Lab Manual 3

Aim: 8 Puzzle Problem with A* search algorithm

A* search algorithm:

A* search algorithm is the best first search algorithm that visits next state based on heristics *f(n) = h + g* where *h* component is same heuristics applied as in Best-first search but *g* component is path from the initial state to the particular state. Therefore it doesn't chooses next state only with lowest heuristics value but one that gives lowest value when considering it's heuristics and cost of getting to that state.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 from this list it chooses the most optimal node 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 it's 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 it's f score, the node with the least f score is picked up and explored.

What is f score?

f score is nothing but the sum of the cost to reach that node and the heuristic value of that node.

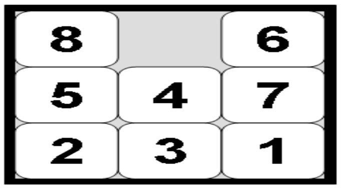
For any give node the f score is defined as:

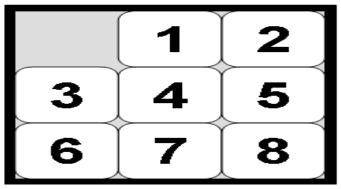
$f(x)=h(x)+g(x)$

where $g(x)$ is the cost of that node, $h(x)$ is the calculated heuristic of that node and x is the current node.

8 Puzzle Problem:

The 8 puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Such a puzzle is illustrated in following diagram.





Start state **Goal State**

P. Design a program to change the initial configuration into the goal configuration, by moving tiles to their goal positions using A* heuristic technique.

So first of all You need to decide what all information is needed to be kept in the node.

- **You need to save the puzzle state at that instance, thus a 3X3grid.**
- **You need to store the heuristic value h.**
- **The pointer that points to the parent node.**

Structure of a Node

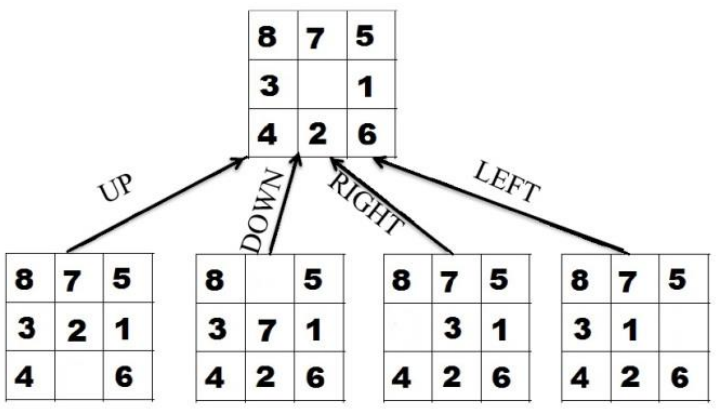
```
Structure Node
{

variable

array grid[N][N] ,

variable h;
pointer parent;}
```

Making a graph

You need to realize the search space as a graph. Each state in the graph is represented with it's puzzle configuration, thus each node is a separate puzzle state which is produced by sliding a tile to the blank space on the previous state. Let's take the following case:

8	7	5
3		1
4	2	6

The above figure is considered as a single node in a graph, let's take it as the starting node. The following will be the expanded graph in the next epoch when the available moves are implemented.



Each node can have maximum of 4 children, the graph fill further expand in a similar fashion with each child pointing to the parent using pointer.

The g score here, which is the cost of the move will increase by 1 at each depth since each tile sliding to the blank space represents one move and in the end You need to get the optimal moves for the puzzle instance, this basically mean that You have to add 1 to the g score of the parent before storing it in the child nodes.

- **You can use Manhattan Distance as heuristic function here.**

Manhattan Distance of a tile is the distance or the number of slides/tiles away it is from it's goal state. Thus, for a certain state the Manhattan distance will be the sum of the Manhattan distances of all the tiles except the blank tile.

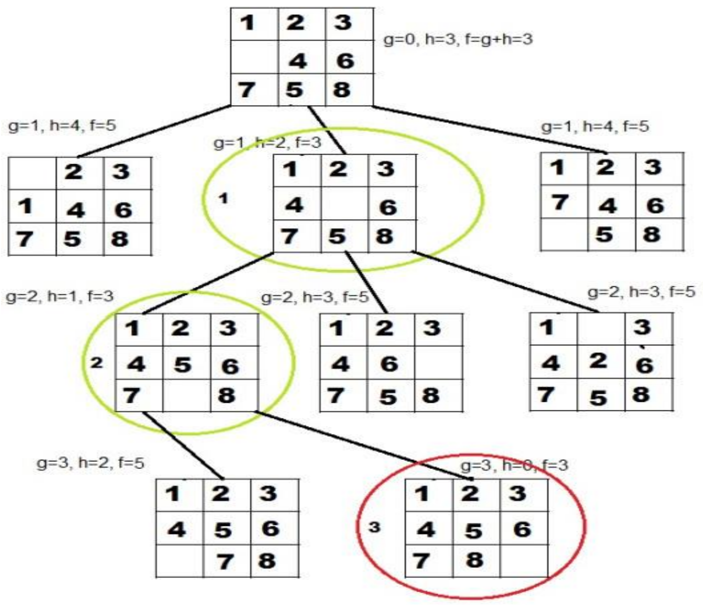
Let's take the following example.

1	2	5
3		6
7	4	8

Here we see the tiles 5, 3, 4, 8 are misplaced so we calculate the Manhattan distances of each of these tiles which are 2, 3, 2, 1 respectively, thus the Manhattan heuristic value for this state will be 8 (h=8). Manhattan Distance is faster than Hamming distance explained above but it's still slow for higher values of N.

<u>Progression</u>

- **Now let's use Manhattan heuristic on a random 8-Puzzle instance and work it towards the solution.**



- **As we know A* selects the node with least f score each cycle after expanding the selected node , we see above clearly how the solution is found in 3 cycle of expansion after choosing the least f score each step.**