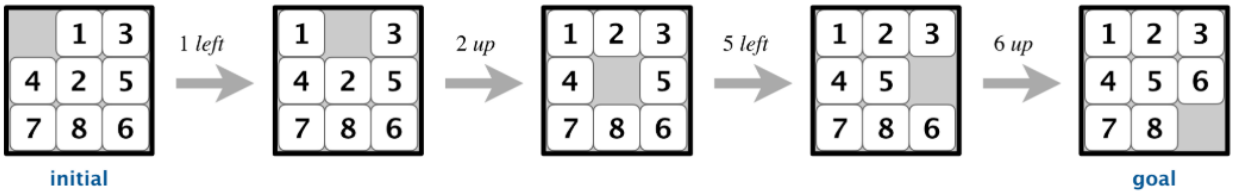
Computer Science 2920

Assignment 2 - Due Friday March 1 @11:55pm

The 8-puzzle is a sliding puzzle that is played on a 3-by-3 grid with 8 square tiles labeled 1 through 8, plus a blank square. The goal is to rearrange the tiles so that they are in row-major order, using as few moves as possible. You are permitted to slide tiles either horizontally or vertically into the blank square. The following diagram shows a sequence of moves from an initial board (left) to the goal board (right).



1. Implement Best-First-Search algorithm

In the previous assignment you implemented a Board class to represent a state of the 8-Puzzle game. You also implemented a Depth-First-Search (DFS) and Breadth-First-Search (BFS) algorithm. In this assignment we want to generalize these search strategies with a Best-First-Search algorithm (not abbreviated for obvious reasons).

In a Best-First-Search we start by generating a board for the initial state of the 8-puzzle. Every time a new node (board state) is generated it will be assigned a value indicating how "good" that state. For every generated node we will push it into a Priority Queue (PQ) using the "goodness" as the priority.

After generating the initial state node and pushing it into the PQ the Best-First-Search algorithm will keep iterating until the goal state is found or a given stopping condition (e.g. a maximum running time is reached). In each iteration the Best-First-Search algorithm will remove the first board in the PQ and process that board. Processing a board means checking whether that board is the goal state, if that is the case the algorithm will terminate and return the sequence of boards (steps) from the initial state to the goal state. If the removed board is not the goal board then the neighbors of that board will be generated, for each neighbor its priority will be calculated and they will be inserted in the PQ. Then the next iteration will begin by removing again the first element in the queue.

In order to easily return the path from a given state to the initial state consider modifying the Board class to accommodate a field containing the states-path traversed before generating that specific board configuration. Updating the path of a new state can be done directly when creating the Neighbors or with a specific UpdatePathToStartState(). If the final state is reached the algorithm should return the corresponding board instance (which should include the path from the initial state), otherwise, If the algorithm fails to find a solution (stopping condition is reached before goal state) then should return null.

Implement an abstract class BestFirstSearch where the CalculatePriority() method is declared abstract. The class should provide a Run method that executes the Best-First-Search algorithm. The run method should receive as an argument the number maximum millisecond to run the algorithm and it should be used as the stopping condition.

Implement two subclasses of BestFirstSearch which must provide a concrete implementation of the CalculatePriority() method:

1. BreadthFirstSearch: Calculate the priority as the depth of the board in the game tree (the same as the number of actions performed to reach that state).
2. HeuristicSearch: Calculate the priority as the number of tiles misplaced (i.e. in a different position from the goal state).

Here is a pseudocode of the BestFirstSearch algorithm:

```
BestFirstSearch(InitState, StoppingCondition)
 Create PQ
 p = CalculatePriority(InitiState)
 PQ.Insert(<InitiState, p>)
 While(StoppingCondition is not True)
 {
        nextState = PQ.RemoveFirst()
        if (nextState is goalState) return nextState
        foreach (neighbor in PQ)
        {
                Neighbor.UpdatePathToStartState()
                p = CalculatePriority(neighbor)
                PQ.Insert(<neighbor, p>)
        }
 }
```

2. **Test *BreadthFirstSearch* and *HeuristicSearch* algorithms.**

Write a code to run both algorithms with the following initial states and stopping conditions:

I. **Initial state:** [1, 2, 3; 4, 0 , 5; 7, 8, 6].
   Time limit: 2000 ms.

II. **Initial state:** [0, 1, 2; 4, 5, 3; 7, 8, 6].
Time limit: 2000 ms.

III. **Initial state:** [0, 8, 6; 2, 7, 5; 1, 3, 4].
Time limit: 2000 ms.

If the algorithm manages to find the goal state print the sequence the path from the initial state to the goal state. Otherwise print "The algorithm was unable to find the solution in the given time."

Submission Guidelines:

Submit your assignment as a single zip file through the moodle link provided. Use the following naming format for your zip file:

lastname-as2.zip

The zip file should contain the code of your solution.