Assignment No-02(Group A)

Title: Implementation of A* Algorithm for any game search problem.

Objectives:-

Student should be able to implement A* algorithm on the 8-Puzzle problem

Prerequisites:

Concept of Heuristic function, state space

Problem Statement:-

Implement A star Algorithm on the 8-Puzzle problem

Software and Hardware requirements:-

1. **Operating system:** Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10,
2. **RAM-** 2GB RAM (4GB preferable)
3. You have to install **Python3** or higher version

Theory-

A* (pronounced as "A star") is a computer algorithm that is widely used in path finding and graph traversal.

However, the A* algorithm introduces a heuristic into a regular graph-searching algorithm, essentially planning ahead at each step so a more optimal decision is made.

A* is an extension of Dijkstra's algorithm with some characteristics of breadth-first search (BFS).

A* uses a function f(n) that gives an estimate of the total cost of a path using that node. Therefore, A* is a heuristic function, which differs from an algorithm in that a heuristic is more of an estimate and is not necessarily provably correct.

A* expands paths that are already less expensive by using this function:

$$f(n)=g(n)+h(n)$$

where,

f(n) = total estimated cost of path through node n

g(n) = cost so far to reach node n

h(n) = estimated cost from n to goal. This is the heuristic part of the cost function, so it is like a guess.

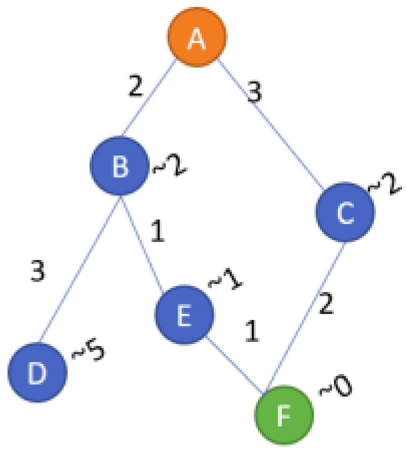
The calculation of h(n) can be done in various ways:

The Manhattan distance (explained below) from node n to the goal is often used. This is a standard heuristic for a grid.

If h(n) = 0, A* becomes Dijkstra's algorithm, which is guaranteed to find a shortest path.

The heuristic function must be **admissible**, which means it can never overestimate the cost to reach the goal. Both the Manhattan distance and h(n)= 0 are admissible.

Consider the following graph with edge cost (distance between two nodes like A-B, C-F, etc) and the heuristics value to reach to the target node.

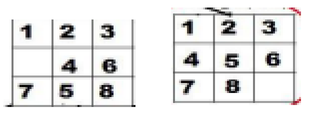


8-Puzzle using A* Algorithms

N-Puzzle or sliding puzzle is a popular puzzle that consists of N tiles where N can be 8, 15, 24, and so on.

In our example N = 8. The puzzle is divided into sqrt(N+1) rows and sqrt(N+1) columns

Eg. 15-Puzzle will have 4 rows and 4 columns and an 8-Puzzle will have 3 rows and 3 columns. The puzzle consists of N tiles and one empty space where the tiles can be moved. Start and Goal configurations (also called state) of the puzzle are provided.



1	2	3
	4	6
7	5	8

1	2	3
4	5	6
7	8	

Inital State **Goal State**

Rules for solving the puzzle.

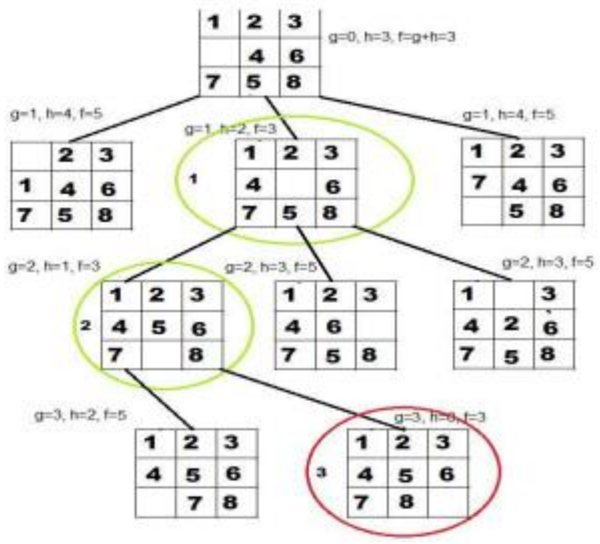
Instead of moving the tiles in the empty space, we can visualize moving the empty space in place of the tile, basically swapping the tile with the empty space. The empty space can only move in four directions viz. 1.Up 2.Down 3.Right 4. Left

The empty space cannot move diagonally and can take only one step at a time.

In our 8-Puzzle problem, we can define the h-score as the number of misplaced tiles by comparing the current state and the goal state or summation of the Manhattan distance between misplaced nodes. g-score will remain as the number of nodes traversed from a start node to get to the current node.

From Figure below, we can calculate the h-score by comparing the initial(current) state and goal state and counting the number of misplaced tiles.

Thus, h-score = 5 and g-score = 0 as the number of nodes traversed from the start node to the current node is 0



How A* solves the 8-Puzzle problem.

We first move the empty space in all the possible directions in the start state and calculate the f-score for each state.

This is called expanding the current state.

After expanding the current state, it is pushed into the closed list and the newly generated states are pushed into the open list. A state with the least f-score is selected and expanded again.

This process continues until the goal state occurs as the current state. Basically, here we are providing the algorithm a measure to choose its actions. The algorithm chooses the best possible action and proceeds in that path.

This solves the issue of generating redundant child states, as the algorithm will expand the node with the least f-score.

Application:

A* Algorithm is used in various diversified areas of research, computer network, game search problem & graph, Google maps to find the shortest path between source and destination.

Conclusion:

In this way we implemented A* Algorithm for 8-Puzzle problem.