CS3310 – Spring 2015
Kaminski

Asgn 6 – Pure **Michigan Tour**
using Dijkstra's MinCost Path Algor.
using Ext Adj Triangular Matrix

OVERVIEW

UserApp finds the shortest distance route between 2 cities using Dijkstra's minimum cost path algorithm. Batch processing is used to facilitate testing: input transaction file (i.e., start/destination city pairs) and output log file. Map data file contains road distances between pairs of neighboring cities (using major roads only). CityNames are loaded into memory for faster searching during processing.

Setup utility converts a raw map data ASCII text file into a more efficient-to-use structure – i.e., an external adjacency matrix (a binary file) for use in UserApp. A triangular matrix is used to save space since there are no 1-way roads. CityNames are stored in a separate file.

PrettyPrint utility nicely displays MapData.bin & CityNames files to Log file to help developer.

PROJECT STRUCTURE:

- Setup, PrettyPrint & UserApp - 3 separate programs - each in a physically separate file.
- [No TestDriver program needed since demo only needs to run 3 programs].
- Setup & PrettyPrint - just short utility programs – so OOP optional.
- UserApp MUST use OOP paradigm - uses MapData & Route classes – each in separate file.
- UserApp does transaction file handling itself & opens/closes Log file (plus doing some writing to it itself, but also passing it to some of Route's methods for writing, as needed).

5 DATA FILES

1 – RawMapData.txt

INPUT to Setup

[Ignore blank lines & comment-lines (beginning with %) included for human readers.]

Two types of DATA lines:

1st) a single line with a list of all UP cities – e.g.,

```
up([ crystalFalls, copperHarbor, . . ., stIgnace ]).
```

2nd) one line per road – e.g.,

```
distance(kalamazoo, grandRapids, 51).
```

that is, cityA, cityB, distanceFromAtoB (an EDGE, not a PATH)

NOTES:

- *2-word cities like Grand Rapids are listed as one word, like grandRapids*
- *UNDIRECTED graph, so a road listed only ONCE, though it reflects AtoB and BtoA.*

2 – MapData.bin

OUTPUT from Setup
INPUT to MapData CLASS (used by UserApp)

A RANDOM ACCESS FILE
A BINARY file (no field-separators, no <CR><LF>'s)
TWO types of data records:

1st) a single headerRecord containing n & upN (2 shorts)
2nd) the upper triangular matrix (no diag) for distances (((N X N) / 2) – N) shorts

3 – CityNames.txt

OUTPUT from Setup
INPUT to MapData CLASS (used by UserApp)

cityNames – strings or charArrays, ASCII or Unicode (your choice) – IN THIS ORDER

1. upCityNames (upN of these) IN THE SAME ORDER as in RawMapData's UP list
2. theBridge
3. lpCityNames (n – upN – 1 of these)
   IN THE SAME ORDER as they appear in RawMapData, whether as cityA or cityB

4 – CityPairsTestPlan.txt (= "Trans Data")

INPUT to UserApp

[Ignore blank lines & comment-lines (beginning with %) included for human readers.]

Data lines' format: startCityName, a space, destinationCityName

NOTES:

- *2-word cities like Grand Rapids are listed as one word, like grandRapids*
- *Includes some invalid city names which must be handled appropriately, although NOT by using hard-coded special cases.*

5 – Log.txt

OUTPUT from SetupUtility & PrettyPrint
& from Route CLASS (used by UserApp)

NOTE:

- *data values below are NOT CORRECT – it just shows the required format/ wording/ spacing/. . .* **USE THIS EXACT FORMAT, including spacing ! ! !**
- *for the PrettyPrint matrix:*
  - ○ ***ETC**'s parts must be filled in with actual data*
  - ○ *, , , parts are actual commas instead of 000's to make it easier-to-check*
  - ○ *bottom triangle displays 3 spaces per value to align this into a square matrix – but the actual data file ONLY CONTAINS THE UPPER TRIANGULAR MATRIX*
  - ○ ***XXX**'s are shown in the diagonal - but they are NOT ACTUALLY IN THE FILE*

PrettyPrint produces the following in the Log file

```
MapData.bin & CityNames.txt FILES – N: 50, UP–N: 15

      00  01  02  03  04  05  06  07  08  09  10  11  12  13  14 ETC  49
   -------------------------------------------------------------------
00 | XXX 005 034 114 ,,, ,,, ,,, ,,, 004 ,,, ,,, ,,, ,,, ,,, 010 ETC ,,,
01 |     XXX ,,, ,,, ,,, 057 ,,, ,,, ,,, 023 ,,, ,,, ,,, 098 ,,, ETC 056
02 |         XXX ,,, ,,, ,,, ,,, ,,, ,,, ,,, ,,, ,,, ,,, ,,, 057 ETC 032
. . .
49 |                                                             ETC XXX
----------------UP CITIES
00) crystalFalls
01) copperHarbor
. . .
14) stIgnace,
----------------THE BRIDGE
15) theBridge
```

```
----------------LP CITIES
16) michiganCity
17) southBend
. . .
48) claire
49) grayling
```

Route class methods (& UserApp's methods) produces the following in the Log file

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
START CITY :  paris
ERROR:  not in Map Data
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
START CITY :  kalamazoo  (13)  LP
DESTINATION:  london
ERROR:  not in Map Data
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
START CITY :  kalamazoo  (13)  LP
DESTINATION:  lansing  (11)  LP

TRACE OF TARGETS:  kalamazoo battleCreek southHaven grandRapids jackson
bentonharbor lansing (7 cities)

TOTAL DISTANCE:  82 miles
SHORTEST ROUTE:  kalamazoo > battleCreek > lansing
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
START CITY :  kalamazoo  (13)  LP
DESTINATION:  copperHarbor  (24)  UP
*** different peninsulas, so 2 partial routes shown ***

TRACE OF TARGETS:  kalamazoo battleCreek grandRapids . . . theBridge (12
cities)

TOTAL DISTANCE:  320 miles
SHORTEST ROUTE:  kalamazoo > battleCreek > . . . > theBridge

TRACE OF TARGETS:  theBridge saultSteMarie . . . copperHarbor (6 cities)

TOTAL DISTANCE:  310 miles
SHORTEST ROUTE:  theBridge > . . . > copperHarbor
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

NOTES:

- *the `...` parts shown in the trace & the route must be actual list of city names*
- *ROUTE must display from START-to-DESTINATION (even though it would be easier to display DESTINATION-to-START)*
- *the wrap-around seen above (for shortest route and trace of targets) happens during printing of Log file in NotePad (or …). Program just writes a **single long line** to file..*

NOTES regarding the TRACE OF TARGETS above:

- *include START city in trace, even though it's NOT actually a target inside the big loop, since it's actually is "selected" first as part of the initialization process before the big loop starts*
- *cities MUST appear **IN THE ORDER IN WHICH THEY WERE SELECTED. D**o **NOT** just write them out based on the `included` array (which would result in city-number order rather than selection-order)*

###################### **5 REQUIRED CODE MODULES** #######################

Setup program

- converts `RawMapData.txt` into `MapData.bin` and `CityNames.txt`
- see descriptions of the 3 data files above
- NOTE: EVERYONE WILL get the SAME EXACT cityNumber for a particular cityName because everyone MUST follow the FOLLOWING way of building the cityNameList
  1. 1st all the UP cities (in the order found in the up-list in RawMapData file)
  2. then theBridge
  3. then the non-UP cities are added IN THE ORDER IN WHICH THEY'RE ENCOUNTERED (whether cityA or cityB) in RawMapData's `distance` lines
- SUGGESTION: This is just a utility, run just once. So you might choose to:
  1. construct an INTERNAL 2Dimensional SQUARE array for the distance values
  2. then after it's completely filled in with the RawMapData, write out JUST the UPPER TRIANGLULAR MATRIX to the MapData.bin file (after writing n & upN) [NOTE: This does NOT include the diagonal].
- The data is an UNDIRECTED graph, so the RawMapData only includes cityA-to-cityB distances, and assumes that cityB-to-cityA is the same distance.
- A COUPLE OPTIONS FOR FILLING the square matrix
  1. Each A-to-B distance you read in gets put in BOTH [A][B] as well as [B][A], though you'll only end up writing the UPPER TRIANGULAR MATRIX to the binary file
  2. Only put data into the UPPER TRIANGULAR AREA of the square matrix, since that's what you'll end up writing to the binary file. That area only includes ROW < COL (vs. the LOWER triangular area where ROW > COL). So for each A-to-B distance you read in, once you determine the 2 cities' numbers, make sure cityA's Num < cityB's Num (swapping the two, if needed) – then insert it into [A][B] only.
- Use 3 TYPES OF DATA VALUES in the square matrix (as per class discussion) rather than 2 (as books/internet often show):
  - i. maxShort's for the non-edges (rather than 0's)
  - ii. 0's for the diagonal [THOUGH THEY'RE NEVER WRITTEN TO THE FILE]
  - iii. actual edge values
- For creating the CityName FILE, first build an INTERNAL cityNameList (an array or arrayList) since you'll have to repeatedly search it to see if the name is already in the list. Then at the end, dump the list to the file.
- Use a 1-pass loading (i.e., 1 pass through RawMapData file) to load data into the 2D array and cityNameList, with counters for n and upN using the STREAM PROCESSING Algorithm - i.e., loop til EOF {read 1 line, deal with it}

+ +

PrettyPrint program

- Nicely displays `MapData.bin` and `CityNames.txt` to Log file
- see description of Log file above

+ +

UserApp program - (the controller & transaction file handler)

```
-   declare mapData & route objects
-   open transaction file (CityPairsTestPlan.txt)  & Log file
-   loop to process transaction file, one pair at a time
            get a city pair,
            find both of city numbers (from mapData's getCityNum),
            find which peninsula the cities are in (from mapData's getPeninsula)
            if either city is theBridge,  fix its peninsula to MATCH the other city's peninsula,
                since a MATCHED PAIR is better than a NON-MATCHED PAIR
            write out intro info to Log file
            if it's a UP/UP or LP/LP city-pair
                        find shortest route (using route's findShortestRoute),
            else it's a UP/LP or a LP/UP city-pair, so for a more efficient search,
                    1)  find shortest route from startCity to theBridge
                    2)  find shortest route from theBridge to destinationCity
-   finish up with mapData object (i.e., the file needs closing)
-   close Log file
```

NOTE: Route class's methods do the actual writing of Trace of Targets, Total Distance & Shortest Route directly to Log file, so that file has to be passed in.

+ +

MapData class

Handles everything to do with MapData.bin file, CityNames.txt file and mapData functionality in general, including:

- open files
- load header data into memory
- load city names into MEMORY into cityNameList (an array or arraylist)
  [OR 2 separate lists, upNameList and lpNameList]
- DO NOT LOAD THE MATRIX INTO MEMORY
- close files

Some public services:

getCityNum	*(based on a cityName)*
getPeninsula	*(based on a cityName)*
getCityName	*(based on a cityNum)*
getRoadDistance	*(based on cityNumA and cityNumB)*

NOTE:

- *getRoadDistance(row, col) uses RANDOM ACCCESS to read the distance (a short) from the BINARY FILE.*
- *To calculate byteOffset, allow for*
  - *a. the sizeOfHeaderRec (2 shorts) WHICH ISN'T PART OF FORMULA BELOW),*
  - *b. followed by the UPPER TRIANGULAR ARRAY. Here's a formula for dealing with this since it's mapped to a linear structure (the file, just a stream of shorts):*

$$(n*(n - 1)/2) - ((n - row)*((n - row)-1)/2) - row - 1 + col$$

from: http://stackoverflow.com/questions/27086195/linear-index-upper-triangular-matrix

NOTES on cityNameList:

- *EVERYONE will have order of the cities (as description of CityNames file above)*
- *The list is NEVER SORTED*
- *The list starts at 0, not 1*

+ +

Route class

NOTE: This class needs access to Log file

NOTE: This class needs access to mapData methods like getRoadDistance and getCityName.

Storage: 3 working array (all of size N, not MAX_N, since you now already know N):
distance, included, predecessor

Public services:

findShortestRoute *(= the local controller which handles a Start/Destination pair of city NUMBERS. **NOTE:** it is NOT this class's responsibility to handle searching for city **names – use** MapData's getCityName)*

Local (private) methods used by findShortestRoute:

initialize3Arrays

searchForPath *(the "search" part of Dijkstra's Algorithm)*
*(PLUS it prints the TRACE OF TARGETS **as it's selecting them**)*

reportAnswers *(the TotalDistance and the ShortestRoute)*
(from Start to Destination, not Destination to Start)

NOTES:

- The Start/Destination pair for this module are not always the cityPair FILE's Start and Destination cities. See note above for LP/UP and UP/LP mismatched pairs where theBridge is used as an intermediate city to make the search more efficient. In that case, 2 partial routes are found and reported. But that is all controlled from UserApp – findShortestRoute has no idea this is happening – and that UserApp is calling findShortestRoute TWICE.
- The TraceOfTargets and ShortestRoute specify cityNAME not cityNUMBER. So the methods above have to have access to MapData's getCityName service.

############################### **MY ALGORITHM** ###############################

You MUST USE THE ALGORITHM I SHOW IN CLASS for Dijkstra's Algorithm. Don't just take code/algorithm from a data structures book or the internet, (even though it would produce the same results) – BIG POINT-LOSS IF YOU DO.

My algorithm assumes that the graph (conceptually) contains THREE different possible values (not 2) in the graph, which differs from what's in the eReadings:

1) actual edge weights if there's an edge
2) 0 on the diagonal for a node to itself
3) "INFINITY" meaning there's NO edge (some books use 0's here too for "NO edge")