Programming Assignment: Searching

ITCS 6150/8150

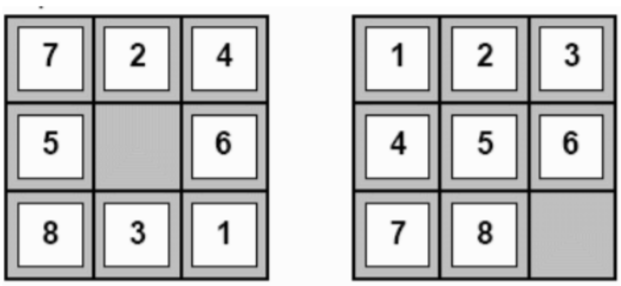
Student Name: Rucha Pramod Visal
Student ID: 801257824

Intelligent Systems Project 1

Solving the sliding puzzle problem using A* search algorithm

Problem Formulation

The problem contains 9 tiles in a grid of 3*3 with numbered tiles 1-8 with one empty space. Tiles can slide over the grid. The goal is to reach the Goal configuration supplied. For example, the figure shows a simple initial and corresponding goal state to achieve by sliding the tiles:



Start State Goal State

The blank tile helps us to reach the goal. The problem formulation for the above puzzle is as follows:

1.) States: The description consists of the location of the 8 tiles on the grid as well as blank tiles.

2.) Initial State: Any state provided by the user can be the initial state of the puzzle. However, this algorithm does not guarantee to generate the goal state from all the possible initial states.

3.) Actions: Movements depend on the blank tile's position. For e.g. If the Centre tile is blank, All UP, DOWN, LEFT, RIGHT actions are possible but if its Right Corner tile is blank, then only RIGHT and DOWN actions are possible.

4.) Transition Model: For every action, a new state will be generated.

5.) Goal Test: This checks if the current state matches the Goal state provided for the problem. This will lead the algorithm to stop if it has achieved the goal test.

6.) Path Cost: The step cost will be 1 as we will be moving 1 tile at a time to get to the goal configuration.

A* Algorithm to solve the N-puzzle Problem

A* Search Algorithm is an optimal algorithm to find a solution to the 8-puzzle problem. The basic operation of a* says that any node with the lowest value for f(n) is expanded first.

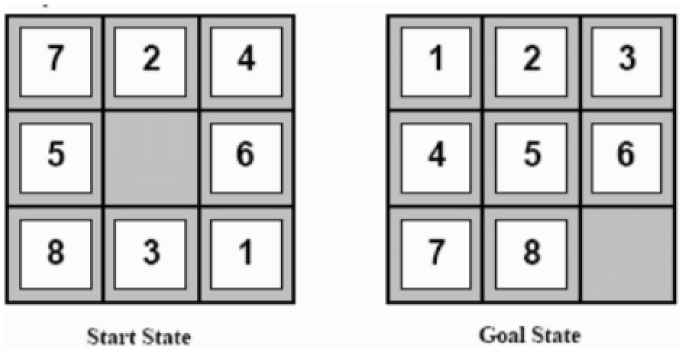
Here, f(n) is an evaluation function which gives a cost associated for each node. This cost decides the next mode to be chosen for expansion. We calculate this cost by following formula:

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

Where, g(n) is the actual cost (cost which has been accumulated until now) of the node n. h(n) is the estimated cost of n to the goal state. H(n) is an estimated value given by the heuristic function. Every problem needs domain knowledge and based on that we need a heuristic function to calculate h(n) according to its configuration and formulation.

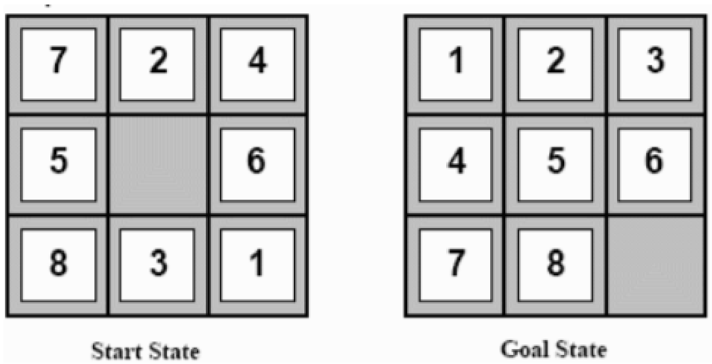
In A* algorithm, for an 8-puzzle problem, we have step cost = 1. We have two heuristics methods to calculate h(n) for every node. Those are:

Misplaced Tiles: This function counts the number of tiles of the current state which are out of their place compared to the goal state. For e.g., in below example,



we see that tile number 7,4,5,8,3,1 is wrongly placed compared to the goal state. Hence the heuristic value for h(n) = 1+1+1+1+1+1 = 6 for the start state. h(n) is calculated for every node that is generated. This node is then put in the priority queue which selects the node with lowest cost f(n) for expansion. The next node will be again with lowest h(n). Priority Queue sorts the nodes in the non-decreasing order of misplaced tiles value h(n). The lowest value would thus occur at the first position in the queue and the corresponding node will be expanded next.

Manhattan Distance: In this heuristic, we calculate cost of node n by number of steps required by each misplaced tile to go from its actual location in the current state to the required location in the goal state. It is the sum of the number of steps of each misplaced tile to go from current to goal state. for e.g.,



In this example, to calculate the h(n) for the start state: h(n)= h(1) + h(3) + h(4) + h(5) + h(7) + h(8) as these are misplaced => 4 + 3 + 3 + 1 + 2 + 1 => 14 Thus, the Manhattan distance for the current state is 14. Similar to misplaced tiles, we store the Manhattan distances along with their states for each iteration. We choose the node with the lowest value to be the next node for expansion. Priority Queue used will be to sort the possible nodes to the increasing order of their Manhattan distances so that initial will be the minimum. The first element of the queue is expanded in each iteration.

This process of node generation and expansion continues until the initial state provided by the user is not equal to the goal state. Every time a node is selected for expansion, it will be from the priority queue. Whenever a node is generated, it is added to the priority queue. The queue sorts according to its value. This process of generating the nodes and adding it to the queue may generate a loop, if a child generates a state exactly like its parent or any ancestors. To handle this problem, we store the explored nodes and check if any new nodes match anyone from the explored. If a match is found, the new node is discarded, otherwise it will be added to the queue for expansion in the next iteration. For some special cases, however, A* cannot find a solution to 8 puzzle problems. The algorithm then needs to terminate if it is unable to reach the goal state at all after a specified number of iterations.

<u>Program Structure</u>

I have implemented the A* search algorithm to solve the 8-puzzle problem using **Python**.

Global Variables:

Variable Name	Description
numberNodesExpanded	Number of nodes expanded while solving the problem
puzzle	Initial State
solution	Final State

Functions and Procedures:

Function Name	Description
generateSuccessors	This function generates possible successors depending upon the current state of the node. For e.g. If the blank tile i.e., 0 value is in the right corner, it will only generate 2 children
findBlankSpace	This function is used to find out the position of 0 nodes in the grid.
generateChildNode	This function will generate the actual state with g(x) and configuration. This function will be called from generateChildren.
getInput	This function is used to get the user input.
printArr	This function is used to print the Array like 8 puzzle views on the console.
checkSolvability	This function checks if the problem is solvable based on initial and current state
calculateHval in HeuristicMisplacedTiles Class	This function will calculate the h(x) for the current state using misplaced tiles heuristic
calculateHval in HeuristicManhattan Class	This function will calculate the h(x) for the current state using the Manhattan heuristic
solve	This function is the main function. This will take the input state, goal state and the heuristic. It will start solving the puzzle based on that

<u>Input/Output:</u>

<u>Case 1:</u> 0,1,3,4,2,5,7,8,6

No of nodes Expanded: 5
Path Cost: 4
Final Path from Initial State to Goal State:

```
['0', '1', '3']
['4', '2', '5']
['7', '8', '6']

   ||
   ||
   \/

['1', '0', '3']
['4', '2', '5']
['7', '8', '6']
```

||
||
\/

['1', '2', '3']
['4', '0', '5']
['7', '8', '6']

||
||
\/

['1', '2', '3']
['4', '5', '0']
['7', '8', '6']

||
||
\/

['1', '2', '3']
['4', '5', '6']
['7', '8', '0']

<u>Case 2:</u> 1,0,3,4,2,5,7,8,6

No of nodes Expanded: 4
Path Cost: 3
Final Path from Initial State to Goal State:

['1', '0', '3']
['4', '2', '5']
['7', '8', '6']

||
||
\/

['1', '2', '3']
['4', '0', '5']
['7', '8', '6']

||
||
\/

['1', '2', '3']

['4', '5', '0']
['7', '8', '6']

||
||
\/

['1', '2', '3']
['4', '5', '6']
['7', '8', '0']

Case 3: 1,2,3,4,0,5,7,8,6

No of nodes Expanded: 3
Path Cost: 2
Final Path from Initial State to Goal State:

['1', '2', '3']
['4', '0', '5']
['7', '8', '6']

||
||
\/

['1', '2', '3']
['4', '5', '0']
['7', '8', '6']

||
||
\/

['1', '2', '3']
['4', '5', '6']
['7', '8', '0']

Case 4: 1,2,3,4,5,0,7,8,6

No of nodes Expanded: 2
Path Cost: 1
Final Path from Initial State to Goal State:

['1', '2', '3']
['4', '5', '0']
['7', '8', '6']

||
||
\/

['1', '2', '3']
['4', '5', '6']
['7', '8', '0']

Case 5: 1,2,3,4,5,6,7,8,0

No of nodes Expanded: 1
Path Cost: 0
Final Path from Initial State to Goal State:

['1', '2', '3']
['4', '5', '6']
['7', '8', '0']

Case 6: 3,4,0,5,6,2,7,1,8

No of nodes Expanded: 548
Path Cost: 20
Final Path from Initial State to Goal State:

['3', '4', '0']
['5', '6', '2']
['7', '1', '8']

||
||
\/

['3', '4', '2']
['5', '6', '0']
['7', '1', '8']

||
||
\/

['3', '4', '2']
['5', '0', '6']
['7', '1', '8']

||

||
\/

['3', '4', '2']
['5', '1', '6']
['7', '0', '8']

||
||
\/

['3', '4', '2']
['5', '1', '6']
['0', '7', '8']

||
||
\/

['3', '4', '2']
['0', '1', '6']
['5', '7', '8']

||
||
\/

['0', '4', '2']
['3', '1', '6']
['5', '7', '8']

||
||
\/

['4', '0', '2']
['3', '1', '6']
['5', '7', '8']

||
||
\/

['4', '1', '2']
['3', '0', '6']
['5', '7', '8']

```
   ||
   ||
   \/

['4', '1', '2']
['0', '3', '6']
['5', '7', '8']

   ||
   ||
   \/

['4', '1', '2']
['5', '3', '6']
['0', '7', '8']

   ||
   ||
   \/

['4', '1', '2']
['5', '3', '6']
['7', '0', '8']

   ||
   ||
   \/

['4', '1', '2']
['5', '3', '6']
['7', '8', '0']

   ||
   ||
   \/

['4', '1', '2']
['5', '3', '0']
['7', '8', '6']

   ||
   ||
   \/

['4', '1', '2']
['5', '0', '3']
['7', '8', '6']
```

```
      ||
      ||
      \/

['4', '1', '2']
['0', '5', '3']
['7', '8', '6']

      ||
      ||
      \/

['0', '1', '2']
['4', '5', '3']
['7', '8', '6']

      ||
      ||
      \/

['1', '0', '2']
['4', '5', '3']
['7', '8', '6']

      ||
      ||
      \/

['1', '2', '0']
['4', '5', '3']
['7', '8', '6']

      ||
      ||
      \/

['1', '2', '3']
['4', '5', '0']
['7', '8', '6']

      ||
      ||
      \/

['1', '2', '3']
['4', '5', '6']
```

['7', '8', '0']

Case 7: 1,5,6,3,7,4,0,2,8

Time limit exceeded.

Case 8: 8,3,0,5,6,1,7,4,2

Time limit exceeded.

Case 9: 1,2,3,4,5,6,8,7,0

The problem is not solvable.