Solving the 8-puzzels problem using A* search, as taught in class and in lecture notes. (You may not use Java built-in linked list in this project!) The three A* functions you used in this program are:

g(n) - # of moves from initial state to node/state n
h*(n) - the total distance for all tiles to move from node/state n to the goal state.
f*(n) - g(n) + h*(n)

Create a few pairs of 8-puzzel configurations to test your program first, then, run your program with the given two pairs of test data: first pair: data1 and data2; 2nd pair: data3 and data4

Include in your hard copies:

- cover sheet
- source code
- print outFile1 for the first pair
- print outFile2 for the first pair
- print outFile1 for the second pair
- print outFile2 for the second pair

****

Language: Java

****

I. Inputs:

a) inFile1 (use args [0]) : A file contains 9 numbers, 0 to 8, represents the initial configuration of the 8-puzzel.
b) inFile2 (use args [1]) : A file contains 9 numbers, 0 to 8, represents the goal configuration of the 8-puzzel.

****

II. Outputs:

a) outFile1: (use args [2]) : For all intermediate Open list and Close list and expanded child list. b) outFile2: (use args [3]): For the display of the sequence of moves from initial state to the goal state.
Make a very nice display from each configuration to next configuration of 8-puzzels.

****

III. Data structure: *** YOU MAY NOT USE Java built-in linked list functions!

****

- AstarNode class // To represent an 8-puzzel node
  - configuration [9] - you can use an integer array of size 9 or a string length of 9.
  - (int) gStar // # moves so far from initial state to current state
  - (int) hStar // the estimated moves from the currentNode to the goal stateNode
  - (int) fStar // is gStar + hStar
  - (AstarNode) parent //points to its parent node; initially point to null

methods:

- constructor (…)

- printNode (node)
// print only node's fStar, configuration, and parent's configuration, in one text line.
For example: if node's fStar is 9, its configuration is 6 3 4 8 7 0 5 2 1
and its parent's configuration is 6 3 4 8 7 1 5 2 0
Then, print <9:: 6 3 4 8 7 0 5 2 1 :: 6 3 4 8 7 1 5 2 0>

- AStar class
- (AstarNode) startNode
- (AstarNode) goalNode
- (AstarNode*) Open // A sorted linked list with a dummy node.
// It maintains an ordered list of nodes, w.r.t. the fStar value
// Built your own linked list, you may not use Java built-in linked list.
- (AstarNode*) Close // a linked list with a dummy node, can be sorted or unsorted.
// It maintains a list of nodes that already been processed
- (AstarNode*) childList // a linked list Stack for the expend node's children.

methods: // all methods are on your own.

- (int) computeGstar (n) // equal to node's parent's gStar + 1 // one move
- (int) computeHstar (n) // count the total distance/moves of tiles from n to goalNode.

- (bool) match (configuration1, configuration2) // check to see if two configurations are identical. // if they are identical, returns true, otherwise returns false.

- (bool) isGoalNode (node) // to check if node's configuration is identical to goalNode's configuration. // you can call match () method, passing node's configuration with //goalNode's configuration.
- listInsert (node) // insert node into OpenList, in ascending order w.r.t. fStar

- (AstarNode) remove (OpenList) // removes and returns the front node of OpenList after dummy.

- (bool) checkAncestors (currentNode) // To avoid cycle; it starts from currentNode, call match () method //to see if currentNode's configuration is identical to its parent's, and recursively call // upward until reaches the startNode. If it matches with one of currentNode's ancestor, //returns true, otherwise return false.

- (AstarNode*) constructChildList (currentNode) // Constructs a linked list Stack for all children // of currentNode (i.e., all moves from currentNode, but NOT one of the currentNode's ancestors. // When finish, returns the linked list head. **<u>This method must call checkAncestors method!</u>**

- printList (listHead, outFile1) // call printNode () to print each node in OpenList, including dummy //node, one printNode per text line.

- printSolution (currentNode, outFile2) // Print the solution to outFile2, make it pretty to look

at. **** You may add more methods if needed.

****
IV. main ()
****
Step 0: inFile1, inFile2, outFile1, outFile2 open
initialConfiguration get from inFile1
goalConfiguration get from inFile2
startNode create a AstarNode for startNode with initialConfiguration

goalNode create a AstarNode for goalNode with goalConfiguration
Open create a linked list with a dummy node
Close create a linked list with a dummy node

Step 1: startNode's gStar 0
startNode's hStar computeHstar (StartNode)
startNode's fStar startNode's gStar + startNode's hStar
listInsert (startNode) // Insert startNode into Open, in ascending order w.r.t. fStar

Step 2: currentNode remove (Open)

Step 3: if (isGoalNode (currentNode))// A solution is found!
printSolution (node, outFile2)
return or exit the program
Step 4: childList constructChildList (currentNode)

Step 5: child pop (childList)

Step 6: child's gStar computeGstar (child)
child's hStar computeHstar (child)
child's fStar child's gStar + child's hStar

Step 7: if child is not in Open and not in Close
Insert child into Open
child's parent currentNode // back pointer
else if child is in Open and child's f* is better (<) than the old node's f* in Open
replace child with the old child in Open,
//i.e., do a delete and an insert
child's parent currentNode // back pointer
else if child is in Close and its f* is better (<) than the f* of old node on CloseList
remove child from Close
Insert child into OpenList
child's parent currentNode // back pointer

Step 8: repeat Step 5 to Step 7 until childList is empty

Step 9: push (currentNode, Close)

Step 10: Print "This is Open list:" to outFile1
printList (Open, outFile1)
Print "This is CLOSE list:" to outFile1
printList (Close, outFile1)
Print up to 30 loops!

Step 11: repeat step 2 to step 10 until currentNode is a goal node or OpenList is empty.

Step 12: if OpenList is empty but currentNode is NOT a goal node,
print error message: "no solution can be found in the search!" to

outFile1 Step 13: close all files