Date ___/___/_____

Assignment - B1

* Title:- A* Algorithm.

* Date of completion:-

* 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 using suitable programming language.

* Outcomes:- Students will be able to :-
— learn about A* algorithm.
— apply A* algorithm to gaming problem.
— Implement A* algorithm using Python / Java / Prolog.

* Software and Hardware Requirements
— OS : Fedora 20 / Ubuntu (64-bit)
— RAM : 4GB
— HDD : 500GB
— Eclipse IDE
— Java JDK V.1.8.
— Python libraries.

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

Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach target cell from the starting cell as quickly as possible. What A* algorithm does is at each step, it picks the node according to a value '-f' 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 possible
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 is often referred to as the heuristic which is nothing but a kind of smart guess.

We really don't know the actual division until we find the path because all sorts of things can be in the way.

* Algorithm :-

```
1.) Initialise the open list.
2.) Initialise the closed list.
      put the starting node on the open list.
3.) While the open list is not empty
      3.1.) Find the node with the least "f" on the open list. Call it "q".
      3.2.) Pop 'q' off open list.
      3.3.) Generate 'q' successors.
      3.4.) For each successor.
            3.4.1.) If successor is the goal, stop search successor.
```

g = q.g + distance (successor.q).
successor.h = distance from goal to successor.
successor.f = successor.g + successor.h.

3.4.2) If a node with the same position as successor is in the "open" list which has a lower 'f' than successor, skip the successor.
3.4.3) If a node with the same position as successor is in the "closed" list which has a lower "f" than successor, skip the successor otherwise, and the node to the open list.
3.5) End for
3.6) Push q on the closed list.
4) End while.

* Test case :-

Initial configuration

1	2	X
4	5	3
7	8	6

Final configuration

1	2	3
4	5	6
7	8	X

* Output :- The puzzle was solved in 18 moves.

* Conclusion :- We successfully implemented A* algorithm for 8-puzzle problem.