PROJECT 1

Solving the 8-puzzle using A* algorithm

Anusha Vudathu

Harshini Koduru

Table of contents

1. **Problem**

   Solve 8-puzzle problem using A* algorithm

2. **Aim**

   To derive the goal state of the puzzle from the given initial state using A* search algorithm which uses following heuristic functions.

   a. Misplaced tiles
   b. Manhattan Distance

3. **Language**

   The code is developed in python 3.7

4. **Abstract**

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals. In each iteration of algorithm's main loop, it determines which of its paths to extend. A* does that based on the estimate of the cost required to extend the path all the way to the goal and cost of the path. A* selects the path that minimizes the cost i.e., at each step it chooses the node that has low f(n) value and processes that node. A* solves any problem by calculating below value of f(n)

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

where n is the next node on the path, g(n) is the cost of the path traversed so far i.e., from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. This project entails solving the 8-puzzle problem using A* algorithm. Two heuristic functions used here are misplaced tiles and manhattan distance. This project will focus on reaching the goal state from initial state of 8-puzzle problem suing A* algorithm.

5. **Introduction**

A* algorithm is the most popular *informed search algorithm* because it is flexible, and it is often used to find the shortest distance from one point to another point. However, *Greedy Best-First-Search algorithm* works in a similar way as A* algorithm, except that it has some estimate (called a heuristic) of how far from the goal any vertex is but does not consider path travelled so far. Instead of selecting the vertex closest to the starting point, it selects the vertex closest to the goal. It is not guaranteed to find the shortest path.

In this project we are using any one of the heuristics to solve the 8-puzzle problem. Depending on the following heuristics values the algorithm chooses which node to expand next.

- **Misplaced Tile Heuristic**

  In this heuristic function, each element at given index in the present node is compared with the corresponding element in the goal state to find the number of mismatched tiles.

- **Manhattan Distance Heuristic**

  In this heuristic function, position of each element is compared with the goal state to find the number of moves required to place the misplaced tile in correct position. If the position of the tile matched with goal state position that means no moves are required and the value is 0. The cumulative sum of the required moves for each misplaced tile is the return value of the Manhattan Distance heuristic function.

6. Experimental Analysis

- **Input**

The program accepts Initial and goal state from the user. User provides the elements of the puzzle to be solved. The program is designed in such a way that it takes user input to choose any one of the two heuristics to solve the problem.

- **Output**

The application prints the expanded nodes along with the initial and goal nodes. It also provides total number of nodes generated, expanded and unexpanded. User will also be notified depth of the goal node. It even displays the total time took for the algorithm to find the goal state.

6.1. Global Variables

(a) *N_Board*	: Size of the puzzle. Eg: 8
(b) MATRIX_SIZE	*:* Size of matrix, int(math.sqrt(N_Board+1)). Eg: 3 if N_Board is 8
(c) Number_of_nodes	*:* keep track of total number of nodes generated.

6.2. Classes defined in the program

6.2.1. *Priority Queue*

- This class contains all the methods which will initialize, updates the priority queue and prioritize and pops the child nodes.

Methods

Def_init_	Initializes the priority queue object
Insert_element	Appends the elements in the priority queue. It has attributes h, g, priority. It also sorts the queue based on the priority property of the node.
Get_child	It pops out the most desired child node i.e., it pops the first element of the queue
Get_rem_elements	This method calculates the number of elements in the priority queue
Empty	It checks whether the length of the queue is zero or not

6.2.2. *Board*

- This class contains methods that solves the puzzle using heuristics functions. It contains methods to select which heuristic function to be used, methods to expand the child nodes by moving the '0' tile in possible directions, methods to print current state of the puzzle.
- It imports the other classes (Priority_queue and Moves)

Methods

def_init	Initializes the Board class object and gives the initial state to it.
get_initial_state	It returns the initial state.
print_initial_state	It prints the initial state
test_if_goal	This method checks whether the popped node is same as the goal state or not
get_explored_length	This method returns number of explored states
is_explored	It checks whether the provided node is already explored or not
get_goal_state	It returns the goal state
misplaced_tile	It will take all the possible generated child nodes and then in-turn call the method that calculates the number of misplaced tiles and puts those child nodes into the fringe (priority queue).
Manhattan_distance	It will take all the possible generated child nodes and then in-turn call the method that calculates the total manhattan distance and puts those child nodes into the fringe (priority queue)
H_misplaced_tiles	This method calculates the number of the misplaced tiles as per the logic.
H_manhattan_distance	This method calculates the total manhattan distance for each child node using row difference and column difference
Select_strategy	Based on the heuristic selected by the user it will take the initial state and the newly generated nodes. This method expands the nodes by popping out the node from the fringe (priority queue) and calling *expansion* method. Even it checks every node whether it goal state or not. If it's a goal state it will print the corresponding nodes details like nodes *generated, expanded, unexpanded, the depth of goal node.*
Expansion	This method calls the methods from Moves class to check the possible moves that can be done by moving the '0' tile to generate the child nodes. It also keeps track of the number of the nodes generated.
Print_state	It prints the current state in the puzzle solving process
Main	It has the main display logic to solve the 8-puzzle program. It contains the code to take the input from the user and to generate and display the output.

6.2.3. Moves

- This class contains the methods to move the ‘0’ or blank tile to generate the child nodes.
- It defines the methods to check whether the tile can be moved in the given direction or not and if it can be moved then it calls the methods with logic to move it.

Methods

Check_move_down	It checks whether the ‘0’ tile can be moved down or not within the matrix boundaries from the present index
Check_move_left	It checks whether the ‘0’ tile can be moved left or not within the matrix boundaries from the present index
Check_move_right	It checks whether the ‘0’ tile can be moved right or not within the matrix boundaries from the present index
Blank_tile_up	It calls the check_move_up method, if it is true then it will move the blank tile up by 1 step
Blank_tile_down	It calls the check_move_down method, if it is true then it will move the blank tile down by 1 step
Blank_tile_left	It calls the check_move_left method, if it is true then it will move the blank tile left by 1 step
Blank_tile_right	It calls the check_move_right method, if it is true then it will move the blank tile right by 1 step

7. Algorithm Strategy and Formulation

- In this project, we accept the input from the user i.e., initial state and goal state of the 8-puzzle problem.
- These inputs are given as lists.
- The user is asked to select the strategy to implement a*. Here we implement misplaced tile heuristic and manhattan distance heuristic.
- Then the Select_strategy() method is called where we pass the initial state and heuristic selected as arguments.
- In the Select_strategy,the following methods are called.
  - a.) misplaced_tile() / manhattan_distance() methods calculate the h based on the heuristic definition and then insert all the corresponding values in the elements[] list. The elements in the list are sorted based on priority in ascending order and then the element with least f(n) is popped out and is checked for the goal state. If it is not goal state, then all the possible children are generated using expansion method. The process continues until the goal state is found.
  - b.) Expansion () method is used to generate all the possible child nodes.
- To generate the child nodes, the blank tile is moved in all the possible directions and for each generated child node heuristic value h(n) and path so far traversed are calculated and the node which has least f(n) is selected to generate next set of nodes.

- Each element removed from the priority queue is checked for the goal state. If the goal state is achieved, following are printed:
  - o Number of nodes expanded

- Number of nodes left in fringe when goal is found.
- The depth of the goal node
- Total number of nodes generated
- Goal Path

8. Results

Sample	Input	Misplaced tiles Function	Manhattan Distance function
Sample 1	**Initial state:** **1 | 2 | 3** **7 | 4 | 5** **6 | 8 | 0** **Goal State:** **1 | 2 | 3** **8 | 6 | 4** **7 | 5 | 0**	The goal path is initial->u->l->l->u->l->d->l->u->r->u->u->l->r->l->d->r->u->d->r->d->r Total number of nodes expanded: 22 Number of unexpanded nodes:20 Depth of the goal node:8 The algorithm has generated 41 nodes in total including goal state.	The goal path is initial->u->l->d->r->l->u->r->d->r Total number of nodes expanded:10 Number of unexpanded nodes:9 Depth of the goal node:8 The algorithm has generated 18 nodes in total including goal state.
Sample 2	**Initial state:** **2 | 8 | 1** **3 | 4 | 6** **7 | 5 | 0** **Goal State:** **3 | 2 | 1** **8 | 0 | 4** **7 | 5 | 6**	The goal path is initial->u->l->u->l->l->d->r Total number of nodes expanded: 8 Number of unexpanded nodes:7 Depth of the goal node:6 The algorithm has generated 14 nodes in total including goal state	The goal path is initial->u->l->u->l->d->r Total number of nodes expanded: 7 Number of unexpanded nodes:6 Depth of the goal node:6 The algorithm has generated 12 nodes in total including goal state
Sample 3	**Initial state:** **7 | 8 | 0** **1 | 2 | 3** **4 | 5 | 6** **Goal State:** **0 | 8 | 1** **7 | 2 | 3**	The goal path is initial->d->l->l->d->l->d->d->l->l->u->u->l->u->d->d->r->d->r->r->r->d->d->r->l->r->u->u->d->r->u->l->l->r->l->r->d->r->d->r->d->d->r->d->d->l->l->r->u->l->l Total number of nodes expanded: 51	The goal path is initial->d->l->d->l->l->d->l->l->u->u->d->l->u->d->d->r->d->d->r->r->r->d->r->d->r->u->l->l Total number of nodes expanded: 29

	4 | 5 | 6	Number of unexpanded nodes:35 Depth of the goal node:10 The algorithm has generated 85 nodes in total including goal state	Number of unexpanded nodes:23 Depth of the goal node:10 The algorithm has generated 51 nodes in total including goal state
Sample 4	**Initial state:** **4 | 5 | 6** **7 | 8 | 0** **1 | 2 | 3** **Goal State:** **4 | 5 | 6** **7 | 8 | 1** **0 | 2 | 3**	The goal path is initial->u->d->l->l->l->u->d->l->l->l->d->d->l->u->l->r->r->u->u->u->r->r->r->d->u->d->l->d->u->l->d->r->r->u->u->d->l->l->r->r->u->r->u->r->d->u->d->u->l->d->d->d->d->r->d->d->l->l->l->u->d->r->r->r->u->r->l->r->u->r->u->u->d->l->u->l->u->d->u->d->d->r->u->r->u->l->l->d->l->l->r->r->u->u->l->l->d Total number of nodes expanded: 98 Number of unexpanded nodes:64 Depth of the goal node:11 The algorithm has generated 161 nodes in total including goal state	The goal path is initial->u->d->l->l->l->u->d->l->l->l->d->d->u->l->l->r->r->u->u->u->r->d->r->d->u->l->l->r->d->d->u->u->u->l->r->r->r->r->d->r->u->l->l->d Total number of nodes expanded: 45 Number of unexpanded nodes:37 Depth of the goal node:11 The algorithm has generated 81 nodes in total including goal state