Project 9: Solving the Traveling salesman's problem using A\* search.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Language: C++

Due date: Soft copy: 12/8/2018 Saturday before midnight

+2 Early submission 12/3/2018 Monday before midnight

+1 Early submission 12/5/2018 Monday before midnight

Hard copy for early submission: 12/6/2017 Thursday in class.

Hard copy for early submission: 12/11/2017 Tuesday in class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I. Input-1 (use argv) : A file, represents an undirected graph,

by a list of edges with costs, <ni, nj, c>,

(a bi-directional edge from ni to nj and from nj to nj where the edge cost is c.)

The first text line is the number of nodes, N, in the graph G,

follows by a list of triplets, <ni, nj, cost>

For example:

5 // there are 5 nodes in the graph

1 5 10 // a bi-directional edge <1, 5, 10> & <5, 1, 10)

2 3 5 // an edge <2, 3, 5>

1 2 20 // an edge <1, 2, 20>

3 5 2 // an edge <3, 5, 2>

:

:

Input-2: from console: start node, given by the user from console.

Input-3: from console: which h\* function to compute the h\* value, (1, 2 or 3)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

II. outFile\_1 (use argv) :

The result of A\* search of a \*simple-path\* that begins at start node and

back to start node such that:

The format of Output-1 should be as follows (assume there are only 5 nodes):

\*\* Below is the input graph for this program \*\*

5

1 5 10

2 3 5

1 2 3

3 5 2

:

:

\*\* The start node is ... (fill in the user given start node)

\*\* h\* function used is choice 1 or 2 or 3

\*\* The search result of the path:

Starts from node 3 using A\* search (assuming the start node given is 3)

3 5 2 // from 3 to 5 cost 2

5 4 6 // from 5 to 4 cost 6

:

:

:

1 3 7 // from 1 to 3 cost 7

The total cost of the simple-path is ... (fill in the total cost)

III. outFile\_2 (use argv): For all debugging outputs to help you

**debugging only. DO \*NOT\* include in your hard copies!**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IV. Data structure:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- AstarNode class // For nodes in the graph

- ID (int)

- numEdgeFromStart (int) // initallized to 0

// to keep track the number of edges from Start node to this node

- gStar (int) // the total cost from Start node to this node

- hStar (int) // the estimate cost from this node to a goal node

- fStar (int) // is gStar + hStar

- next (AstarNode\*)

- parent (AstarNode\*) //points to its parent node)

methods:

- constructor (node) // the constructor create a AstarNode,

// you may op to perform

// the computation of gStar, hStar, and fStar functions

// within the constructor or let the computations perform

// outside of the constructor

- AStarSearch class

- numNodes (int) // get from input file

- start (int) // the start node ID given by the user from console

- whichHFunction (int) // ask the user from console : only can be 1, or 2 or 3

- costMatrix [ ][ ] // a 2-D array size, a cos matrix to represent the input graph

// initiallize to -1 to denote as infinity cost and then

// set costMatrix[i][i] to 0

\

- OpenList (AstarNode\*) // OpenList initally should point to a dummy node.

// It maintains an ordered list of nodes, w.r.t. the fStar value.

// delete from the front of the queue,

// insert into queue by the node's fStart value, NOT at the back.

- CloseList (AstarNode\*) // CloseList is a linked list stack, it does not need a dummy node.

// CloseList maintains a list nodes that already been processed

- childAry (int \*) // an 1-D array of size numNodes

methods:

- constructor

- loadMatrix (…)// load input-1 to costMetrix

- copyChildList (index) // copy costMatrix[index] to childAry

- (int) computeHstar (AstarNode\* node, whichHFunction )

// You will learn three (3) Hstar functions in the class

// h1Star, h2Star, and h3Sta.

- (int) computeGstar (AstarNode\* node )

// equal to node's parent's Gstar + cost from parent to this node

- (int) computeFstar (AstarNode\* node)

// equal to node's Hstar + node's Gstar

- IsOnCloseList (AstarNode\* currentNode, AstarNode\* oldNode)

// It search closeList (check ID) to see if currentNode is in the closeList or not.

// if currentNode is NOT in CloseList, returns false

// else if currentNode is in CloseList, returns true AND set oldNode to

// the SPOT where SPOT points to the node in closeList that has the

// same ID as currentNode.

- OpenInsert (…) // insert node into OpenList w.r.t. node's fStar value.

- printOpen (…) // debug print to outFile\_2:

// with a caption: \*\*\* OPEN list \*\*\*

// print only 6 nodes after dummy, or less if there are less than 6 nodes.

- printClose (…)// debug print to outFile\_2:

// with a caption: \*\*\* CLOSE list \*\*\*

// print only 6 nodes after dummy, or less if there are less than 6 nodes.

- OpenRemove (…) // removes the front node, after dummy, from OpenList

// and returns the front node.

- ClosePush (AstarNode\* node) // push node on the top of the CloseList

- CloseDelete (AstarNode\* node) // delete node from the CloseList

- checkPath (AstarNode\* node) // returns true if all nodes in the graph is

// on the path (should have no repeated node!!

// , i.e., we have found a solution!

// returns false otherwise.

// \*\*\* This method traverses the path from the given

// node via its parent pointer

// upward until reaches the start node with startID.

// You may check numEdgeFromStart it should equal the

// numNodes, but if your program has bugs, you may not want to

// depends on this!

- tracePath (AstarNode\* node) // when node passes checkPath, this method

// outputs to OutPut-1, the path with cost of each edge from node to SNode.

// with the format given in the output specs

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

V. Main(…) A start algorithm steps for the traveling salesman's program

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: open all files

numNodes <-- read from input-1

startID <-- get from the user from console

// make sure startID is within the range of the nodes in graph!!

// if out of range, your program must inform the user!!!

whichHFunction <-- startID <-- get from the user from console;

// make sure whichHFunction is 1, 2, or 3,

// if out of range, your program must inform the user!!!

step 1: dynamically allocate costMatrix size of numNodes by numNodes

dynamically allocate childAry size of numNodes

loadMatrix (…) // read from input-1

step 2: SNode <-- create a AStarNode for startID

SNode. hStar <-- computeHstar (SNode) // maybe perform in the constructor

SNode. gStar <-- computeGstar (SNode) // maybe perform in the constructor

SNode. fStar <-- computeFstar (SNode) // maybe perform in the constructor

step 3: OpenInsert (SNode) // insert SNode into OpenList w.r.t. SNode's Fstar value

printOpen (outFile\_2)

step 4: currentNode <-- OpenRemove () // remove the head of OpenList

step 5: matrixIndex <-- currentNode's ID

step 6: copyChildList (matrixIndex) // copy costMatrix[matrixIndex] to childAry

step 7: childIndex <-- 0

step 8: if childAry[childIndex] > 0 // means there is an edge from matrixIndex to childIndex

{

childNode <-- create a AStarNode where childNode's ID is childIndex

and cost is childAry[childIndex]

childNode. hStar <-- computeHstar (childNode) // maybe perform in the constructor

childNode. gStar <-- computeGstar (childNode) // maybe perform in the constructor

childNode. fStar <-- computeFstar (childNode) // maybe perform in the constructor

if (IsInCloseList (childNode, oldNode) is false) // means it has not been processed

{

childNode's parent <-- currentNode

numEdgeFromStart ++

OpenInsert (OpenList, childNode)

printOpen (outFile\_2)

}

else { // although childNode was processed, but its fStar is better, then

if (childNode's fStar < oldNode's fStar)

{

removeClose (oldNode)

childNode's parent <-- currentNode

OpenInsert (OpenList, childNode)

printOpen (outFile\_2)

}

}

step 9: childIndex ++

step 10: repeat step 8 to step 9 while there are more children

// i.e., childIndex is less than numNodes

step 11: ClosePush(currentNode) // put currentNode to the top of CloseList

printClose(outFile\_2)

step 12: repeat step 4 to step 11 while \*checkPath (currentNode) is false\*

step 13: tracePath (currentNode)

print all that are required to Output-1

step 14: close all files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

computeHstar (AstarNode\* node, whichHFunction )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Use case whichHFunction

Case 1: call h1 function

Case 2: call h2 function

Case 3: call h3 function

Default: print error