CSCI 310 Advanced Algorithms
Instructor: Dr. Sebastian van Delden
Email: vandeldensa@cofc.edu

ASSIGNMENT 1 – INDIVIDUAL WORK ONLY!!!

Implement the **Eight Piece Puzzle** Program that utilizes the **A* Search Algorithm**, the **Manhattan Distance Heuristic**, and a **Priority Queue** (linked list and heap) that stores board states that are yet to be considered. You ***must*** follow the below details.

- We will implement both a linkedlist-based priority queue and a heap-based priority queue, and run time analysis reports using each of these on all possible goal states given a start state!!
- Skeleton code is provided for all of the below classes.

PROGRAM GUIDELINES THAT MUST BE FOLLOWED

Main Class:

- Simply creates a new EightPuzzle object
- It is passed either a linkedlist-based priority queue (PriorityQueueLinkedList) or a heap based priority queue (PriorityQueueHeap) to EightPuzzle's constructor.
    - The EightPuzzle constructor itself expects the abstract class PriorityQueue as an parameter which both of these above classes must implement. **Polymorphism is a beauty thing!!!**

EightPuzzle Class:

- See Skeleton code provided for you.
- The **Constructor** does all the work and calls the below helper methods are needed. You pass it the underlying data structure that you want to use for the priority queue. So nice!! You also pass it the start state as a 1D array of ints.
- The **checkReachable** method takes two states, as 1D arrays of ints of size 9, and returns true if one is reachable to the other.
- The **printPath** method MUST be recursive. It prints the path to the goal, and returns an int: number of states on the path.
- The **manhattan** method takes two 1x9 arrays of ints (board states) and returns the Manhattan distance between them.
- The **move** method takes a BoardState object and a row/column offset which must be a 0, 1, or -1. It returns the new board state based on the row/column values, or null if the move is not possible.
- The **makeKey** method converts the 1D array of ints inside a BoardState object to a String so that it can be used as the key value in the closedNodes HashMap which will store visited board states.
- The **generateAllPossibleStates** method populates the AllPossibleStates[362880][9] array which contains every possible permutations of a board state array.
- You can create additional methods if needed.

BoardState Class:

- Implements the Comparable interface - must include the compareTo() method.
- You need to implement compareTo(), equals() and toString(). See skeleton code.

PriorityQueue<T> Abstract Class:

- Provided for you. No changes needed. See skeleton code.

PriorityQueueLinkedList<T> Class:

- Extends the PrioirityQueue<T> abstract class that is provided to you with this assignment.
- You need to implement PriorityEnqueue and PriorityDequeue using a simple linked list strategy.

PriorityQueueHeap<T> Class :

- Extends the PrioirityQueue<T> abstract class that is provided to you with this assignment.
- You need to implement PriorityEnqueue and PriorityDequeue using a heap strategy.

Output Report – Linked List.docx:
Output Report – Heap.docx:

- You must populate both of these report documents with the requested information.
- See templates that have been provided to you along with this assignment.

OTHER DETAILS THAT MUST BE FOLLOWED

- A board state will be stored as a one dimensional array of nine integers: 0 through 9. Zero will represent the blank space on the board. At no point in the program should you use a 3 by 3 array of integers to store a board state. Doing so will result in the loss of 10 points for each usage.

SUBMISSION DETAILS

Upload a ZIP file to OAKS by the due date. This file must include all .java files and the two output reports. ***THIS ASSIGNMENT IS INDIVIDUAL WORK***. A score of zero will be assigned to all programs which contained portions of code that have been duplicated.