CS1027

ASSIGNMENT 1

Computer Science Fundamentals II

This assignment is due on Tuesday, February 9, 2021 by 11:55pm. See the bottom for submission details.

Learning Outcomes

To gain experience with

- Creating classes with simple methods
- Using linear arrays and multi-dimensional arrays
- Algorithm design

Introduction

It is important in Computer Science to consider memory and to use compression to reduce the amount of memory being used by a program. A matrix in linear algebra is considered *symmetric* if it is identical to its transpose. In other words, the element at (i,j) is the same as the element at (j,i) for all i and j in the bounds of the matrix. The *diagonal* (the set of elements from the top-left corner to the bottom-right corner) is the reflection line. We can represent a matrix as a 2D array in Java and consider the array to be symmetric by the same definition as that of a matrix. For the purpose of reducing memory, we can cut out a roughly half the array since the elements are already stored in the other half of the array.

If we calculate the distances between various locations, a matrix or 2D array can be used to store the distances. For a set of n cities, the array would have n rows and n columns to contain all the distances between each pair of cities. However, this would be an example of a symmetric array since the distance between A and B is the same as the distance between B and A, and thus that distance would exist twice in the array. Similarly, the diagonal of the array would contain all zeroes since the distance between a city and itself is always 0. Due to these observations, we can compress the distance array by storing just the lower-left corner (all the elements below and to the left of the diagonal) rather than the entire array. This compressed array will still contain all the information needed to represent the distance between every pair of cities, but will not contain any unnecessary or redundant data. The Figures below help illustrate this concept more clearly.

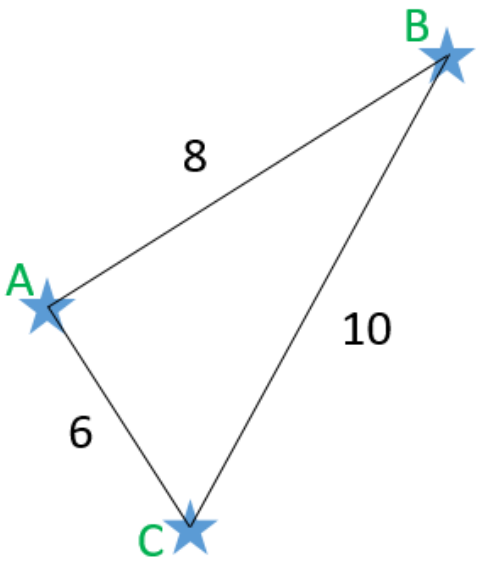

Figure 1. An example of a map of 3 cities, A, B, and C, and the distances between them.

	A	B	C
A	0	8	6
B	8	0	10
C	6	10	0

Figure 2. A matrix (2D array) representing the distances between each of the cities from the map in Figure 1 above.

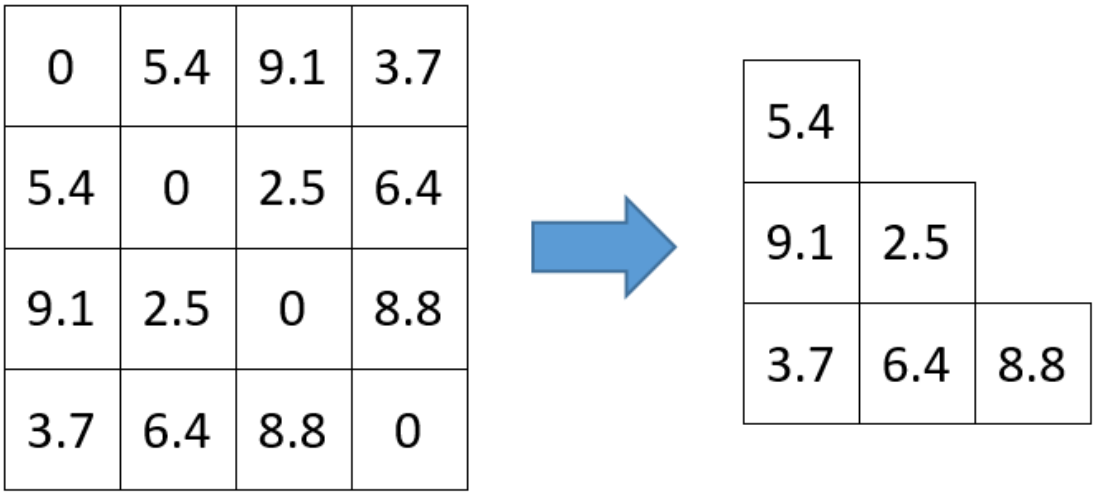

Figure 3. Another example of a matrix (2D array) containing hypothetical distances between cities or objects (left), and the CompressedArray of the same matrix (right).

Assume an "original" 2D array has n by n elements. When compressing this into a CompressedArray structure, we want to only include the elements that are below and to the left of the diagonal, as illustrated in Figure 3. To determine the number of elements needed in the CompressedArray, take the number of the elements in the original array, and subtract the number of elements on the diagonal (this is also n) to determine the number of non-diagonal elements. However, this number also include the top-right triangle which should also be cut out. Divide this number by 2 to determine the number of elements in the lower-left triangle only.

Provided files

The following is a list of files provided to you for this assignment. Please do not alter these files in any way.

- CityMarker.java – adds a city marker icon on the visual GUI
- Map.java – provides a visual GUI for the program
- MyFileReader.java – reads in text files
- TestCompressedArray.java – tests that CompressedArray is working
- TestProgram.java – tests that Program is working

Classes to implement

For this assignment, you must implement 3 Java classes: *City*, *CompressedArray*, and *Program*. Follow the guidelines for each one below.

In all of these classes, you can implement more private (helper) methods, if you want to, but you may **not** implement more public methods. You may **not** add instance variables other than the ones specified below. Penalties will be applied if you implement additional instance variables or change the variable types or modifiers than what is described here.

City.java

This class represents each city that is loaded in from a text file. It is a simple class that just contains the city's name, x and y coordinates, and a CityMarker icon.

The class must have the following *private* variables:

- name (String)
- x (int)
- y (int)
- marker (CityMarker)

The class must have the following *public* methods:

- public City(String, int, int) [constructor]
  - Takes in parameters for name, x, and y and assigns these values to the corresponding instance variables. The marker variable must also be initialized in this constructor as a new CityMarker object.
- public String getName()
  - Returns name
- public int getX()
  - Returns x
- public int getY()
  - Returns y
- public CityMarker getMarker()
  - Returns marker
- public void setName(String)
  - Sets name
- public void setX(String)
  - Sets x
- public void setY(String)
  - Sets y

- public void setMarker(CityMarker)
  - Sets marker
- public String toString()
  - Returns the city name

CompressedArray.java

This class represents the array that has been compressed from a matrix (2D array) into a linear array that excludes elements from the diagonal and any elements above or to the right of the diagonal. Only elements below or to the left of the matrix diagonal must be included in the CompressedArray.

The class must have the following *private* variables:

- origArraySize (int)
- array (double[])

The class must have the following *public* methods:

- public CompressedArray(double[][]) [constructor]
  - Takes in a 2D double array (double[][]) which represents the "original array"
  - Initialize the linear double array (instance variable array) and copy the elements from the original array into this linear array so that it contains only the lower-left triangle elements of the original array (the elements to the left and below the diagonal). The elements from this triangle must be added to the CompressedArray in order from left to right and top to bottom.
  - Hint: Read the description near the top of this document to determine the required capacity for this CompressedArray.
- public int getLength()
  - Returns the length of the new, compressed array
- public double getElement(int)
  - Returns the element in the new, compressed array stored at the given index
- public boolean equals(CompressedArray)
  - Checks equality between the two CompressedArray objects by checking if they have the same length and that all the elements are identical in the same order
- public String toString()
  - Builds a string that contains the CompressedArray and formats it in a trianglular structure to look like the lower left corner of a matrix. Each element should take up exactly 8 characters and show 2 decimal places.
  - Hint: Use String.format("%8.2f", element) for each element and remember to add a newline at the correct places

Program.java

This class, as its name suggests, will be the main heart of the program. It will be the entry point of the program, read in a file of cities and create objects for each of them, contain the array of those cities, and create a CompressedArray containing the distances between each of the cities read in from the file.

The class must have the following *private* variables:

- cityCount (int)
- cityArray (City[])
- array (CompressedArray)

The class must have the following methods:

- public Program(String, boolean) [constructor]
  - Takes in a String representing the file to load (i.e. "cities1.txt") and a boolean (showMap) that indicates whether or not the map GUI should be displayed
  - Initialize cityArray with 3 cells
  - Create an object of MyFileReader or use your own code to read in a text file, and load in the file with the given name.
  - Read in each line from the file and create a City object containing the city name, and the x and y values from the file (look at the text file to see this format). Add the city object to the cityArray and expand the capacity if needed.
  - If the boolean (showMap) is true, then create a Map object and call addCity() on the Map object for each city in the cityArray. This will add the marker icons to the map for each city.
- public City[] getCityList()
  - Returns cityArray
- private void expandCapacity()
  - Expands the capacity of cityArray by adding 3 additional slots to the array
- public double distBetweenCities(City,City)
  - Calculates the Euclidean distance between the two given Cities
- public void compareDistances()
  - Create a 2D double array (i.e. double[][]) with a size of N by N, where N is the number of cities in cityArray. Loop through every combination of pairs of cities and call distBetweenCities() for each pair. Save this result into the double[][] array in the appropriate cell.
- public CompressedArray getArray()
  - Returns array

ASSIGNMENT 1

Marking notes

Marking categories

- Functional specifications
  - Does the program behave according to specifications?
  - Does it produce the correct output and pass all tests?
  - Are the class implemented properly?
  - Are you using appropriate data structures?
- Non-functional specifications
  - Are there Javadocs comments and other comments throughout the code?
  - Are the variables and methods given appropriate, meaningful names?
  - Is the code clean and readable with proper indenting and white-space?
  - Is the code consistent regarding formatting and naming conventions?
- Penalties
  - Lateness: 10% per day
  - Submission error (i.e. missing files, too many files, etc.): 5%
  - "package" line at the top of a file: 5%

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through cheating-detection software.

Submission (due Tuesday, February 9, 2021 at 11:55pm ET)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.

- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

Files to submit

- City.java
- CompressedArray.java
- Program.java

Grading Criteria

- Total Marks: [20]
- Functional Specifications:
  - [1] City.java
  - [2.5] CompressedArray.java
  - [2.5] Program.java
  - [10] Passing Tests
- Non-Functional Specifications:
  - [1.5] Meaningful variable names, private instance variables
  - [0.5] Code readability and indentation
  - [2] Code comments

Canada map obtained from: https://freevectormaps.com/canada/CA-EPS-01-0002?ref=atr.