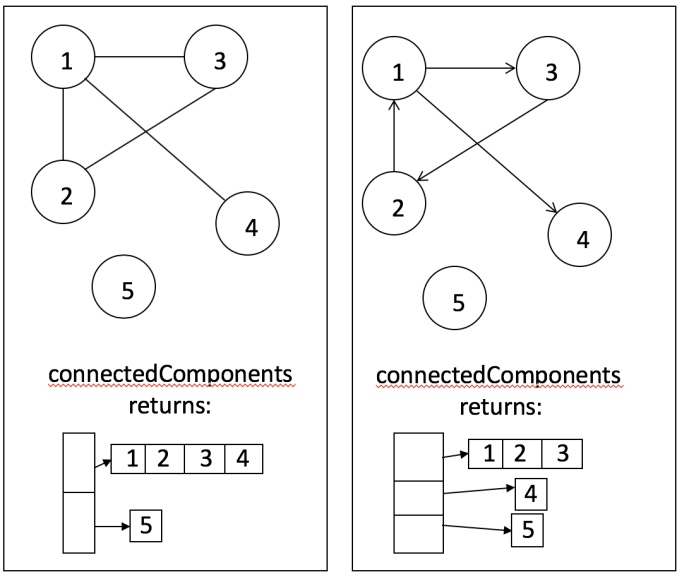
Assignment 3

Graphs

Exercise 1

Implement two classes for unweighted graphs (MyDirectedGraph and MyUndirectedGraph) with operations addVertex(int vertex), addEdge(int sourceVertex, int targetVertex).

- Implement in each class a method "public boolean isAcyclic()" that returns true if the graph is acyclic.
- Implement in each class a method "public boolean isConnected()" that checks if the graph is connected (in the case of MyDirectedGraph, isConnected() returns true if the graph is strongly connected).
- Implement in each class a method connectedComponents() that returns a List of Lists of integers (List<List<Integer>> connectedComponents()) that returns the nodes in each connected component of the graph (in the case of MyDirectedGraph, in each strongly connected component). Next figure shows examples of what connectedComponents() should return for directed and for undirected graphs.



To make easier the implementation, you can assume that vertices will always be added with sequential indexes starting from 0. This means that it will NOT happen the following:

```
MyDirectedGraph graph = new MyDirectedGraph();

graph.addVertex(100);

graph.addVertex(15);
```

Implement operations isAcyclic() and isConnected() as fast as you can. Attach a pdf document where you explain your solution and you analyze the time complexity of all operations. The grade of the exercise will NOT consider the quality of the implementation of connectedComponents() (do not worry about making it fast), but it will consider the correctness of its complexity analysis.

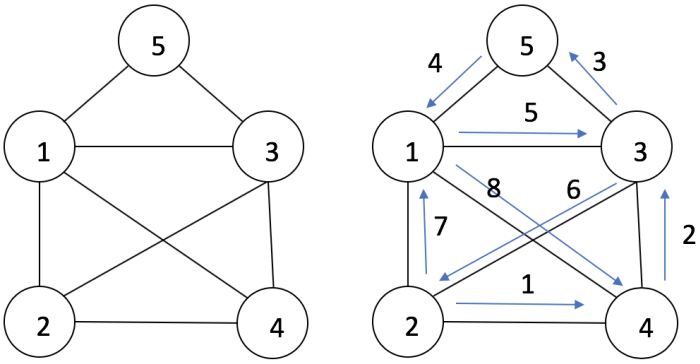
Exercise 2

A common game is to find how to traverse all lines in a drawing passing only one time through each line and without lifting the pen. In graphs, an Euler path is a path that traverses every edge exactly once (see Chapter 9.6.3 in the course reference book, which explains how to implement it).

Implement in MyUndirectedGraph methods:

- public boolean hasEulerPath(), which returns true if the graph has an Euler path

- public List<Integer> eulerPath(), which returns a list of vertices of length |E|+1. The vertices in the list represent how to traverse the graph to complete an Euler path. Next figure shows an example.



EulerPath() returns a list with elements: <2,4,3,5,1,3,2,1,4>

Attach a pdf document where you explain your solution and you analyze the time complexity of all operations.

Exercise 3

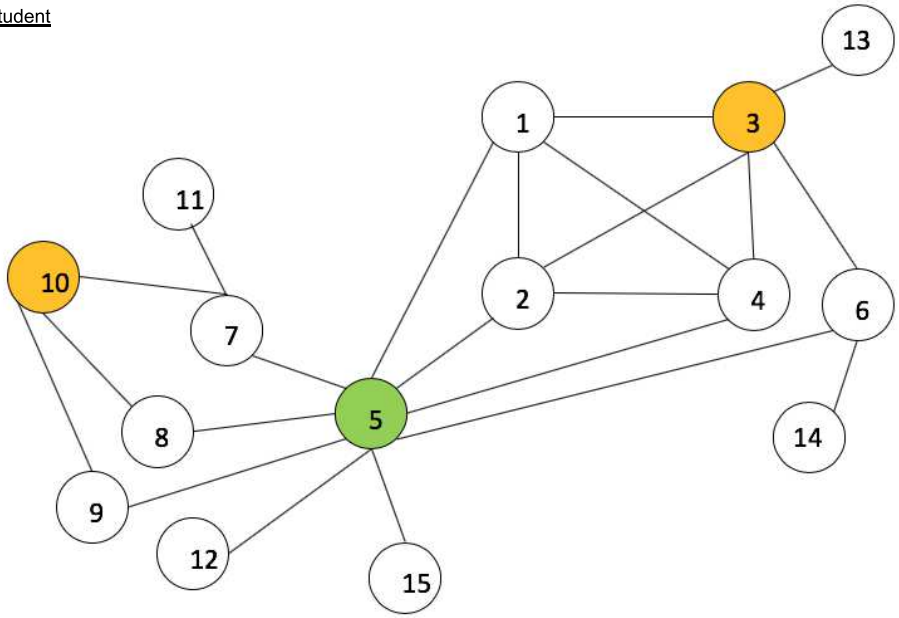
Friendship relationships in a social network can be represented as a graph, where vertices are people and edges represent the existence of a friendship relationship between two people. Assuming that friendship is reciprocal, we can use an undirected graph.

Implement in MySocialNetwork class the following operations useful for social network:

- public int numberOfPeopleAtFriendshipDistance(int vertexIndex, int distance). It returns the number of people that are at distance "distance" of the person given by "vertexIndex". For instance, for distance=2, it returns the number of people who are friends of my friends (but who are not directly my friends or myself). In the graph in the next figure, numberOfPeopleAtFriendshipDistance(5, 2) will return 4 (there are 4 vertices at distance 2 of vertex 5: <10,11,14,3>)

- public int furthestDistanceInFriendshipRelationships(int vertexIndex). Given a person in "vertexIndex", it returns the distance to furthest person in the graph from vertexIndex (this is, returns the highest value of the shortest paths between "vertexIndex" and the rest of nodes). In the graph in the next figure, furthestDistanceInFriendshipRelationships(5) will return 3 (node 13 is at distance 3 of node 5, and there is not any other node more distant than node 13)
- public List<Integer> possibleFriends(int vertexIndex). Given a person in "vertexIndex", it returns the list vertices that are at distance 2 of "vertexIndex" (this is, they are friends of friends), where each of these vertices shares at least three common friends with "vertexIndex". In the graph in the next figure possibleFriends(5) are <3,10>. Vertex 3 shares 4>3 friends with vertex 5 (1,2,4, and 6); while vertex 10 shares exactly 3 friends with vertex 5 (7,8,9). Note that also vertices 14 and 11 are at distance 2 of vertex 5, but they share only one friend with node 5, therefore they are not considered possible friends.

- *(Optional method)* public List<Integer> probableFriends(int vertexIndex). Given a person in "vertexIndex", it returns a list vertices. Each vertex *v* in the returned list must satisfy:
  - v is at distance 2 of "vertexIndex" (this is, they are friends of friends),
  - v shares at least three common friends with "vertexIndex", and
  - the induced subgraph (https://en.wikipedia.org/wiki/Induced_subgraph) that contains vertex v and these three common friends is a complete graph. In other words, v and the three common friends create a clique of size 4. This means that all four people are friends of all of them, and the only lacking friendship relationship to create a clique of size five is the edge between "vertexIndex" and v. In the graph in the next figure probableFriends(5) will return <3>, because vertex 3 has at least three common friends with vertex 5 (vertices <1,2,4,6>) and vertices <1,2,3,4> are a clique.



Implement all operations as fast as you can, and attach a pdf document where you explain your solution and you analyze their time complexity. The grade of the exercise will consider the quality of the implementation and the correctness of the analysis. **NOTE:** The previous sentences do not apply to the optional exercise. A sketch of a solution in high-level pseudocode and some reasonable description of the time complexity would be enough.

Important: You must implement the ADTs without reusing Graph data structures from libraries. This applies to all exercises in this assignment.

For extra-points: Instead of using "int" for referring to vertices, implement your operations assuming a generic type for the vertices. Therefore, all the operations with directed, undirected, and social network that require or return a vertex or a list o vertices will consider this generic type. If you follow this option, see the interfaces, classes and tests inside the "generic" folder in the attached .zip. If you don't follow this option, see the tests folders (our tests will follow a very similar structure, eliminating the code for generics and using vertices of "int" type).

- assignment3AADS.zip 22 October 2019, 12:57 PM

Submission status

Attempt number	This is attempt 1.
Submission status	No attempt
Grading status	Not graded
Due date	Thursday, 7 November 2019, 11:55 PM
Time remaining	1 day 1 hour
Last modified	-
Submission comments	Comments (0)

Add submission

You have not made a submission yet.

Previous exams ►