CS3310 - Spring 2016 — Asgn 6 – **Pure Michigan**
Dr. Kaminski — Dijkstra's Minimum Cost Path (Graph)

################################# **OVERVIEW** #################################

The program finds the shortest route between pairs of cities using Dijkstra's minimum cost path algorithm. Batch processing is used to facilitate testing, with the user's list of city pairs stored in a transaction file and results going to a log file. The raw map data is stored in a file – it contains road distances between neighboring major cities in Michigan (and a few out-of-state cities just across the borders) using major roads only. Before any route-finding is done, the external data is loaded into memory, storing the graph data edges in an adjacency matrix. Route-finding is done using the internal graph, not the external raw data file.

NOTE: NO Setup or PrettyPrint – only UserApp, the controller, which uses
MapData & ShortestRoute OOP classes.

############################ **THE 3 DATA FILES** ############################

<u>`MichiganMap.txt`</u>
Ignore blank lines and comment-lines (i.e., those beginning with %).
Two kinds of DATA lines:

1st) ONE line with list of cities in the Upper Peninsula ("the UP") in the form:
```
up([ crystalFalls, copperHarbor, . . ., stIgnace ]).
```
There are 3 categories of cities:

- in the UP - that is, those in the list
- theBridge
- in the LP (Lower Penninsula) – i.e., if it's NOT in the UP or theBridge.

2nd) ONE line PER ROAD (i.e., graph edge) in the form:
```
road(kalamazoo, grandRapids, 51).
```
that is, (cityA, cityB,roadDistanceBetweenAandB

<u>*NOTES:*</u>
- *all 2-word cities like Grand Rapids will be listed as one word, like grandRapids*
- *data represents an UNDIRECTED graph, so a particular road is LISTED ONLY ONCE in the data file, though it must be stored in the internal GRAPH in BOTH spots [A][B] and [B][A].*

<u>`CityPairsTestPlan.txt`</u> (the transaction data)
Ignore blank lines and comment-lines (i.e., those beginning with %).
FORMAT: 2 city names per line (with a space between), first is **<u>start</u>**, second is **<u>destination</u>**.
<u>*NOTES:*</u>
- *all 2-word cities like Grand Rapids will be listed as one word, like grandRapids*
- *there will be some invalid city names which must be handled appropriately.*

<u>`Log.txt`</u> **USE THIS EXACT FORMAT, including spacing ! ! !**

```
%    %    %    %    %    %    %    %    %    %    %    %    %    %    %

START:  paris
>>> ERROR >>> city not in Michigan Map

%    %    %    %    %    %    %    %    %    %    %    %    %    %    %

START:  kalamazoo (13 - LP)
DESTINATION:  london
>>> ERROR >>> city not in Michigan Map

%    %    %    %    %    %    %    %    %    %    %    %    %    %    %

START:  kalamazoo (13 - LP)
DESTINATION:  lansing (11 - LP)

TRACE OF TARGETS:  kalamazoo battleCreek southHaven grandRapids jackson
bentonharbor lansing

TOTAL DISTANCE:  82 miles
SHORTEST ROUTE:  kalamazoo > battleCreek > lansing

%    %    %    %    %    %    %    %    %    %    %    %    %    %    %

START:  kalamazoo (13 - LP)
DESTINATION:  copperHarbor (24 - UP)

>>> NOTE >>> 2 different peninsulas, so 2 partial routes

TRACE OF TARGETS:  kalamazoo battleCreek grandRapids . . . theBridge

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

TRACE OF TARGETS:  theBridge saultSteMarie . . . copperHarbor

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

%    %    %    %    %    %    %    %    %    %    %    %    %    %    %

START:  kalamazoo (13 - LP)
DESTINATION:  theBridge (43 - BRIDGE)

TRACE OF TARGETS:  kalamazoo battleCreek grandRapids . . . theBridge

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

%    %    %    %    %    %    %    %    %    %    %    %    %    %    %

START:  theBridge (43 - BRIDGE)
DESTINATION:  theBridge (43 - BRIDGE)
```

```
TRACE OF TARGETS:  theBridge

TOTAL DISTANCE:  0 miles
SHORTEST ROUTE:  theBridge

%    %    %    %    %    %    %    %    %    %    %    %    %    %    %
```
NOTES:

- *data values above are NOT necessarily CORRECT – it's just to show the required format/ wording/spacing/…*
- *the* `...` *parts shown in SHORTEST ROUTE and TRACE OF TARGETS must be the full list of city names that are applicable*
- *SHORTEST ROUTE must display from START-to-DESTINATION (even though it'd be easier to display DESTINATION-to-START)*
- *IMPORTANT: if start & destination cities are in DIFFERENT peninsulas, then 2 different route-searches are done (i.e., doing the 3 steps of Dijkstra's Algorithm each time)*
- *theBridge is considered to be in BOTH peninsulas – so if start OR destination is theBridge, don't do 2 searches - just do what you'd do for a UP→LP or for an LP→ UP pair*
- *the wrap-around seen above for SHORTEST ROUTE and TRACE OF TARGETS happens during printing of Log.txt file in NotePad/WordPad - the program just keeps writing a SINGLE LINE to the file*
- *include START city in TRACE OF TARGETS, even though it's not actually selected as a target inside the loop, (since it's "selected" first, as part of initialization, before big loop starts*
- *TRACE OF TARGETS, cities must appear **IN THE ORDER THEY WERE SELECTED** – do **NOT** just write them out based on the Included array (which would be in city-number order)*

########################### **REQUIRED MODULES** ###########################

NOTES: You MUST

- *have physically separate files for UserApp program, MapData class, RouteFinder class*
- *MODULARLIZE – i.e., have separate modules (not just nameless contiguous lines of code) to do the functions described below*
- *use the naming described in these specs (for program, classes, methods, variables) since the specs serve as external documentation (to reduce amount of internal documentation)*
-

You MAY have other additional methods and classes besides what's described here.

UserApp program (the overall controller & transaction handler)

- create mapData & shortestRoute objects
- call MapData's loadMapData
- open CityParisTestPlan file & Log file
- process transaction file (the CityPairsTestPlan), one pair at a time:
  - get a city pair
  - find both their city numbers (use MapData's getCityNumber)
  - write out intro info to Log file
  - if UP/UP or LP/LP city-pair

    (NOTE: theBridge considered to be in BOTH LP & UP)

    find shortest route (using ShortestRoute's findRoute)
  - else it's a UP/LP or a LP/UP city-pair, so for a more efficient search,
    - find shortest route from startCity to theBridge
    - find shortest route from theBridge to destinationCity
- close CityParisTestPlan file & Log file

NOTE: ShortestRoute's methods do actual writing of Trace of Targets, Total Distance & Shortest Route directly to Log file

#   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #

MapData class

storage: the map (the graph) including:

n	*(number of nodes)*
roadDistance	*(adjacency matrix (2D array) for the edges)*
cityNameList	*(an array, not a list, to allow for random & serial access)*
upCityList	*(a list or an array, only needs linear search access)*

public services:

loadMapData	*(from file into the map in memory) [does its own open/close of file]*
getCityNumber	*(based on cityName)*
getCityPeninsula	*(based on cityName – theBridge or UP (check the list) or LP)*
getCityName	*(based on cityNumber)*
getRoadDistance	*(based on 2 cityNumbers)*

private service:

storeCityName	*(and return cityNumber)*

NOTES:

- *May have additional methods*
- *Program won't know what N is til run time (after it finishes reading MichiganMap file). So use MAX_N (defined constant = 200 for now) for declaring static array sizes for cityNameList array and roadDistance 2D array.*

- *To ensure EVERYONE gets the SAME cityNumber / cityName correspondence for A6*
    - *Arrays start at [0] not [1]*
    - *List NEVER sorted. It's built based on order of input file's ROAD data lines and STAYS in THAT ORDER*
    - *Do NOT use the UP city list to start assigning cityNumbers to cities – Only use the ROAD data lines in MichiganMap file. So michiganCity is 0, NOT crystalFalls. And southBend is 1.*

- *cityNameList MUST be stored in an ARRAY (not a list, not an arrayList) so direct address can be used for getCityName (used in printing TraceOfTargets and Shortest Route)*

- *storeCityName does linear search since cities are NOT in sorted-order*
- *storeCityName will either*
    - *find name already stored in cityNameList*
    - *OR NOT find name already stored, in which case, ADD IT to cityNameList*

- *getCityNum & getCityPeninsula do linear search since cityNameList & upCityList not sorted*
- *getCityName uses direct address of cityNameList based on cityNumber*

- *To fill the internal graph:*
    - *Initialize all cells to "infinity" (i.e., int.MaxValue)*
    - *Initialize diagonal to 0's*
    - *Use road distances from data file to fill in the appropriate cells with good data*
- *It's an UNDIRECTED graph. Data file contains 1 line for 1 road, but that data must be stored in both [A][B] and [B][A] in the graph in memory*

<u>NOTES on processing the Roads file</u>
- *Process MichiganMap file using ONE PASS. Read a ROAD line for an edge; store both names in cityNameList (if they're not yet stored) and store roadDistance in graph (map).*
  Do NOT do 2 passes through data, e.g., to build cityNameList, then to fill roadDistances, OR to determine N for sizing the arrays, then to use them).

- *Process MichiganMap FILE using sequential stream processing algorithm. Do NOT read the entire file into memory just to store it, and THEN process data from memory to build the cityNameList and roadDistance matrix.*

# # # # # # # # # # # # # # # # #

<u>ShortestRoute class</u> (DijkstrasAlgorithm)
storage: 3 parallel "scratch" arrays (all of size N, not MAX_N, since program now know N):
    distance, included, predecessor
public services:
    findRoute (*= local controller which handles a Start/Destination pair of city **numbers.***
    ***NOTE:** it is NOT this class's responsibility to handle city **names.**)*
private services:
    initializeArrays
    searchForPath (*the "search" part of Dijkstra's Algorithm)*
        *(plus it prints the TRACE OF TARGETS **<u>as it's selecting them</u>**)*
    reportAnswers *(TotalDistance and ShortestRoute)*
        (from Start to Destination, not Destination to Start)

    selectTarget *(the "search" part of Dijkstra's Algorithm)*

<u>NOTES:</u>
- Start/Destination pair for findRoute are not always the transaction file's Start and Destination cities. See notes (and example in Log file description) above for LP/UP and UP/LP pairs where "theBridge" is used as an intermediate city to make the search more efficient - in which case 2 partial routes are found and reported. But this is all controlled from UserApp – RouteFinder's findRout has NO IDEA that this is happening.

- TraceOfTargets & ShortestRoute specify cityNAME not cityNUMBER.
  So methods above have to have access to MapData's getCityName.

############################## **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 contains THREE different possible values (not 2, as is sometimes used) in the graph:
1) actual edge weights if there IS an edge
2) 0 on the diagonal for a node to itself
3) INFINITY (i.e., maxInt) meaning there's NO edge
   (some examples' graph implementations use 0's here for "NO edge")

############### **SELECT TARGET - CHANGE TO ALGORITHM** ################

EFFICIENCY ISSUE (NOT DESCRIBED IN MY ALGORITHM DOC ON COURSE WEBSITE)

Because of the 2 peninsulas situation, we KNOW that any LP/LP path will ONLY include LP cities (including possibly theBridge). Similarly for UP/UP paths ONLY including UP cities. And all UP/LP (or LP/UP) requests are split into 2 single-peninsula searches.

SO, when selecting a target, NEVER choose a target city which is in the OTHER peninsula from the one that's being searched. And theBridge is always considered to be in BOTH peninsulas.