Assignment 1
Using Informed and Uninformed Search Algorithms to Solve 8-Puzzle

1 Overview

This problem appeared as a project in the edX course ColumbiaX: CSMM.101x Artificial Intelligence (AI). In this assignment an agent will be implemented to solve the 8-puzzle game.

An instance of the 8-puzzle game consists of a board holding 8 distinct movable tiles, plus an empty space. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0.
Given an initial state of the board, the search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order 0,1,2,3,4,5,6,7,8 .
The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions 'Up', 'Down', 'Left', 'Right', one move at a time. The cost of moving from one configuration of the board to another is the same and equal to one. Thus, the total cost of path is equal to the number of moves made from the initial state to the goal state.

Suppose the program is executed starting from the initial state 1,2,5,3,4,0,6,7,8 as follows:

$parent =$ | 1 | 2 | 5 |
|---|---|---|
| 3 | 4 | |
| 6 | 7 | 8 |

$\Longrightarrow$ $child =$ | 1 | 2 | |
|---|---|---|
| 3 | 4 | 5 |
| 6 | 7 | 8 |

$parent =$ | 1 | 2 | |
|---|---|---|
| 3 | 4 | 5 |
| 6 | 7 | 8 |

$\Longrightarrow$ $child =$ | 1 | | 2 |
|---|---|---|
| 3 | 4 | 5 |
| 6 | 7 | 8 |

$parent =$ | 1 | | 2 |
|---|---|---|
| 3 | 4 | 5 |
| 6 | 7 | 8 |

$\Longrightarrow$ $child =$ | | 1 | 2 |
|---|---|---|
| 3 | 4 | 5 |
| 6 | 7 | 8 |

2 Application

2.1 Algorithms

You will need to implement the 8 puzzle search problem using the three search algorithms: BFS ,DFS and A*

BFS search

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

 return FAILURE
```

DFS search

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.push(neighbor)

    return FAILURE
```

Taken from the edX course ColumbiaX: CSMM.101x Artificial Intelligence (AI)

A* search

Taken from the edX course ColumbiaX: CSMM.101x Artificial Intelligence (AI)

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) + h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

2.2 Heuristics

For the A* (the informed search) we are going to use Manhattan heuristic and Euclidean heuristic and compare between number of nodes expanded and output paths, and to report which heuristic is more admissible.

1. Manhattan Distance

   It is the sum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively,

   $$h = \text{abs}(\text{current\_cell:x} - \text{goal:x}) + \text{abs}(\text{current\_cell:y} - \text{goal:y})$$

2. Euclidean Distance

   It is the distance between the current cell and the goal cell using the distance formula

   $$h = \text{sqrt}((\text{current\_cell:x} - \text{goal:x})^2 + (\text{current\_cell.y} - \text{goal:y})^2)$$

3 Notes

You will be given a random initial state and your code should output the goal state 0,1,2,3,4,5,6,7,8 The output should be in a traceable format; as every step should be printed.(best visualization of the puzzle table will be grant a bonus).

3.1 Deliverables

- Your well commented code.
- A report showing your work,
  including:
  - path to goal
  - cost of path
  - nodes expanded
  - search depth
  - running time

Also the report should contain the data structure used (if any) and algorithms, Assumptions and details you find them necessary to be clarified, Any extra work and Sample runs. You should show your algorithm and how it operate.

3.2 Further Notes

- You may use Java, Python or C++ for your implementation.
- Copied assignments will be severely penalized.
- You can work in groups of 3 or 4.

4 References

- Tutorial 1
- Tutorial 2
- edX course ColumbiaX: CSMM.101x Artificial Intelligence (AI)

Good Luck