ITCS 6150 – Intelligent Systems

Programming Project 1

Archit Parnami

(800960108)

SOLVING THE 8-PUZZLE USING A* ALGORITHM

1. Problem Formulation

- The 8-puzzle problem consists of a 3x3 board with eight numbered tiles and blank space. A tile adjacent to blank space can slide into the space. The object is to reach a specified goal state.

1	2	3
7	5	6
8	4	

Start State

1	2	3
4	5	6
7	8	

Goal State

- **State:** A 3x3 matrix is used to specify the contents of a tile in each location. A blank tile is represented by 0.

1	2	3
7	5	6
8	4	0

- **Initial State:** any user given set of numbers (0-8) can be represented as initial state.
- **Actions**: Movements of the blank space (0) *Left*, *Right*, *Up* or *Down*. Different subsets of these are possible depending on where the blank is.
- **Transition model**: Given a state and action, this returns the resulting state; for example if we apply *Left* to the start state, the resulting state has the 4 and the blank switched.
- **Goal test**: This checks whether the state matches the goal state.
- **Path cost g(n)**: Each step costs 1, so the path cost is the number of steps in the path to reach state n.
- **Heuristic function h(n)**: Estimated cost of the cheapest cost of reaching the goal state from the current state n. For 8 puzzle problem we use **Hamiltonian distance** as the heuristic.
- **Evaluation function f(n):** Sum of cost of reaching state n from initial state and estimated cost of reaching the goal state from state n i.e. sum of path cost and heuristic function.

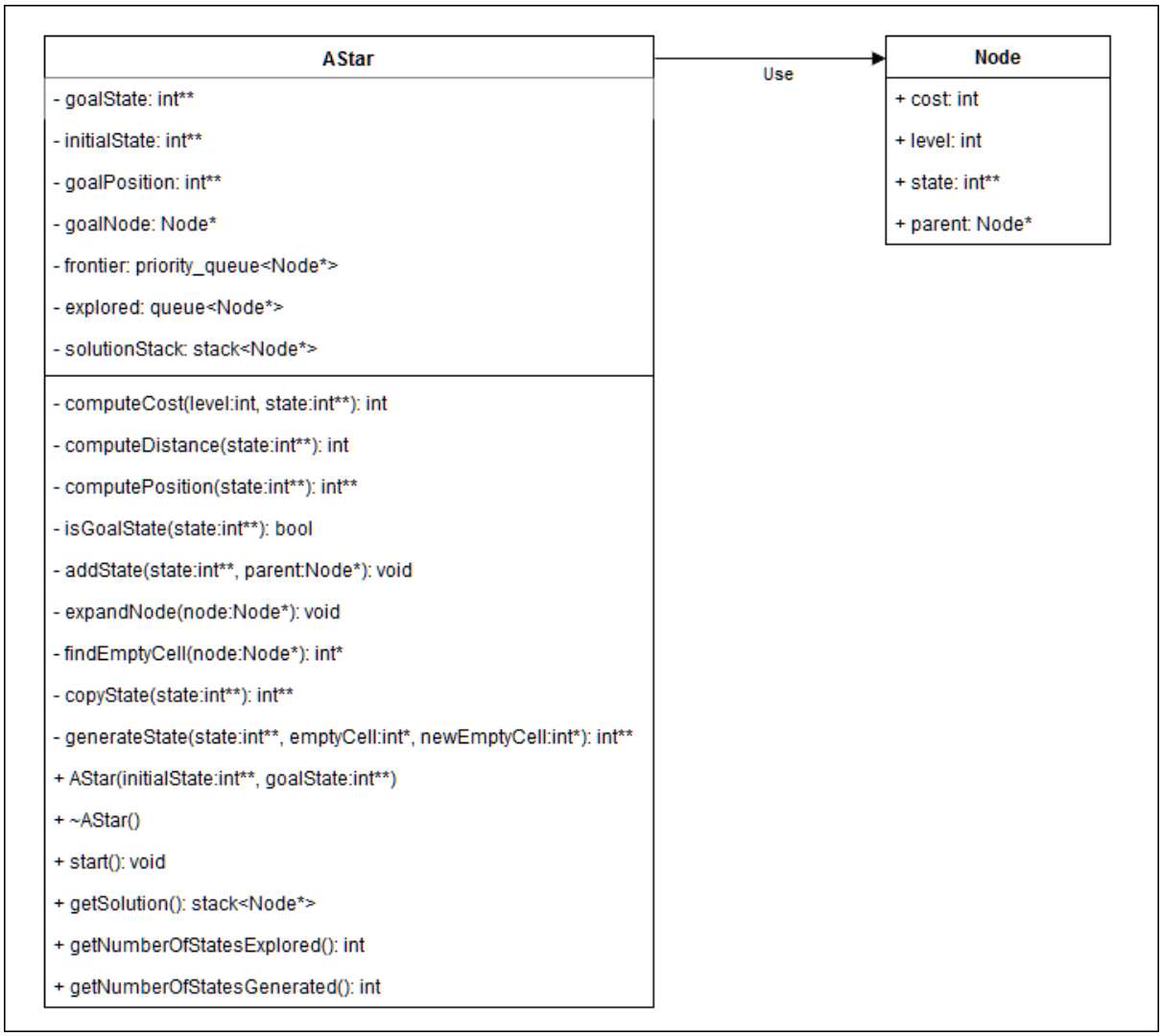
$$f(n) = g(n) + h(n)$$

2. The Program Structure

Programming Language: C++
IDE: Microsoft Visual Studio Community 2015
Files:

- **AStar.h :** This file defines the blueprint of two classes i.e. *Node* and *AStar*
- **AStar.**cpp: This file implements all the functions declared in AStar.h for the class AStar
- **Main.**cpp: This file is the entry point of the application. On execution it takes user input and creates an object of type AStar and begins the search on calling start().

Class Diagram:



Function Descriptions

1. ***computeCost(level:int, state:int**): int***
   This is the evaluation function f(n) which returns the cost of selecting the state n given its level (depth) and the current state.

2. ***computeDistance(state:int**): int***
   Heuristic function which returns the estimated cost of reaching the goal state from given state in terms of Hamiltonian distance.
3. **computePosition(state:int**): int****
   Given a state, this function returns the position of each tile in terms of (x, y) coordinates.
4. ***isGoalState(state:int**): bool***
   Verifies whether the given state is a goal state or not.
5. ***addState(state:int**, parent:Node*): void***
   Creates a node from the given state and attaches it to the parent node. Also adds the newly created node to the frontier queue.
6. ***expandNode(node:Node*): void***
   Expands the input node and generate the next possible combinations of states. Also adds the generated nodes to the frontier.
7. ***findEmptyCell(node:Node*): int****
   Return the location of blank tile in the 3x3 matrix in terms of (x, y) coordinate.
8. ***copyState(state:int**): int****
   Returns a copy of an input state. Helper function used for generating new states.
9. ***generateState(state:int**, emptyCell:int*, newEmptyCell:int*): int****
   Creates a new 3x3 matrix using the copyState function and swaps the contents of emptyCell with newEmptyCell to generate new state.
10. ***AStar(initialState:int**, goalState:int**)***
    Constructor for instantiating an object of type AStar. Initializes the problem with given initial and goal states.
11. ***~AStar()***
    Destructor for deallocating all the memory allocated during the execution of the program. Deletes all the nodes in the frontier queue and the explored queue.
12. ***getSolution(): stack<Node*>***
    Return a stack which contains order of nodes to be traversed to reach the solution for the 8-puzzle. The last node in the stack is the goal state and the first node is the start state.
13. ***getNumberOfStatesExplored(): int***
    The number of nodes actually expanded to reach the goal state i.e. the size of explored queue.
14. ***getNumberOfStatesGenerated(): int***
    The number of nodes generated all along. It is the sum of the number nodes in frontier queue, the number of nodes in explored queue and 1 (the goal state).
15. ***start():void***
    Starts the search for solution. Entry point of the search.

3. Results

- **Example 1**

2	8	3
1	6	4
7	0	5

Start State

1	2	3
8	6	4
7	5	0

Goal State

Number of states expanded: 10
Number of states generated: 21
Moves required to goal state: 7
(Cost): f(n)

Solution:

2	8	3
1	6	4
7	0	5

0(5)

2	8	3
1	0	4
7	6	5

1(7)

2	0	3
1	8	4
7	6	5

2(7)

0	2	3
1	8	4
7	6	5

3(7)

1	2	3
0	8	4
7	6	5

4(7)

1	2	3
8	0	4
7	6	5

5(7)

1	2	3
8	6	4
7	0	5

6(7)

1	2	3
8	6	4
7	5	0

7(7)

- **Example 2**

5	4	0
6	1	8
7	3	2

Start State

1	2	3
4	0	5
6	7	8

Goal State

Number of states expanded: 511
Number of states generated: 884
Moves required to goal state: 22

Solution:

5	4	0
6	1	8
7	3	2

0(16)

5	0	4
6	1	8
7	3	2

1(18)

5	1	4
6	0	8
7	3	2

2(18)

5	1	4
6	8	0
7	3	2

3(20)

5	1	4
6	8	2
7	3	0

4(20)

5	1	4
6	8	2
7	0	3

5(20)

5	1	4
6	0	2
7	8	3

6(20)

5	0	4
6	1	2
7	8	3

7(22)

5	4	0
6	1	2
7	8	3

8(22)

5	4	2
6	1	0
7	8	3

9(22)

5	4	2
6	1	3
7	8	0

10(22)

5	4	2
6	1	3
7	0	8

11(22)

5	4	2
6	1	3
0	7	8

12(22)

5	4	2
0	1	3
6	7	8

13(22)

0	4	2
5	1	3
6	7	8

14(22)

4	0	2
5	1	3
6	7	8

15(22)

4	1	2
5	0	3
6	7	8

16(22)

4	1	2
0	5	3
6	7	8

17(22)

0	1	2
4	5	3
6	7	8

18(22)

1	0	2
4	5	3
6	7	8

19(22)

1	2	0
4	5	3
6	7	8

20(22)

1	2	3
4	5	0
6	7	8

21(22)

1	2	3
4	0	5
6	7	8

22(22)

- **Example 3**

7	2	4
5	0	6
8	3	1

Start State

0	1	2
3	4	5
6	7	8

Goal State

Number of states expanded: 2706
Number of states generated: 4550
Moves required to goal state: 26

Solution:

7	2	4
5	0	6
8	3	1

0(18)

7	2	4
0	5	6
8	3	1

1(18)

0	2	4
7	5	6
8	3	1

2(18)

2	0	4
7	5	6
8	3	1

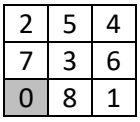
3(20)

2	5	4
7	0	6
8	3	1

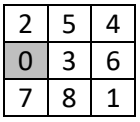
4(22)

2	5	4
7	3	6
8	0	1

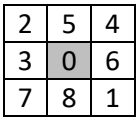
5(22)



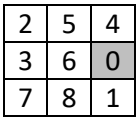
6(22)



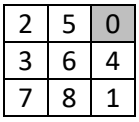
7(22)



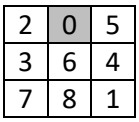
8(22)



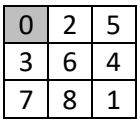
9(22)



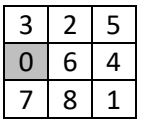
10(22)



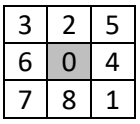
11(22)



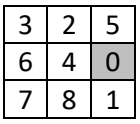
12(22)



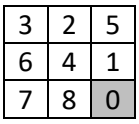
13(24)



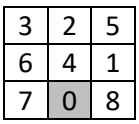
14(24)



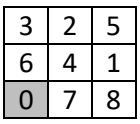
15(24)



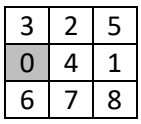
16(24)



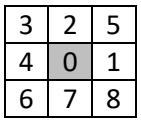
17(24)



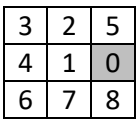
18(24)



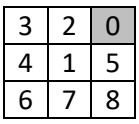
19(24)



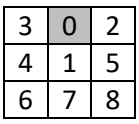
20(26)



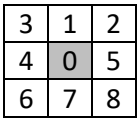
21(26)



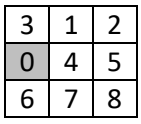
22(26)



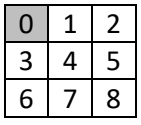
23(26)



24(26)



25(26)



26(26)