ASSIGNMENT.

Title:- A* algorithm.

Problem statement:

Solve 8-puzzle problem using A* algorithm. Assume any initial configuration and define goal configuration clearly.

Objectives:

- To learn and understand use and need of A* algorithm.
- To apply A* algorithm to real time problem.
- To implement A* algorithm with suitable programming language.

Outcomes:-

We will be able to:-

- learn about A* algorithm.
- apply A* algorithm to gaming problem.
- implement A* algorithm using Prolog/Python/Java.

Hardware and software requirements:

- OS: Fedora 20 / Ubuntu (64-bit)
- RAM: 4GB
- HDD: 500 GB
- Eclipse IDE / Jupyter Notebook.
- Python libraries

Theory :

- A* is one of the most popular heuristic search algorithm for finding paths in a graph.
- It is a really smart algorithm which separates it from other algorithms

- Consider a square grid having many obstacles and ~~use~~ we are given a starting cell and a target cell,
- We want to reach targed cell from starting cell as quickly as possible.
- What A* algorithm does is ~~that~~ at each step, it picks the node according to a '-f' value which is a parameter equal to sum of other two parameters '-g' and 'h'.
- At each step, it picks the node/cell having least 'f' and process that node/cell.

- We define 'g' and 'h' simply as follows:

  g : the movement cost to move from the starting point to a given square on the grid following the path generated to get there.

  h = the estimated movement cost to move from that given square on the grid to the final destination. This often refers to as heuristic which is nothing but a kind of smart guess

- We really dont know the actual division until we find the path because all sorts of things can happen in the way

Algorithm:

```
1. Initialize the open list.
2. Initialize the cleared list.
      put the starting node on the open list.
3. While the open list is not empty.
    1. Find the node with the least f on the open list.
       Call it 'q'.
    2. Pop 'q' off the open list.
    3. generate 'q's' successors.
    4. for each successors
        1. if successor is the goal, stop search
           successor.g = q.g + distance (successor.g)
           successor.h = distance from goal to successor
           successor.f = successor.g + successor.h.
        2. if a node with the same position as
           successor in in the OPEN list which has
           a lower 'f' than successor, skip this successor.
        3. if a node with the same position as successor
           is in the CLOSED list which has a lower
           'f' that successor, skip this successor otherwise
           add the node to the open list.
    5. End for.
    6. push q on the closed list
4. End while.
```

# Test cases:

Initial configuration:			Final configuration:		
1	2	x	1	2	3
4	5	3	4	5	6
7	8	6	7	8	x

Output:

The puzzle was solved in 18 moves

Conclusion:

We successfully implemented A* algorithm for 8. puzzle problem.