CSE 332: Data Structures and Parallelism

Exercise 7 - Spec

Exercise 7 Spec (25su)

The objectives of this exercise are

- **Interpret a non-graph problem into a graph problem**
- **Implement a graph using an adjacency list representation**
- **Adapt a standard graph algorithm (depth-first search) to solve a new problem**

Overview

This exercise consists of the following parts:

1. Use given data in order to construct a graph using an adjacency list representation
2. Determine whether a graph has a particular cycle by implementing a modified depth-first search
3. Further modify this depth-first search to identify the cycle.

Motivating Application: Directed Kidney Donation

Kidneys are organs in the human body whose function is critical for maintaining life. Put simply, the kidneys primarily serve to regulate the salinity, water content, and pH of one's blood, and also remove toxins from the body. If someone has no working kidneys then they must either receive frequent kidney dialysis (which cycles blood through a machine to perform the function of a kidney), or else receive a kidney transplant from an organ donor.

Humans are generally born with two kidneys, and one functioning kidney is sufficient for survival, so it is possible (though not risk-free) for a living person to donate a kidney to a patient in need. To protect from disease, the human immune system has evolved to recognize and attack cells from other individuals. As such, before donation, we must check whether a donor and recipient are compatible. Donor-patient compatibility is determined by weighing many factors, but one important factor is HLA match compatibility.

Human Leukocyte Antigens (HLAs) are proteins that attach to the membranes of cells to help the immune system identify which cells are "self" cells and which are "invader" cells. Each person has a collection of different HLAs, and the more HLAs that are shared between the donor and the recipient the safer the transplant will be. The similarity is measured by a "HLA match score". **For this assignment, we will consider a patient and donor to be compatible in the case that their match score is at least 60.**

When someone wishes to donate a kidney, there are two main ways they can donate. They can give a “directed donation”, meaning they wish to donate to a specific person. They could also give a “non-directed donation”, meaning they choose to donate without knowing the recipient.

Because of all the factors at play for donor-recipient compatibility, it may be the case that a recipient does not know anyone who would be a matching directed donor. To help patients find donors, donation networks have been set up. The idea is that if Recipient A needs a kidney, but their friend Donor A is not compatible, then perhaps Donor A could give a kidney to compatible Recipient B in exchange for Donor B giving a kidney to Recipient A. So in short, if Recipient A has no known donors, they can try to identify a "kidney trade" with another recipient using one of their directed donors. These trades could also be larger "kidney donation cycles", where Donor A gives to Recipient B, Donor B gives to recipient C, and Donor C gives to recipient A. For the sake of this assignment, our cycles can be of any length, but they must be cycles (the idea being that Donor A is a directed donor and so is only willing to donate a kidney if the result is Recipient A receiving one).

Problem Statement

For this assignment you will implement an algorithm to identify whether a given recipient can receive a kidney through a donation cycle.
A donation cycle is defined to be a sequence of kidney recipients r_0, r_1, ..., r_x, r_0 such that for each choice of i there is a directed donor for recipient r_i who has an HLA match score of at least 60 with recipient r_{i+1}. In other words, it is a cycle of recipients such that each recipient in the cycle can receive a kidney from one of the donors associated with the recipient before them in the cycle.

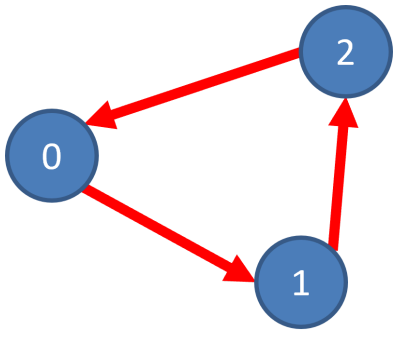
For this assignment we will give you a list of directed donors (and the recipients whom they wish to benefit) as well as HLA match scores for each donor-recipient pair. You will then convert that information into a graph, and then implement a method that will give a donor cycle which includes a given recipient, or else indicate that no such cycle exists. You may assume that no donor has a match with the recipient they wish to benefit.

Input Format: For this assignment, input will be encoded in txt files. Supposing that each test consists of n recipients and m donors, the files will contain m+4 lines as follows:

- The first line contains the value n (the number of recipients)
- The next line contains the value m (the number of donors)
- The next line contains m comma-separated integers between 0 and n-1 to indicate the recipient whom each donor wishes to benefit (so if the ith integer is j, that means donor i wishes for recipient j to receive a kidney)
- The next m lines each contain n comma-separated integers. Collectively, these n*m integers give the HLA match scores for all donor-recipient pairs. We consider a score of 60 or higher to be suitable for donation.
- The last line contains an integer between 0 and n-1 to indicate a recipient. This recipient will be given as input to your algorithm, which will attempt to find a cycle that includes that recipient.

For example, if the file's contents were as shown on the left, then the graph would appear as on the right.

```
3
9
0,0,0,1,1,1,1,2,2
10,15,20
20,20,9
0,71,20
2,2,2
3,3,3
4,4,4
5,5,100
90,0,0
90,0,0
0
```



When processing the file line-by-line we see that:

- There are 3 recipients (hence 3 nodes in the graph)
- There are 9 donors
- Donors 0, 1, and 2 wish for recipient 0 to receive a kidney, donors 3, 4, 5, and 6 wish to benefit recipient 1, and donors 7, and 8 wish to benefit recipient 2
- Donor 0 does not have a strong enough match to donate to any of the three recipients (no scores are 60 or greater)
- Donor 1 does not have a strong enough match to donate to any of the three recipients (no scores are 60 or greater)
- Donor 2 is compatible with recipient 1, therefore we draw an edge from node 0 (Donor 2's beneficiary) to node 1 (Donor 2's match)
- Repeat the above for donors 3-8
- Since the last line has the value 0, your algorithm should find a cycle which includes recipient 0. In the graph this would be [0, 1, 2, 0].

Implementation Guidelines

Your implementations in this assignment must follow these guidelines

- You may not add any import statements beyond those that are already present (except for where expressly permitted in this spec). This course is about learning the mechanics of various data structures so that we know how to use them effectively, choose among them, and modify them as needed. Java has built-in implementations of many data structures that we will discuss, in general you should not use them.
- Do not have any package declarations in the code that you submit. The Gradescope autograder may not be able to run your code if it does.
- Remove all print statements from your code before submitting. These could interfere with the Gradescope autograder (mostly because printing consumes a lot of computing time).
- Your code will be evaluated by the gradescope autograder to assess correctness. It will be evaluated by a human to verify running time. Please write your code to be readable. This means adding comments to your methods and removing unused code (including "commented out" code).
- The autograder on Gradescope is running on Java 11. You should avoid using methods in `List` such as `List.addFirst(element)`, `List.addLast(element)` - these are new methods introduced in Java 21 and gradescope might complain about that. They should have the same behavior as `List.add(0, element)` and `List.add(element)`.

For this assignment, there **is partial credit for passing only a few test cases**. For example, if you failed on any test case on `Constructor`, you will receive partial credit corresponding to that test case. `hasCycle` is worth 8 points, `FindCycle` and `Constructor` are all worth 16 points. There are still 10 points of manual inspection for your CycleDetection. thus, there are 40 points of autograder and 10 points of manual inspection. Meanwhile, you will be able to see the details on the testcases you failed. **Please note that we have two extra files that we have not provided you as our hidden test.** However, you will see if you failed the hidden test on Gradescope (that is, if the autograder says "40/40" you should be good to go).

Provided Code

Several java classes have been provided for you in the zip file on the exercises page. Here is a description of each.

- `Client`
  - This class contains the main method, which does the following:
    - Reads the input file named in the field testFileName
    - Parses the file and converts all the information into 2 arrays:
      - `donorToBenefit`: an array of ints with the property that index i contains the recipient whom donor i wishes to benefit
      - `matchScores`: a 2-d array with the property that index x,y contains the match score of donor x and recipient y. Donation is permitted if that score is at least 60.
    - Calls the constructor that you will implement, providing these arrays as input
    - Invokes the `findDonorCycle` method you will implement, giving the int on the last line of the file as the argument
  - *Do not submit this file*
- `Match`
  - These objects will be used to indicate a potential donation. It's primary role is just to encapsulate 3 fields:
    - `donor`: A donor
    - `beneficiary`: Whom that donor seeks to benefit
    - `recipient`: A recipient compatible with the donor
  - In effect, these objects will also be used to store edges in our adjacency list representation of a graph, specifically representing an edge from `beneficiary` to `recipient`
  - *Do not submit this file*
- `DonorGraph`
  - This is the file that you will be modifying for this assignment. You will create a constructor in part 1, the method `hasCycle` in part 2, and the method `findCycle` in part 3.
  - It additionally has a method called `isAdjacent` which returns a boolean to indicate whether the given pair of recipients form an edge in the graph.
  - Finally, it has a method called `getDonor` which returns an int representing the donor who enabled edge between the pair of recipients given (or returns -1 if the pair are not adjacent).
  - ***You will submit this file to Gradescope***
- Various text files
  - `example.txt`
    - The file described above
  - `straightline.txt`
    - Produces a 10-node graph that is just a line of nodes. No cycles should be present

- `slides.txt`
  - The directed graph from the slides with the right-most node as the query. No cycles should be found for the query node, but some other choices of the query should result in cycles.
- `biggerloop.txt`
  - 10 vertices in a circle. Every node should have a cycle (which is the whole graph)
- `almostcycle.txt`
  - 10 vertices in a circle, but with one edge in the opposite direction from the rest. No cycles should be present
- `complete.txt`
  - 10 vertices that are each connected to all the other vertices. Every node should have a cycle (and there are many options for what that cycle may be)
- `lollypop0.txt`
  - A graph with vertices in a "lollipop" shape, meaning there is a straight line of nodes connected to nodes in a loop. For this file, the query node is node 0 which is at the start of the "stick" of the lollipop, so no cycle should be present.
- `lollypop5.txt`
  - The same lollypop graph as above, but now the query node is in the candy portion of the lollipop. There should be a cycle found.

Part 1: Making the graph

For this part you will make the `DonorGraph` constructor. The main method in Client reads the text file, then provides its contents as arguments to your constructor. These arguments are:

- `donorToBenefit`: an array of ints with the property that index `i` contains the recipient whom donor `i` wishes to benefit
- `matchScores`: a 2-d array with the property that index `x,y` contains the match score of donor `x` and recipient `y`. Donation is permitted if that score is at least 60.

Your constructor should use those two arrays to populate the `adjList` field with `Match` objects to become an adjacency list representation of a directed graph. All of the Match objects at `adjList.get(i)` should have `i` as the value in the `beneficiary` field. It will then represent an edge to node `j` provided that `j` is in the recipient field. The match object should only exist if `donor` has a match score of at least 60 with `recipient`. None of the tests used will include a case where a donor is a suitable match for their beneficiary.

There are multiple ways you can implement this constructor. If you would like for the graph to be a simple graph, note that having duplicate `Match` objects with the same beneficiary and recipient (but different donors) will not change the presence or identity of any cycles. This means you can make sure there is only one `Match` object to represent each edge in the graph. There's nothing wrong with having multiple, but you may need to remember that this will cause the graph to be non-simple in parts 2 and 3.

Part 2: hasCycle

For this part you will write the method `hasCycle`. The argument to this method is a recipient, and the method should return true if there is a donation cycle which includes that recipient, and false if there is not. We did not explicitly discuss cycle detection in lecture, but these slides should provide a bit more background. The only difference between the problem solved in these slides and the behavior of this method is that on the slides we want to return true whenever *any* cycle is present, for this method you want to return true if there is a cycle which includes a specified node. The running time of the algorithm should be O(n*m) where n is the number of recipients and m is the number of donors.

Part 3: findCycle

For this part you will write the method `findCycle`. The argument to this method is a recipient, and the method should return a list of matches to indicate the cycle. If you think of each Match object as an edge in the graph, this list of matches is the list of edges followed to create the cycle. The list should be ordered such that the `recipient` given as the argument is the `beneficiary` of the first match in the list and also is the `recipient` of the last match of the list. For all matches in between the `recipient` of one match must be the `beneficiary` of the next

(i.e. the edges can be followed in the order they are listed). The running time of the algorithm should be O(n*m) where n is the number of recipients and m is the number of donors.