Project 1

Solving the 8-puzzle using A* algorithm

-KINTALI CHAITANYA
-VAMSHI GOUD VALDAS

Problem:

To solve an 8-puzzle problem by using A* algorithm.

Algorithm:

In this project we are using a A* algorithm with two heuristics functions (Manhattan, Misplaced tiles) in order to solve the 8-puzzle problem.

Aim:

To find the solution path from a given initial state to a goal state by implementing A* algorithm.

Heuristics used:

The program provides a facility to the user where he can use any of the above-mentioned heuristics (misplaced tiles, Manhattan distance) to choose and expand the generated nodes.

1) Misplaced tiles
2) Manhattan Distance

Algorithm Functioning:

A* search algorithm uses an evaluation function f(n) to decide which node needs to be expanded further to reach the goal state (here f(n) is sum of a heuristic function h(n) and g(n) which specifies the cost to reach the goal). We mainly use two heuristics in this problem.

1.Misplaced Tiles Heuristic

In this heuristic function, each element at a certain index position in the present node is compared with the goal state to find the number of misplaced tiles. The number of misplaced tiles for that state is the heuristic value by using misplaced tiles heuristic function.

Based on this value, the algorithm chooses the best child node (from all the child nodes generated) which is expanded further to reach the goal state.

2.Manhattan Distance

In this heuristic function each element in the present node is compared with goal state to calculate the number of steps required for the misplaced tile to be placed in its exact position.

The return value for this heuristic function is the cumulative sum of the steps required each misplaced tile to reach its exact position.

Formula: | (difference of x coordinates of both initial and goal state) | + |(difference of y coordinates of both initial and goal state)|

Language:

The code is developed using Python 3.7.

Editor:

Jupiter Spyder

Input:

The user has the facility to give the initial state and goal state and also has the option of choosing any of the two heuristic functions as per the user's convenience.

Output:

The program prints the initial state, goal state along with the expanded nodes. It also provides the solution details such as mentioned below.

- *Path to Goal*
  Example: ['Right', 'Left', 'Down']
- *Cost of solution path*
  Example: 4 (Assuming each tile move costs 1)
- *Number of Nodes Explored*
  Example: 4
- *Number of elements left in frontier*
  Example: 7
- *Search depth*
  Example: 1

-KINTALI CHAITANYA
-VAMSHI GOUD VALDAS

- *Puzzle output sequence*

  Example:

```
1   2   3

4   5   6

7   0   8

      ||
      ||
      ||
      \/


1   2   3

4   5   6

7   8   0

 **************** GOAL STATE FOUND!!!!!! *************
```

Global Variables:

- *Puzzle_len: Size of the puzzle. E.g. 8*
- *Puzzle_side: Length of the puzzle. E.g. 3*
- *Goal_node: Goal Node State is stored in this variable*
- *Nodes_expanded: Nodes expanded during the execution of the algorithm*

Classes:

1) ***State –*** Contains the State Node properties and methods such as comparators to compare two States at any point of time.

   Properties:

Property	***Data Type***
State	*ArrayList<Integers>*
Parent	*State*
Move	*Integer*
Depth	*Integer*
Cost	*Integer*
Cumulative Cost	*Integer*

Methods:

1. **__eq__:**
   This method is used to compare the sequence strings of two states (instances of the class) and return the Boolean value.
2. **__lt__:**
   This method is used to compare two states with their cumulative costs and return the Boolean value.

2) **Utilities -** Contains methods to print the puzzle when given an input list sequence.
   Methods:
   1. **printThePuzzle:**
      This method accepts an input list of sequence of a state, and a Boolean flag to check whether to print the sequence or just the puzzle.
3) **SearchExecution –** This class contains methods, properties for initializing the puzzle, storing the initial state, goal state, calculating heuristic values, frontiers, heap, sequence dictionary and move functions.

Properties	**DataType**
Initial_state	ArrayList<Integers>
Goal_node	State
Frontier_Heap	List<State>
Explored	List<State>
Heap_Dictionary	{“string”: State}
Puzzle_len	Integer
Puzzle_side	Integer
Moves	List

***Method*s:**

a. PrintInitialMessages(): Prints the initial Welcome messages in the Console.
b. manhattanHeuristic(node_list): This method calculates Manhattan heuristic value for an input sequence.
c. misplaced Tiles (node list) – This method calculates the misplaced tiles count and returns the sum as heuristic function value.
d. backtrace(): This method backtracks the sequence from goal state to initial_state and prints the sequence to reach from initial_state to goal_state.
e. printTheSolution(). This method prints the goal sequence from initial state to goal state.

f. SearchAStar(): This method takes input, goal state sequences, calculates the Heuristic and cost path values, generates, expands and pushes and pops from the priority queue by maintaining the order of the queue.
g. readtheBoard(): This method reads the input state from the user.
h. Expand(node): This method expands the node and returns the list of neighbours with all the possible moves in the puzzle.
i. Move(): This method moves all the tiles to the adjacent possible moves by checking the boundary of the puzzle and returns their states.
j. Main(): This method calls the printinitialMessage(), readTheBoard(), SearchAStar(), printTheSolution() methods.
k. Function_map: This map stores the mapping between the user input integer and heuristic function names.

Formulation:

1) We take the input sequence and the goal sequence from the user and will store them in the global variables.
2) The input sequence and goal sequence are stored in the array list of integers.
3) The programme also takes input from the user to store which heuristic function is used to evaluate the heuristic value of a state.
4) The SearchAStar method inserts nodes into the frontier_heap (priority queue).
5) The heap is iterated till it is empty and the element with least cost is removed the heap and examined and explored.
6) The variable nodes expanded is auto incremented each time the element is removed from the queue.
7) The recently popped node is goal tested and return the node if the goal node is found.
8) If the goal node is not found, expand the current node and generate the neighbours and assigns the cost, state, move, depth, cumulative cost values to the nodes.
9) Each node inserted into the Priority Queue is sorted based on the f(n) value which is calculated as
   $F(n) = g(n) + h(n)$
   Where $g(n)$ = cost for reaching the node
   $H(n)$ = heuristic function output
10) Each element removed from the priority queue is checked for the goal state. If the goal state is found, following are printed.
    a. Path to Goal
    b. Cost_to_path
    c. Nodes_explored
    d. Number of elements left in frontier
    e. Search depth.

-KINTALI CHAITANYA
-VAMSHI GOUD VALDAS

Sample Input / Outputs

Initial State	Goal state	Path to goal	Cost of path	Nodes explored	Nodes Generated	Number of states left in frontier	Search depth	Heuristic Function
3 2 1 4 5 6 8 7 0	1 2 3 4 5 6 7 8 0	U, L, U, L, D, D, R, R, U, L, U, R, D, L, L, U, R, R, D, D, L, U, R, D	24	4230	6605	2375	24	Manhattan
1 2 3 4 6 5 8 7 0	1 2 3 4 5 6 7 8 0	L, U, R, D, L, L, U, R, R, D, L, U, L, D, R, R	16	239	387	148	16	Manhattan
1 2 3 7 4 5 6 8 0	1 2 3 8 6 4 7 5 0	U, L, D, L, U, R, D, R	8	22	42	20	8	Misplaced Tiles
1 2 3 5 4 6 7 8 0	1 2 3 4 6 5 7 8 0	U, L, L, D, R, U, R,D,L , L,U, R, R,D	14	270	444	174	14	Misplaced Tiles

-KINTALI CHAITANYA
-VAMSHI GOUD VALDAS

INFERENCE:

By running the program for different initial and goal states we can observe that the nodes generated, explored by misplaced tiles heuristic function is higher than that of Manhattan distance heuristic. Therefore, in order to save memory and quicker results Manhattan distance is the preferred heuristic function for 8 puzzle problem.

Example:

INITIAL STATE

1	2	3
8	6	4
7	5	0

GOAL STATE

1	2	3
8	6	4
7	5	0

In the above case the nodes generated using misplaced tiles are 42 whereas when Manhattan distance heuristic is implemented the number of nodes generated are 21.

-KINTALI CHAITANYA
-VAMSHI GOUD VALDAS