Lab #5: Informed search (cont.)

The main aim of this lab is to solve the problem of 8 puzzle using **Greedy Best First Search** and **A* Search**.

For a given states of the 8 puzzle problem as follows:

1	5	3
4		8
7	2	6

Initial State

1	2	3
4	5	6
7	8	

Goal State

Two measures that can be used for computing heuristic:

- h_1: The number of tiles out of place
- h_2: Sum of distances of tiles from goal positions, using Manhattan distance

The computation of these measures is described in the following table:

Tile	Index in Initial State	Index in Goal State	h_1	h_2
1	(1,1)	(1,1)	0	0
2	(3,2)	(1,2)	1	2
3	(1,3)	(1,3)	0	0
4	(2,1)	(2,1)	0	0
5	(1,2)	(2,2)	1	1
6	(3,3)	(2,3)	1	1
7	(3,1)	(3,1)	0	0
8	(2,3)	(3,2)	1	2
SUM			4	6

The costs used in Greedy, and A* is as follows:

▸ **Greedy best first search**: expand the node that is closest to the goal

$$f(n) = h(n)$$

▸ **A* search**: combine UCS and Greedy (minimizing the total estimated solution cost)

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

==

The main aim of algorithm for solving 8 puzzle is based on moving white tile (UP, DOWN, LEFT, RIGHT) to get new successors. Then using the measure $f(n) = g(n) + h(n)$ (as in A* algorithm) or $f(n) = h(n)$ (in the case of Greedy Best First Search) to choose the next state - with the lowest f(n) to expand. The procedure is repeated until f(n) = 0.

In this lab, the initial state and the goal state are represented in **PuzzleMap.txt** and **PuzzleGoalState.txt.** The code for loading these states was implemented in **Puzzle** class.

Note: Ignore moves that return you to the previous state

Task 1. Implement the following methods in **Puzzle** class:

```
//Move white tile UP, DOWN, LEFT, RIGHT. Remember to check when a tile is
//out of the map.
public Node moveTile(Node currentState, char operator) {...}

public int computeH1(Node currentState) {...}

public int computeH2(Node currentState) {...}
```

The frontier is a PriorityQueue with comparator implementations are defined in PuzzleUtils class:

```
PriorityQueue<Node> frontier = new
PriorityQueue<Node>(PuzzleUtils.HeuristicComparatorByH);
```

The pseudo code based on graph search that can be used**:**

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
    initialize the frontier using the initial state of *problem*
    initialize the explored set to be empty
    loop do
        if the frontier is empty **then return** failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state **then return** the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
   *only if not in the frontier or explored set*

Task 2. Solve the 8 puzzle problem using **Greedy Best First Search** with *h1 or h2* introduced in the previous section (implements IPuzzleAlgo interface).

Task 3. Solve the 8 puzzle problem using A* Search with $h(n)$, $g(n)$ as the number of steps traversed from start node to get to the current node, function h could be $h1$ or $h2$ as introduced in the previous section.

Case 1: f(n) = g(n) + h₁(n)
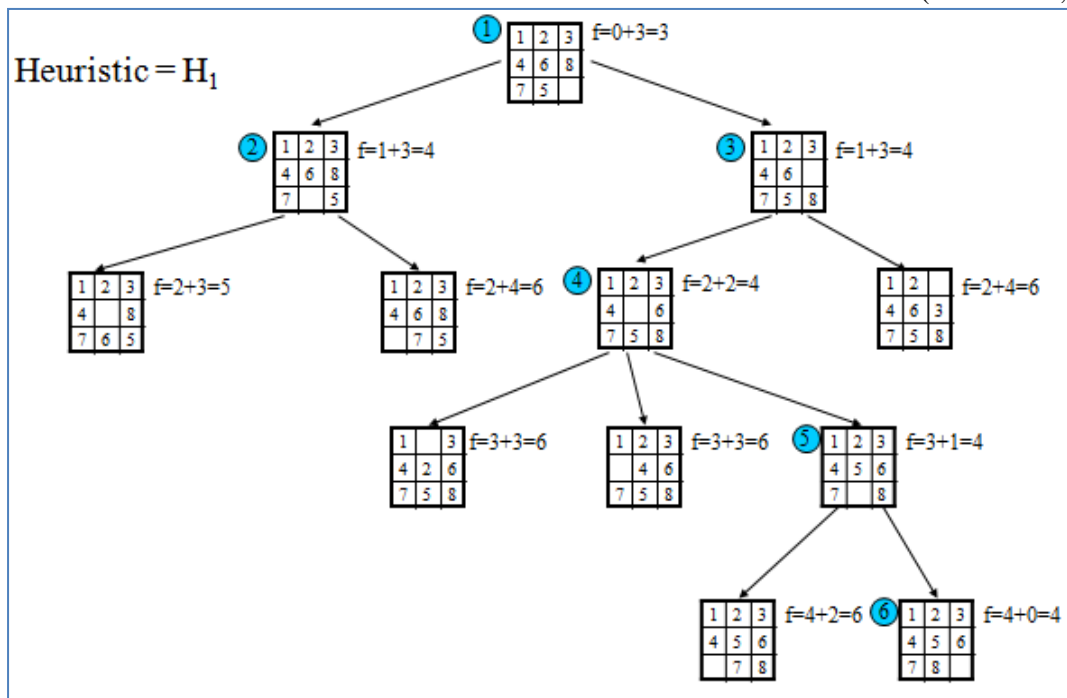
Case 2: f(n) = g(n) + h2(n)
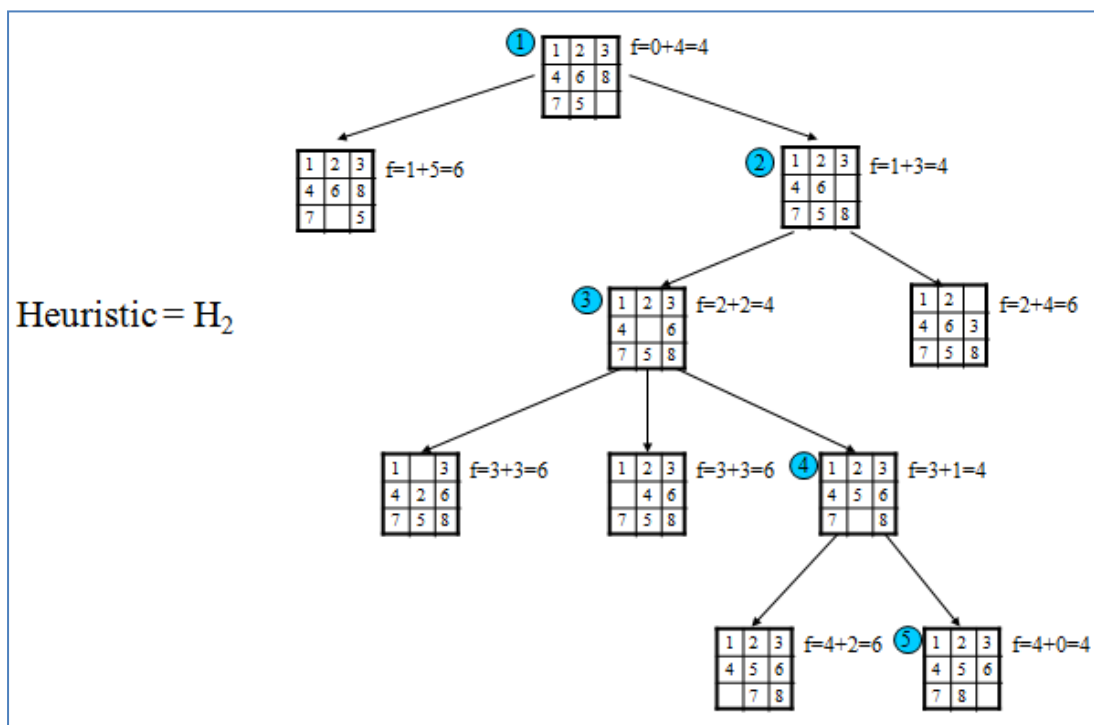
Task 4. (**Advanced**) Apply other alogrithms to solve the 8 puzzle problem such as _DFS_, _BFS_ and _local search_ beside _Greedy best first search_ and _A*_ algorithms (implemented in the previous tasks). Then, present a comparison of selected algorithms in terms of **running time**, **the number of steps** to reach the goal state from the initial state.