Problem Formulation

A* searchalgorithm is mainly used for path finding and path traversal in solving problems by searching. It plays an important role in **Informed Search Strategies**.It solves the limitations of the greedy algorithm by considering the best heuristic function as well as considering the uniform cost value. The main aim here is to avoid expanding paths that are already **expensive**. This search algorithm is thus **Optimal**.

8 Puzzle Problem:A 3×3 board with 8 tiles is given where every tile has one number from 1 to 8 and one empty space with number 0 in our case. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

Evaluation function:f(n) = g(n) + h(n)
g(n) = cost so far to reach node
h(n) = estimated cost to goal from node
f(n) = estimated total cost of path through node to goal

A∗ search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$
where $h^*(n)$ is the true cost from n.

A∗ search expands lowest g + h

Program Structure

- The problem consists of various problem elements like:
- **Initial State:**It is the first basic state where the agent is in and from where it starts solving the problem. Here the user enters the initial state.
- **Goal State:**It is a sequence of steps or a procedure to identify if the goal or desired state is reached. Here the user enters the goal state.
- **Possible Actions:**Given a particular state, it returns the sequence of actions that can be executed. Here, the actions of moving the empty tile in any of the 4 possible directions i.e up, down, left and right is possible.
- **Formulation:**A successor formulation of explored states is used which tracks the reachable state from any given state. Successor here decided on the basis of two heuristic functions and cost function.
- **State:**It is the integer location of tiles.
- **State Space:**It is all the possible reachable states from the start state.
- **Path:**It is a sequence of actions that is followed while traversal.
- **Path Cost:**It is a cost function that shows the amount of cost to reach a particular state.
- **Step Cost:**It is cost of single step action taken. Here it is cost one per move.
- **Solution:**It is a sequence of actions that we take from our initial state to the goal state.
- **Strategy:**It is the order of node expansion. Here we decide that on the basis of minimum value of nodes that we get from calculating the heuristic function and uniform cost function.

  Strategies are evaluated along the following dimensions:
- **Completeness** — It always find a solution unless there are infinitely many nodes with $f \leq f(G)$
- **Time complexity**—number of nodes generated/expanded i.e Exponential in [relative error in h × length of soln.]
- **Space complexity**—maximum number of nodes in memory
- **Optimality**— it always find a least-cost solution

  A∗ expands all nodes with $f(n) < C*$

  A∗ expands some nodes with $f(n) = C*$

  A∗ expands no nodes with $f(n) > C*$

Global Variables

1. **N =** Size of matrix
2. **Fringe List =** List of nodes which are yet to explored
3. **Explored List =** List of nodes which are explored
4. **Value =** Current state of the node
5. **Level =** Current level of nodes
6. **Fval =** Evaluation function value

Functions

Class Name: Eight_Puzzle

Functions:

1. **get_values()**
   - Accepts values for Initial and Goal state from user

2. **h_misplaced_tiles()**
   - Function calculates the misplaced tiles(values) between the Initial and Goal state

3. **h_manhattan_dist()**
   - Function that calculates the manhattan distance between the values of Initial and Goal state

4. **f_manhattan_dist()**
   - It calculates the sum of heuristic value
   - f(x) = g(x) + h(x)
   - g(x) = level order
   - h(x) = manhattan distance or misplaced tiles

5. **f_misplaced_tiles()**
   - It calculates the sum of heuristic value
   - f(x) = g(x) + h(x)
   - g(x) = level order
   - h(x) = manhattan distance or misplaced tiles

6. **f1()**
   - It makes a copy of the f(x) function

7. **stage_process()**
   - This function takes the Initial and Goal state from the user.
   - Further it calls the other function to expand the children of parent to reach the goal
   - It also removes the current state node and adds to explored state
   - Finally it checks if the goal state is reached or not

Class Name: State

Functions:

1. **generate_children()**
    - The function generates the children of the parent by moving the zero(0) element in the feasible position
    - There are 4 possible moves : left,up,right,down
    - Feasible position depends on where the element zero(0) is placed.
    - If it is corner then 2 possible moves.
    - If it is an edge then 3 possible moves.
    - If it is a center then 4 possible moves.

2. **search_space()**
    - Function searches for the position of the specific element in our case it is zero(0)

3. **valid_position()**
    - Checks for the valid position of the zero(0) element and returns it.

4. **copy_matrix()**
    - Makes the copy of matrix

Procedures

Phase 1:

Initially, create the object of the class **Eight_Puzzle** and pass the size of the input. Constructor of the class will initialize the **size of the matrix**and two lists **explored and fringe.**Then it calls the function **stage_process** where user enters the value of the initial and goal state. Then user chooses the heuristic function i.e. Manhattan distance or the Misplaced tiles(value).

Phase 2:

Function **f_manhattan_dist** or**f_misplace_tiles** will calculate the value of **Evaluation function f(x)**.Once the value is calculated the node is added to the fringe list. Now it will call the functions **generate_children, search_space and valid_position** successively. Initially it finds the position of the element **"0".** Then it checks for the valid position or not. Now there are theoretically three cases possible here:

1. If it is a **corner** then element "0" can move in 2 ways
2. If it is an **edge** then element "0" can move in 3 ways
3. If it is at the **center** then element "0" can move in 4 ways

But in code we just find the four different index i.e. **up, down, left and right**from element "0" and check it the index lies in the size of the matrix or not i.e. the index is valid or not. All the valid index get appended in the list.

Phase 3:

Now depending on the list, it will explore the given state and generate the children. Again, evaluation function gets called and calculates the value of each path i.e. parent->child value i.e. **calculates the f(x) by summing up with g(x) and h(x).**

Then it **removes the node from fringe list and added in the explored list**.Further it sorts the generated children in the ascending order of the evaluation function value.

Then it selects the first node from it and perform the operation until the initial state and goal state is not matched.

Analyzing four input /output cases:

First Case:

Initial State:

1	2	3
8	0	4
7	6	5

Goal state:

1	3	4
8	0	5
7	2	6

1. **Solution path from initial state to goal state using Manhattan Distance**

1	2	3
8	0	4
7	6	5

↓

1	0	3
8	2	4
7	6	5

1	3	0
8	2	4
7	6	5

1	3	4
8	2	0
7	6	5

1	3	4
8	2	5
7	6	0

1	3	4
8	2	5
7	0	6

Goal State:

1	3	4
8	0	5
7	2	6

g(x): 6

h(x): 0.0

f(x): 6.0

Path Cost: 6.0

Generated Nodes: 12

Expanded Nodes: 6

2. Solution path from initial state to goal state using Misplaced Tiles

1	2	3
8	0	4
7	6	5

1	2	3
8	4	0
7	6	5

1	0	3
8	2	4
7	6	5

1	2	3
8	6	4
7	0	5

1	2	3
8	4	5
7	6	0

1	3	0
8	2	4
7	6	5

1	2	3

8	4	5
7	0	6

1	3	4
8	2	0
7	6	5

1	3	4
8	2	5
7	6	0

1	3	4
8	2	5
7	0	6

1	3	4
8	0	5
7	2	6

g(x): 6
h(x): 0.0
f(x): 6.0
Path Cost: 6.0
Generated Nodes: 19
Expanded Nodes: 10

Second case:

Initial State:

4	1	3
0	2	6
7	5	8

Goal state:

1	2	3
4	5	6
7	8	0

1. **Solution path from initial state to goal state using Misplaced Tiles**

4	1	3
0	2	6
7	5	8

0	1	3
4	2	6
7	5	8

1	0	3
4	2	6
7	5	8

1	2	3
4	0	6
7	5	8

1	2	3
4	5	6
7	0	8

Goal State

1	2	3
4	5	6
7	8	0

g(x): 5
h(x): 0.0
f(x): 5.0
Path Cost: 5.0
Generated Nodes: 11
Expanded Nodes: 5

(2)Solution path from initial state to goal state using Manhattan DistanceTiles

4	1	3
0	2	6
7	5	8

↓

0	1	3
4	2	6
7	5	8

1	0	3
4	2	6
7	5	8

1	2	3
4	0	6
7	5	8

1	2	3
4	5	6
7	0	8

Goal State:

1	2	3
4	5	6
7	8	0

g(x): 5
h(x): 0.0
f(x): 5.0
Path Cost: 5.0
Generated Nodes: 11
Expanded Nodes: 5

Third case:

Initial State:

3	2	1
4	5	0
6	7	8

Goal State:

3	2	1
0	4	5
6	7	8

1. **Solution path from initial state to goal state using Manhattan Distance**

3	2	1
4	5	0
6	7	8

3	2	1
4	0	5
6	7	8

3	2	1
0	4	5
6	7	8

g(x): 2
h(x): 0.0
f(x): 2.0
Path Cost: 2.0
Generated Nodes: 6
Expanded Nodes: 2

2. Solution path from initial state to goal state using Misplaced Tiles

3	2	1
4	5	0
6	7	8

3	2	1
4	0	5
6	7	8

3	2	1
0	4	5
6	7	8

g(x): 2
h(x): 0.0
f(x): 2.0
Path Cost: 2.0
Generated Nodes: 6
Expanded Nodes: 2

Fourth case:

Initial State:

1	3	4
8	6	2
0	7	5

Goal state:

1	2	3
8	0	4
7	6	5

(1)Solution path from initial state to goal state using Manhattan distance

1	3	4
8	6	2
0	7	5

1	3	4
8	6	2
7	0	5

1	3	4
8	0	2
7	6	5

1	3	4
8	2	0
7	6	5

1	3	0
8	2	4
7	6	5

1	0	3
8	2	4
7	6	5

1	2	3
8	0	4
7	6	5

g(x): 6

h(x): 0.0

f(x): 6.0

Path Cost: 6.0

Generated Nodes: 12

Expanded Nodes: 6

(2) Solution path from initial state to goal state using Misplaced Tiles

1	3	4
8	6	2
0	7	5

1	3	4
8	6	2
7	0	5

1	3	4
8	0	2
7	6	5

1	3	4
8	2	0
7	6	5

1	0	4
8	3	2
7	6	5

1	3	0
8	2	4
7	6	5

1	0	3
8	2	4
7	6	5

1	2	3
8	0	4
7	6	5

g(x): 6
h(x): 0.0
f(x): 6.0
Path Cost: 6.0
Generated Nodes: 14
Expanded Nodes: 7

Summary of four input /output cases:

	Initial State	Goal State	Path Cost	Generated Nodes	Expanded Nodes
Manhattan Distance	1 2 3 8 0 4 7 6 5	1 3 4 8 0 5 7 2 6	6	12	6
Misplaced Tiles			6	19	10
Manhattan Distance	4 1 3 0 2 6 7 5 8	1 2 3 4 5 6 7 8 0	5	11	5
Misplaced Tiles			5	11	5
Manhattan Distance	3 2 1 4 5 0 6 7 8	3 2 1 0 4 5 6 7 8	2	6	2
Misplaced Tiles			2	6	2
Manhattan Distance	1 3 4 8 6 2 0 7 5	1 2 3 8 0 4 7 6 5	6	12	6
Misplaced Tiles			6	14	7