**CSCI 4041, Fall 2018, Programming Assignment 9**

Due Tuesday, 11/6/18, 10:30 AM (submission link on Canvas)

This is not a collaborative assignment; you must design, implement and test the solution(s) on your own. You may not consult or discuss the solution with anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class. Obtaining or sharing solutions to any programming assignment for this class is considered academic misconduct. If you are not sure what this means, consult the class syllabus or discuss it with the course instructor.

(Note: the first page is entirely fluff information. If you don’t like ridiculous stories used to justify doing the problem, you’re free to skip to page 2 where we start talking about actual implementation)

For eons, the Chromatic Crusaders have anonymously defended the Earth from the powers of evil and boring colors. They use the power of Friendship to fend off mystical threats that most of the world cannot even begin to fathom. A Crusader’s greatest weapon is their Chromatic Call, a technique which draws upon the Crusaders’ unbreakable bond to summon the other Crusaders to their side. You are the Red Crusader, and you have discovered that at long last, the Gray Legion has returned. Surely now is the time to unleash the Chromatic Call.

The problem is, it’s been a LONG time since evil has risen, and you haven’t really bothered to keep in touch with most of the other Crusaders. It’s to the point where you wouldn’t even consider all of them your friends: some are merely acquaintances. The Crusaders have a shared pool of Friendship Power that they use to fuel their powers, and Chromatic Call draws upon that pool. The amount of Friendship Power required to summon another Crusader to your location is based on the distance between you: where distance here is the figurative distance in your friendship and not the literal distance in space. You should not attempt to summon another Crusader who is no longer your friend: the amount of Friendship Power required to summon an acquaintance is infinite, so you’ll just fall unconscious if you try.

However, you have a backup plan. While you won’t be able to summon all of the Chromatic Crusaders to battle, the ones that you are able to call upon might still be friends with some of the others, so they can in turn use their own Chromatic Call to summon more of your allies, and you can hopefully continue this process until all of the Crusaders are present. In fact, since the Friendship Power you use to fight is a shared resource, it makes sense to find a sequence of Chromatic Calls that uses as little Power as possible, so that you have more in reserve to battle the Gray Legion. Thankfully, your Crusader suit happens to have a chart display that measures the friendship distance between every pair of Chromatic Crusaders, so you just need to compute the Minimum Spanning Tree of the distance graph in order to compute which pairs of Crusaders should use their Chromatic Call. Since there’s not a large number of you, you’re not really concerned with optimizing runtime: either Kruskal’s or Prim’s algorithm will do for this task.

Download the PA9.py template from the course website.

The template contains the ChromaticCrusader class (essentially a single vertex for the purposes of MST), which stores information on each Crusader like

* their color (visual representation only, not needed for the algorithm)
* index number (a value between 0 and n-1, where n is the number of nodes: this is used for indexing into the FriendshipMatrix/distance lists)
* a list of their friends (an adjacency list in no particular order)
* a distance list (an integer list of the edge weights between this node and every other node, in order of index number: essentially this node’s row in the edge weight matrix; unlike the friends list this one will include an entry (infinity) for nodes not adjacent to this one)

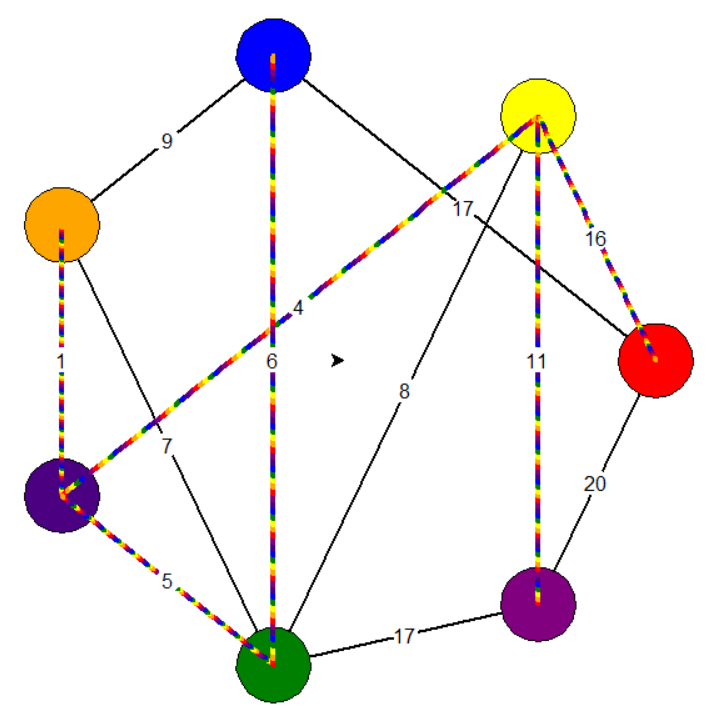
It also contains some instance variables specific to either Kruskal’s or Prim’s algorithm: .prev and .key for Prim’s, and .p and .rank for the set operations needed to implement Kruskal’s. Finally, it contains several methods, including some of the methods used for set operations for Kruskal’s algorithm, and a few methods for drawing the graph/string representation that you can safely ignore.

The template also includes a FriendshipMatrix class, representing the graph as a whole, which notably has the chromatic\_call method, which is what you will be using to add edges to your final MST.

* chromatic\_call takes two Chromatic Crusader objects and locks in the decision to have one of them Chromatic Call the other (order does not matter here: it’s assumed that whoever arrives first will call the other). This will color in the edge between the two Crusaders within the visual distance graph, as shown on the next page. Since this is essentially declaring that the edge between those two vertices will appear in the final MST, make sure that you only call this method when you are certain that it is an optimal edge.

The class also contains a .distance instance variable, which represents the full distance (weight) matrix for the entire graph, in case you would rather use that representation than the .distance list for each node.

There are also functions within the template that you may use for implementing a min-priority-queue if you want to use Prim’s algorithm, and some test cases.



Visualization of the distance graph. The colored edges represent the final MST.

You must implement the gather\_friends function, which takes as input a list of ChromaticCrusader objects (essentially a list of vertices), and a FriendshipMatrix object (essentially the weight matrix and storage for the final MST), and uses Kruskal’s or Prim’s algorithm to compute the minimum spanning tree of the graph. This function does not return anything: instead it needs to repeatedly call the chromatic\_call method of the FriendshipMatrix friend\_matrix on two vertices that it wants to connect in the final MST.

Requirements:

* You must download the template file PA9.py and edit the gather\_friends function. You can create your own helper functions, but don’t edit the code beyond the “DO NOT EDIT” line.
* You must complete the functionality of gather\_friends by using Prim’s algorithm or Kruskal’s algorithm to find Minimum Spanning Tree for the distance graph.
* Code has been included to implement the min priority queue data structure for Prim’s algorithm, or the disjoint set data structure for Kruskal’s, but you are not required to use these: feel free to edit them or implement your own.
* Your program must run without errors on the version of Python installed on the CSELabs machines, Python 3.5.2. (if you’re testing this on CSELabs, you need to type python3 or idle3 instead of python or idle to start the correct version from the terminal)
* You are still not permitted to use the input() function as this will break the grading script. However, for this assignment, sorted() and .sort() are permitted.
* You must implement either Kruskal’s or Prim’s algorithm, as found in Chapter 23 of the textbook. Any other algorithm will receive very little credit.
* This assignment will be graded automatically based on the number of test cases your program passes. There will be several secret test cases in addition to the ones included in the template to ensure you’re not hard-coding in the solutions.
* This program will only run test cases until you fail one, avoiding the problem of having to scroll through test output to find the one broken test case.
* The grading breakdown for this assignment is as follows:
  + 30%: File runs without syntax errors
  + 70%: Passing test cases without breaking any requirements.
* The unedited template file already runs without syntax errors. This means that if your program causes syntax errors, you will get a better score by just submitting the original template unedited.
* Submit your edited PA9.py file to the Programming Assignment 9 link on Canvas before 10:30 AM on 11/6/18. No credit will be given for late submissions.