

Project 8: Solving the 8-puzzels problem using A\* search. The three heuristic functions are:

$g(n)$  - # of moves from initial state to node  $n$

$h^*(n)$  - # of misplaced tiles

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

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: C++

Project points: 12pts

Due Date: Soft copy (\*.zip) and hard copies (\*.pdf):

12/12 on time: 11/28/2020 Saturday before midnight

+1 early submission: 11/24/2020 Tuesday before midnight

-1 for 1 day late: 11/29/2020 0 Sunday before midnight

-2 for 2 days late: 11/30/2020 Monday before midnight

-12/12: after 11/30/2020 Monday after midnight

-6/12: does not pass compilation

0/12: program produces no output

0/12: did not submit hard copy.

\*\*\* Follow "Project Submission Requirement" to submit your project.

\*\*\*\*\*

I. Inputs:

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

b) inFile2 (use argv [2]) : A file contains 9 numbers, 0 to 8, represents the goal configuration of the 8-puzzel.

\*\*\*\*\*

II. Outputs:

a) outFile1: (use argv [3]) : For all intermediate Open list and Close list and expanded child list.

b) outFile2: (use argv [4]): 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:

\*\*\*\*\*

- AstarNode class // To represent an 8-puzzle node
    - configuration - you can use an integer array of size 9 or a string length of 9.
    - gStar (int) // # moves so far from initial state to current state
    - hStar (int) // the # of misplaced title from the currentNode the goal stateNode
    - fStar (int) // is gStar + hStar
    - parent (AstarNode\*) //points to its parent node; initially point to null
    - methods:
      - constructor (node)
      - printNode (node)
        - // print only node's configuration, fStar and parent's configuration, in one text line.
  - AStarSearch class
    - startNode (AstarNode)
    - goalNode (AstarNode)
    - OpenList (AstarNode\*) // A sorted linked list with a dummy node.
      - // It maintains an ordered list of nodes, w.r.t. the fStar value.
    - CloseList (AstarNode\*) // a linked list with a dummy node, can be sorted or unsorted.
      - // It maintains a list of nodes that already been processed
    - childList (AstarNode\*) // a linked list Stack for the expend node's children.
    - methods:
      - (int) computeGstar (node)
        - // equal to node's parent's gStar + 1 // one move
      - (int) computeHstar (node) // count # of misplaced tiles.
      - (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) // 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) // construct a linked list Stack. Each node (child of
      - //currentNode in the Stack is a possible move from currentNode, but NOT one of the
      - //currentNode's ancestors (call checkAncestors (...) method to check it out). Otherwise,
      - //your program will have an infinite loop!!! Also set each child's parent to currentNode.
      - //When finish, returns the linked list head.
    - printList (listHead, outFile1) // call printNode () to print each node in OpenList, including dummy
    - node, follow by an //arrow, "-->" after each node is printed.
    - printSolution (currentNode, outFile2) // Print the solution to outFile2, make it pretty to look at.
- \*\*\*\* You may add more methods if needed or not use any of the method list in the above.

\*\*\*\*\*

#### IV. main () // A\* algorithms

\*\*\*\*\*

Step 0: initialConfiguration  $\leftarrow$  get from inFile1

goalConfiguration  $\leftarrow$  get from inFile2

startNode  $\leftarrow$  create a AstarNode for startNode with initialConfiguration

goalNode  $\leftarrow$  create a AstarNode for goalNode with goalConfiguration

OpenList  $\leftarrow$  create a linked list with a dummy node

CloseList  $\leftarrow$  create a linked list with a dummy node

Step 1: startNode's gStar  $\leftarrow$  0

startNode's hStar  $\leftarrow$  computeMissTiles (StartNode)

startNode's fStar  $\leftarrow$  startNode's gStar + startNode's hStar

listInsert (startNode) // Insert startNode into OpenList, in ascending order w.r.t. fStar

Step 2: currentNode  $\leftarrow$  remove (OpenList)

Step 3: if (isGoalNode (currentNode))// A solution is found!

printSolution (node, outFile2)

return or exit the program

Step 4: childList  $\leftarrow$  constructChildList (currentNode)

Step 5: child  $\leftarrow$  pop (childList)

child's parent  $\leftarrow$  currentNode // back pointer

Step 6: child's gStar  $\leftarrow$  computeGstar (child )

child's hStar  $\leftarrow$  computeHstar (child)

child's fStar  $\leftarrow$  child's gStar + child's hStar

Step 7: if child is not in OpenList and not in CloseList

Insert child into OpenList

else if child is in OpenList and child's f\* is better ( $<$ ) than the old node's f\* in OpenList

replace child with the old child in OpenList,

//i.e., do a delete and an insert

else if child is in CloseList and its f\* is better ( $<$ ) than the f\* of old node on CloseList

remove child from CloseList

Insert child into OpenList

Step 8: Print "This is Open list:" to outFile1

printList (OpenList, outFile1)

Print "This is CLOSE list:" to outFile1

printList (CloseList, outFile1)

Step 9: repeat step 2 to step 8 until currentNode is a goal node or OpenList is empty.

Step 10: if OpenList is empty but currentNode is NOT a goal node, print error message to outFile1

Step 11: close all files