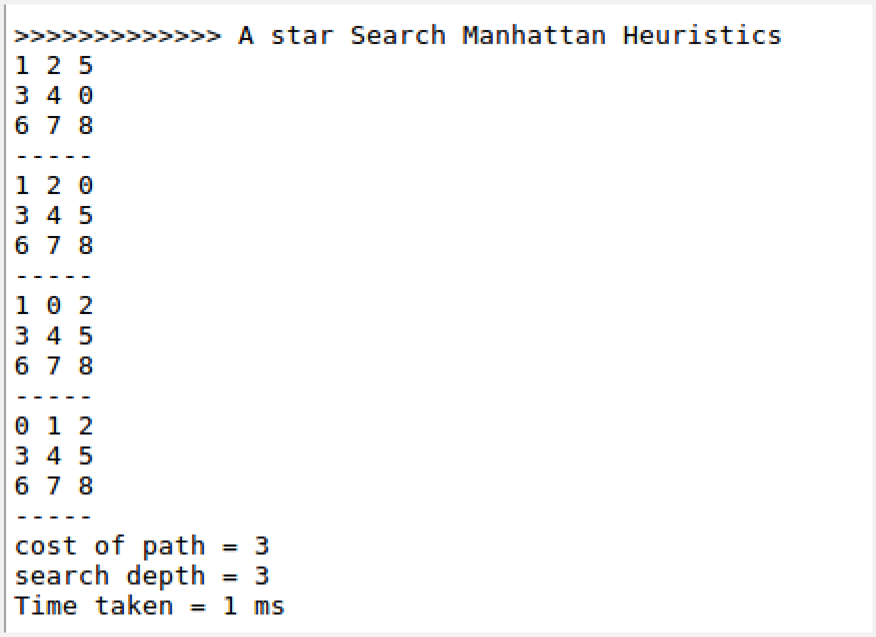


```
6 3 2
0 1 5
7 4 8
-----
0 3 2
6 1 5
7 4 8
-----
3 0 2
6 1 5
7 4 8
-----
3 1 2
6 0 5
7 4 8
-----
3 1 2
6 4 5
7 0 8
-----
3 1 2
6 4 5
0 7 8
-----
3 1 2
0 4 5
6 7 8
-----
0 1 2
3 4 5
6 7 8
-----
```

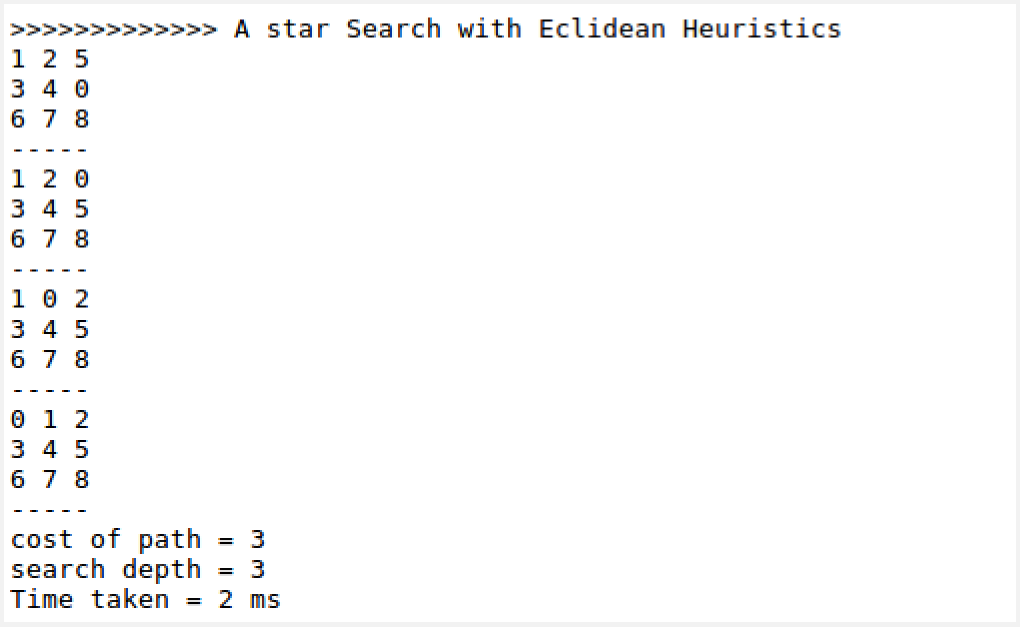


```
cost of path = 31
search depth = 31
Time taken = 4 ms
```

A star outputs :



```
>>>>>>>>>>>>> A star Search Manhattan Heuristics
1 2 5
3 4 0
6 7 8
-----
1 2 0
3 4 5
6 7 8
-----
1 0 2
3 4 5
6 7 8
-----
0 1 2
3 4 5
6 7 8
-----
cost of path = 3
search depth = 3
Time taken = 1 ms
```



```
>>>>>>>>>>>>> A star Search with Eclidean Heuristics
1 2 5
3 4 0
6 7 8
-----
1 2 0
3 4 5
6 7 8
-----
1 0 2
3 4 5
6 7 8
-----
0 1 2
3 4 5
6 7 8
-----
cost of path = 3
search depth = 3
Time taken = 2 ms
```