AI Lab 05
Section A

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. The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal configuration.

Initial State

1	2	3
8		4
7	6	5

Goal State

2	8	1
	4	3
7	6	5

The tiles in the initial(start) state can be moved in the empty space in a particular order and thus achieve the 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 or

4. Left

The empty space cannot move diagonally and can take **only one step at a time** (i.e. move the empty space one position at a time).

Before we talk about the A* algorithm, we need to discuss Heuristic Search.

Basically, there are two types of searching techniques:

1. **Uninformed Search** and

2. **Informed Search**

You might have heard about Linear Search, Binary Search, Depth-First Search, or the Breadth-First Search. These searching algorithms fall into the category of uninformed search techniques i.e. these algorithms do not know anything about what they are searching for and where they should search for it. That's why the name "uninformed" search. Uninformed searching takes a lot of time to search as it doesn't know where to head and where the best chances of finding the element are.

Informed search is exactly opposite to the uninformed search. In this, the algorithm is aware of where the best chances of finding the element are and the algorithm heads that way! **Heuristic** search is an informed search technique. A heuristic value tells the algorithm which path will provide the solution as early as possible. The heuristic function is used to generate this heuristic value. Different heuristic functions can be designed depending on the searching problem. So we can conclude that **Heuristic search is a technique that uses a heuristic value for optimizing the search.**

A* Algorithm

A* is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently traversable path between multiple points, called nodes. Noted for its performance and accuracy, it enjoys widespread use.

The key feature of the A* algorithm is that it keeps a track of each visited node which helps in ignoring the nodes that are already visited, saving a huge amount of time. It also has a list that holds all the nodes that are left to be explored and it chooses the most optimal node from this list, thus saving time not exploring unnecessary or less optimal nodes.

So we use two lists namely 'open list' and 'closed list' the open list contains all the nodes that are being generated and are not existing in the closed list and each node explored after it's neighboring nodes are discovered is put in the closed list and the neighbors are put in the open list this is how the nodes expand. Each node has a pointer to its parent so that at any given point it can retrace the path to the parent. Initially, the open list holds the start(Initial) node. The next

node chosen from the open list is based on its **f score**, the node with the least f score is picked up and explored.

f-score = h-score + g-score

A* uses a combination of heuristic value (h-score: how far the goal node is) as well as the g-score (i.e. the number of nodes traversed from the start node to current node).

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.

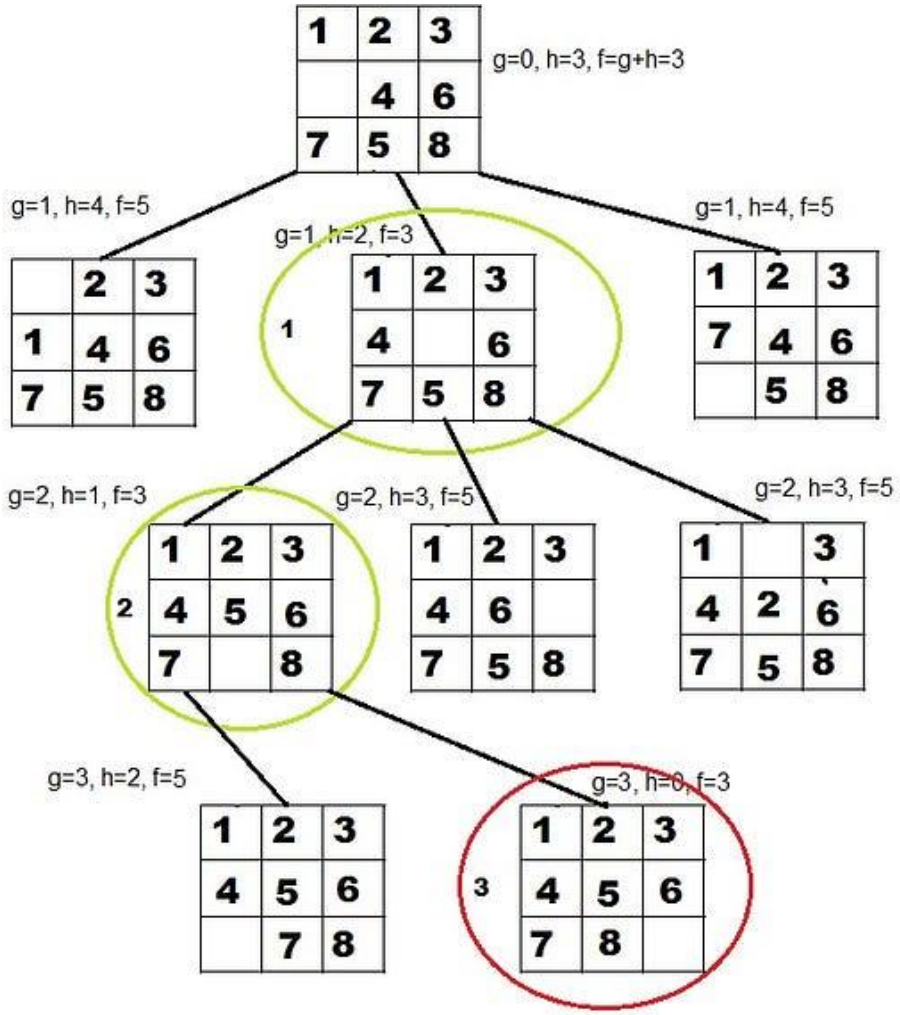
g-score will remain as the number of nodes traversed from a start node to get to the current node.

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**.



Your task is to implement A* search for this problem. You have to implement two classes in the code: Node & Puzzle.

Node class defines the structure of the state(configuration) and also provides functions to move the empty space and generate child states from the current state. Puzzle class accepts the initial and goal states of the N-Puzzle problem and provides functions to calculate the **f-score** of any given node(state).