Programming Assignment #2

Some Preliminaries

Programming assignments are to be done individually. Do not make your code publicly available (such as a Github repo) as this enables others to cheat and you will be held responsible. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

The Goals

This programming assignment is due **Friday, October 27th at 11:59 PM.** If you are unable to complete the assignment by this time, you may submit the assignment late until Sunday, October 29th at 11:59 PM for a 20-point penalty.

The goals of this lab are:

- Recognize and deploy algorithms you have learned in class in new contexts.
- Ensure that you understand certain aspects of graphs and greedy algorithms.
- Recognize and reason about social implications of employing algorithms in a diagnostic application.

The Problem: Detecting Features in Brain Scans

Functional Magnetic Resonance Imaging (fMRI) can be used to detect or measure changes in the blood flow within the brain and can be used to help detect and monitor a variety of diseases affecting the brain (ranging from epilepsy to Alzheimer's to ADHD). One way in which this data is analyzed (not kidding!) is through the application of graphs and minimum spanning trees.

Researchers can use data from prior brain imaging studies, stored in *atlases* (more about these later) to identify functional regions in imaging data of a new individual's brain. These functional regions are connected by *white matter tracts*, which are effectively bundles of neurons that carry signals from one region of the brain to another. We can create brain networks (graphs) out of this information, where the regions are the vertices in the graph and the white matter tracts are the edges. These edges have weights, which can signify a variety of things, but for our purposes, we'll assume that the weights indicate the magnitudes of interactions between pairs of brain regions.

If you're interested in this application domain (and who wouldn't be?!), you can read more about how these networks are constructed in this paper:

> Rubinov, Mikail, and Olaf Sporns. "Complex network measures of brain connectivity: uses and interpretations." Neuroimage 52.3 (2010): 1059-1069. `https://www.sciencedirect.com/science/article/pii/S105381190901074X`

This paper is really accessible to someone who is familiar with graphs and graph algorithms (that's you!). In particular, you might find Figures 1 and 2 in this paper to be pretty helpful.

In this assignment, we will provide you with a graph representation for a **connected, undirected graph**. In this graph, vertices represent brain regions, edges represent functional connections between the regions, and the weight on each edge represents the strength of interactions between those regions. These weights range from 0 to 1000. Two regions that do not interact are not connected by an edge. The graph will be provided as a list of regions. Each region has its adjacency list defined by the ArrayList `neighbors` and `weights`. Each index in `neighbors` represents a neighboring region and the weight of that edge corresponds to the same index in `weights`.

One common mechanism for analyzing brain networks like the one you will be given is to construct a "maximum" spanning tree over the network and then analyze properties of that tree. The idea is that this is the tree connecting all the regions with maximum total weight (i.e., strongest connections). If you want to read about some of the details of this, check out this article:

> Blomsma, Nicky, et al. "Minimum spanning tree analysis of brain networks: A systematic review of network size effects, sensitivity for neuropsychiatric pathology, and disorder specificity." Network Neuroscience 6.2 (2022): 301-319. `https://direct.mit.edu/netn/article/6/2/301/109995/Minimum-spanning-tree-analysis-of-brain-networks-A`

The first thing you will do in this assignment is to use Prim's algorithm to construct a maximum spanning tree of a provided graph. Think about how you can use Prim's to *maximize* the total weight of the tree rather than minimize it. The input to your algorithm will be a graph of functionally connected brain regions in the adjacency list form described above, and the output of your algorithm should be the total length of the MST. Additionally, you should modify the `MST_neighbors` and `MST_weights ArrayList` to represent the maximum spanning tree.

(Note: a given graph may have more than one MST if there are ties in edge weights. For the purposes of this assignment, we will consider all MSTs of the graph to be equivalent, given that they have the same total cost.)

Next, if you clicked into the above reference and looked at Table 1, you will have discovered that there are a few metrics that brain researchers use when analyzing MSTs of brain networks. We're going to implement the first one: *diameter*. The diameter of an MST is the length of the longest path in it (***in number of edges, not in total cost***). So the second thing you will do in this assignment is to provide an algorithm that efficiently computes the diameter of the brain network, given the MST as input.

We will provide you with the following classes:

- `Region`
  Keeps track of `int minDist` (the minimum total distance needed to get to this `Region` from the starting `Region`) and `int name` (the numerical id of the region, as written on the map). Also contains `ArrayList<Region> neighbors` (adjacent `Region`s) and `ArrayList<Integer> weights` (strengths of interactions with adjacent `Region`s).

- `Program2`
  The class that you will be implementing your algorithms in.
- `Problem`
  The class that will be passed to each function in `Program2`. It contains the `startRegion` for `findFurthestRoute()` and the parsed `Regions` list.
- `Driver`
  Reads file input and populates `ArrayList regions`. You can modify `testRun()` for testing purposes, but we will use our own `Driver` to test your code.
- `Heap`
  Provides a generic min-heap implementation that can be used with Regions and heapifys based on a region's `minDist` value. Please review the *Heap.java* file to see what public functions are provided.
- `HeapMember`
  This abstract class contains the necessary properties per node in a heap and is the class that the Heap class expects to be working on. Region.java extends this class, which enables us to use the Region class in the heap.

** Part 1: Finding the Maximum Spanning Tree of the Graph ** [30 points]

Implement an algorithm to find the length of the maximum spanning tree connecting every `Region`, given a list of `Regions` and edges with weights indicating the strength of interactions between them. You will be heavily penalized for a non-polynomial time (in terms of the total number of `Regions`) algorithm. The specific inputs provided and outputs expected are provided below. **Note that the terminology below is based on *minimum* spanning trees rather than maximum spanning trees. However, you will have to use the given code in a way that maximizes the total weigth, rather than minimizes it.**

Input(s):

- `problem`
  Contains the `ArrayList regions`, which is a list of regions that describe the current problem's graph.

Output:

- A `minLen` value where `minLen` represents the sum of the lengths of the edges connecting the minimum spanning tree. It is NOT necessary to explicitly return the MST as long as your algorithm returns the correct `minLen`. However, a later part requires the MST of a graph. It will therefore be beneficial to update each region's `MST_neighbors` and `MST_weights` to track the MST of the graph.

Method Signature:

- `int findMinimumLength(Problem problem);`

**** Part 2: Finding the Diameter of the MST **** [30 points]
Implement an algorithm to find the diameter of a given MST. The specific inputs provided and outputs expected are provided below. When graded, the MST will be provided in the `MST_neighbors` and `MST_weights ArrayList`. However, in order for you to test and debug your code, you will need to call your `findMinimumLength(problem)` in order to find the MST. This is why your `findMinimumLength()` function should keep track of the MST.

Input(s):

- `problem`
  Contains the ArrayList `regions`, which is a list of regions that describe the current problem's MST. Each region has an ArrayList `mst_neighbors`, which list which regions are the current regions' neighbors in the MST. Problem also contains an ArrayList `mst_weights`, which defines the weight of the corresponding edge in `mst_neighbors`.

Output:

- A `distance` value where `distance` represents the diameter distance. It is NOT necessary to explicitly return the actual diameter path as long as your algorithm returns the correct distance. Remember, distance is measured in **number of edges**, not the sum of the weights on the path.

Method Signature:

- `int calculateDiameter(Problem problem);`

**** Part 3: Write a report **** [40 points]
Write a short report that includes the following information:

(a) Give the pseudocode for your implementation of an algorithm to find the diameter of a MST. Further, give the runtime of your algorithm and justify it. Make sure to give enough detail in your pseudocode so that the runtime can be accurately determined, and when justifying the runtime, go into sufficient depth to discuss how your actual implementation maintains the desired runtime.

(b) Earlier in the assignment, we mentioned that much of the study of fMRI data is based on *atlases* that collect statistical information about structure and function of human brains. Atlases are expensive to create, and the data they contain depend on particular characteristics of the individuals whose brains are imaged. Consider the following quote:

> *Brain structure in old age and early life is different to brain structure in younger and middle-aged adults (Gur et al., 1991; Courchesne et al., 2000; Good et al., 2001; Sowell et al., 2003). [...] Because of these differences in brain structure, the use of an atlas based on only younger subjects and a limited range of sequences can create a bias in life course population studies [...] Therefore, population brain atlases must include information on age, sex, ethnicity, relevant medical history, and cognitive testing to have broad uses and relevance. Further, brain atlases should be derived*

> *using statistical methods that effectively characterize the wide and irregular variance in brain structure across the life course (Dickie et al., 2013).*[1]

Thus factors like age, sex, ethnicity, etc. may impact the *structure* of the brain. Comment on the potential reasons why a lack of diverse representation of demographics in the brain atlases would exist. Considering the uses of fMRI scans we described above, comment on the potential pitfalls of *not* having a diverse representation in the brain atlas used to influence the creation of the brain network that is the input to your algorithms? What steps could you take to avoid these pitfalls?

Input File Format

The first line of file is the total number of Regions and the total number of edges between the Regions. The first number on each even line is the id of a region (natural number), and every number that follows it is a region that is connected to it by an edge. The first number on each odd line is the id of a region, and every number that follows it is the weight on the edge between it and the connected region in the line above.

For example, if the input file is as follows:
3 2
0 1
0 9
1 0 2
1 9 8
2 1
2 8
Then there are 3 regions, 2 edges.
Region 0 has an edge to Region 1 with weight 9.
Region 1 has an edge to Region 0 with weight 9 and Region 2 with weight 8.
Region 2 has a edge to Region 1 with weight 8.

We will parse the input files for you, so you don't need to worry too much about the input file format except when trying to make your own test cases.

We will use different inputs for testing, but test cases will have valid inputs so you do not need to handle invalid inputs throughout your whole program.

[1] Dickie, David Alexander, et al. "Whole brain magnetic resonance image atlases: a systematic review of existing atlases and caveats for use in population imaging." Frontiers in neuroinformatics 11 (2017): 1." `https://www.frontiersin.org/articles/10.3389/fninf.2017.00001/full`

Getting Started

(a) Download the starter material from canvas.

(b) Do an initial compile of the starter code in your favorite Java IDE or on the ECE LRC Linux machines (recommended), see note below on how to compile.

(c) Test your code using the inputs given and the Driver class.

(d) Submit your code to Gradescope to validate your code. You have unlimited submissions before the due date. Your latest submission will be the only submission we look at.

NOTE: Do not modify the signatures or remove existing methods of `Program2.java`, `Region.java`, `Heap.java`, or `HeapMember.java`. Do not add package statements. Do not add extra imports.

What To Submit

You should submit only `Program2.java` to Gradescope. Please do not submit any other Java file. Failure to follow these instructions will result in a penalty of up to 10 points.

Your PDF report should be legibly scanned and submitted to Gradescope. Both your code and PDF report must be submitted by 11:59 PM on Friday, October 27, 2023. If you are unable to complete the assignment by this time, you may submit the assignment late until Sunday, October 29, 2022 at 11:59 PM for a 20 point penalty.