# CS 251, Data Structures and Algorithms, Fall 2016

# Project 5: Computing graph properties

- **Part 1 Due Date:** Friday, December 2, 11:59pm. A maximum of 1 late day (including slip days).
- **Part 2 Due Date:** Friday, December 9, 11:59pm. A maximum of 3 late days (including slip days).
- **Late Penalty**: 20% per day.
- **Skeleton Code and Sample Graphs**: Project5
- **Part 2 Graph Files**: Graph Implementation

## Introduction

In this final project you will write methods to determine various properties of graphs. Graphs will be represented by either adjacency lists or adjacency matrices, they will be undirected or directed, and the properties will range from simple checks to graph traversals to more expensive computations. Some of the properties are likely to arise in your later courses.

## Definitions and notation

- For this project we will use the following commonly used notation:
  - n - number of vertices in the graph
  - m - number of edges in the graph
  - vertices are numbered from 0 to n-1
  - (u,v) represents the *undirected* edge from vertex u to vertex v
  - <u,v> represents the *directed* edge from vertex u to vertex v

## Part 1

### Warmup: Implementing Graphs

Before you can calculate anything about a graph, we need a data structure representing graphs. Recall that in class we have discussed two ways of representing a graph:

- Adjacency Matrix: A n x n matrix M; M[u][v] = 1 if (u,v) (or <u,v>) is an edge of graph G and M[u][v] = 0 otherwise. For undirected graphs, M[u][v] = 1 implies M[v][u] = 1.
- Adjacency lists: An array A of size n; A[u] points to the list of vertices incident to vertex u. For directed graphs, the adjacency list of vertex u contains the vertices of the outgoing edges. For undirected graphs, edge (u,v) has u in v's list and v in u's list.

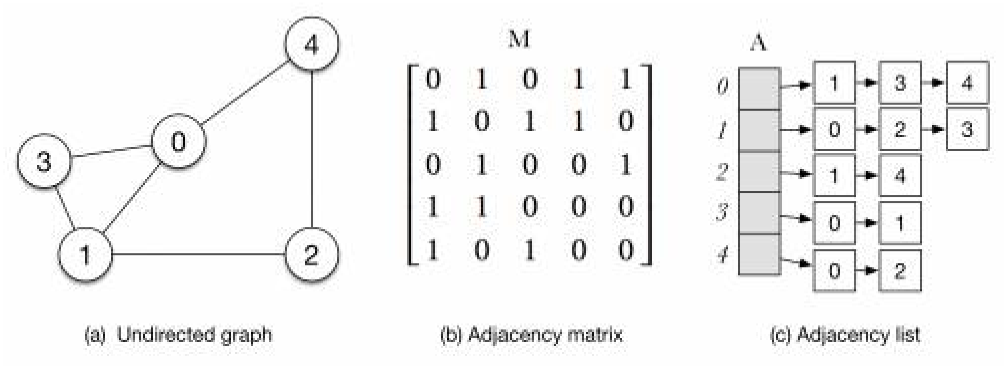

Figure 1: undirected graph and corresponding representations

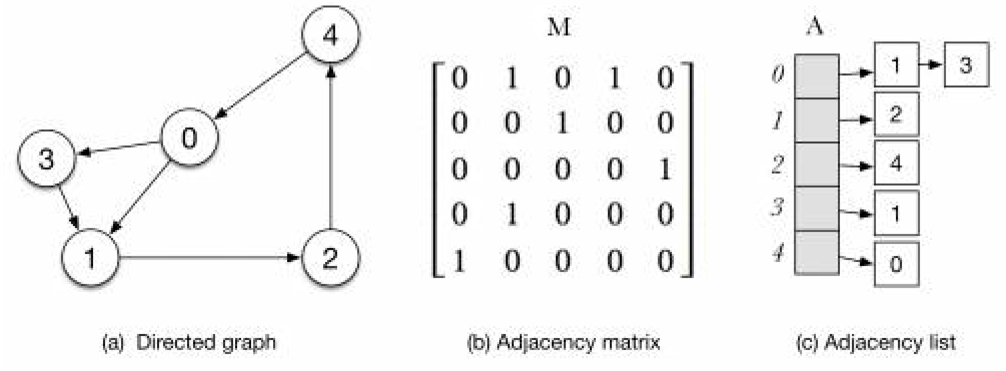

Figure 2: directed graph and corresponding representations

You will be implementing both adjacency matrix and adjacency lists graph representations. Your adjacency matrix version should be implemented in MatrixGraph.java and your adjacency list version should be in ListGraph.java. Both of these classes implement the abstract class found in Graph.java. The constructors for these files have already been written to work with the specified input format.

For the warmup, implement the basic methods required from a graph class:

- addEdge(u,v) - Adds an edge from u to v
- hasEdge(u,v) - Returns true if the graph contains an edge from u to v
- getAdjacentVertices(v) - Returns a list of vertices adjacent to v
- removeEdge(u,v) - Removes the directed edge from u to v, if it exists

Implement each method in the two Graph .java files according to the data structure specified. For part 1 you may add additional methods to your graph files if you believe they would be helpful. For part 2 you must use only the implementation that we provide.

## Undirected Graph Properties

Let G=(V, E) be an undirected graph. In many applications using undirected graphs, testing for certain properties is common. You should consider the following properties:

- What is the median degree of the vertices in G? The degree of vertex u is the number of edges incident to u.
- Does graph G contain a connected component that contains at least $\lceil \frac{n}{2} \rceil$ vertices? If yes, return a list containing the vertices in this connected component. If not, return an empty list. Your method should use DFS or BFS.
- Given a graph G and 5 vertices, determine if the 5 vertices form a clique. A clique is a graph

containing all possible edges. A clique on 5 vertices has thus 10 edges. Check that you are given five distinct vertices.
- Determine if graph G contains 6 vertices with degree at least $\lfloor \sqrt{n} \rfloor$. If yes, return the indices of the vertices you found. If not, return an empty list.
- Determine if graph G contains 6 vertices having degree less than 6. If yes, return the indices of the vertices you found. If not, return an empty list.

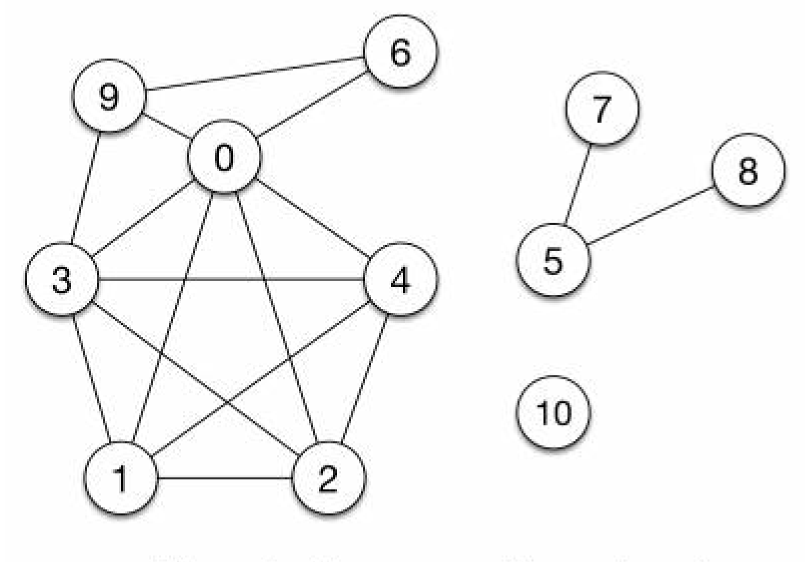

Figure 3: 11-vertex undirected graph

Figure 3 shows an 11-vertex undirected graph. This graph has the following properties:

- The median of the degrees is 3. Degrees of the 11 vertices are 6,4,4,5,4,2,2,1,1,3, and 0.
- The graph contains a connected component consisting of 7 vertices. The list of vertices is {0,1,2,3,4,6,9}.
- Vertices 0,1,2,3, and 4 form a clique of size 5. If given these five vertices, return true. The graph has no other clique of size 5.
- Since n = 11, $\lfloor \sqrt{n} \rfloor$ = 3. Six vertices have degree at least 3; they are 0,1,2,3,4, and 9.
- Except for vertex 0, all vertices have degree less than or equal to 5. Thus, the graph contains 6 such vertices. Return, for example, list {1,2,3,4,5,6}.

Your implementations of the graph properties should go in Part1.java under the following methods:

- double medianDegree(Graph G)
- List<Integer> hasGiantConnectedComponent(Graph G)
- boolean is5Clique(Graph G, List<int> vertices)
- List<Integer> has6DegreeRootN(Graph G)
- List<Integer> has6Degree6(Graph G)

**Hint:** You may want to add some helper methods to your graph files to accomplish the degree problems efficiently

## Part 2: Properties of Directed Graphs and Clustering Coefficients

For completing Part 2, we will release an official version of Graph.java, ListGraph.java (along with a LinkedList.java for the adjacency lists), and MatrixGraph.java. Use these versions for your Part 2 submission. If you finish Part 1 early you can begin your implementation by using your version to work on Part 2, but be sure to move your code over to the version of Part 1 which we release and make sure that your code still works with this version (which contains only the methods listed in the skeleton). You can move your helper functions to the official version of Project 1 if needed.

### Properties of Directed Graphs

Let G=(V,E) be a directed graph. Consider the following properties:

- Determine maximum in-degree of the vertices in G.
- Determine the maximum out-degree of the vertices in G.
- Use BFS or DFS to determine whether the graph contains exactly one directed cycle. If so, return a list of the vertices in that one cycle. (Starting vertex does not matter). If not, return an empty list.

Your implementations of the graph properties should go in Part2.java under the following methods:

- int maxInDegree(Graph G)
- int maxOutDegree(Graph G)
- list<Integer> hasOneCycle(Graph G)

### Clustering Coefficients

Let G=(V,E) be an undirected graph. Consider the following triangle related properties:

- Given an edge (u,v), determine the number of triangles containing edge (u,v). A triangle is defined as three vertices u, v, and w such that (u,v),(v,w), and (w,u) are all edges in the graph G. Edge (7,8) = (u,v) of the graph shown in Figure 4 is contained in 4 triangles.
- Given a graph G, determine the total number of triangles in the graph. The graph shown in Figure 4 contains a total of 8 triangles.

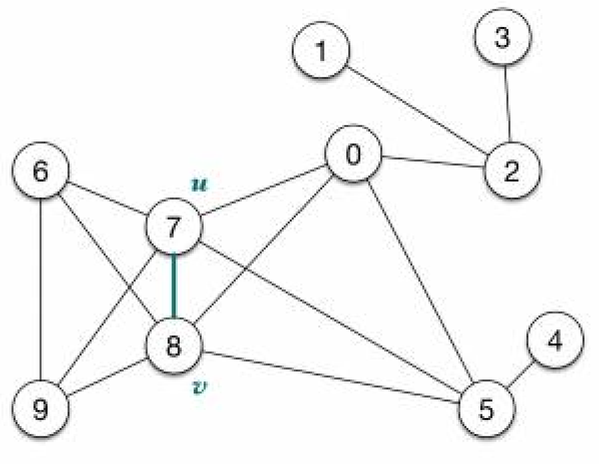

Figure 4: 10-vertex undirected graph

In graph G, assume an edge between vertices A and B means that A and B are friends. The **clustering coefficient** of a person is the fraction of pairs of friends of A that are friends with each other. The clustering coefficient is 0 if when none of A's friends are friends with each other and is 1 when all of A's friends are friends with each other. The **global clustering coefficient** is defined as the average of the n clustering coefficients.

For the graph shown in Figure 4, the clustering coefficient of vertex 9 is 1: all its friends (i.e., vertices 6, 7, and 8) are friends with each other. The clustering coefficient of vertex 2 is 0 since none of its friends (0, 1, 3) are friends with each other. The clustering coefficients for the 10 vertices are are 0.5, 0, 0, 0, 0, 0.5, 1, 0.6, 0.6, and 1; the global clustering coefficient is 0.42.

The global cluster coefficients for four undirected graphs provided are listed below.

| **File** | **GCC** |
|---|---|
| complete5 | 1.000000 |
| cluster | 0.420000 |
| largerandom | 0.019706 |
| smallworld | 0.526389 |

**NOTE:** When testing you may see that you are getting numbers very close to the answers you expect (like 0.42000000001 instead of 0.42). This is fine, and happens often when working with floats/doubles. As long as your answer is within 1/10,000 [correct to 5 decimal places] it will be counted as correct.

Your implementations of the graph properties should go in Part2.java under the following methods:

- int numEdgeTriangles(Graph G, int u, int v)
- int numTrianlges(Graph G)
- double vertexClusterCoeff(Graph G, int v)
- double globalClusterCoeff(Graph G)

## Input format

The skeleton code provides all input and output handling code for you. The following information can be used to work with your own testcases and to test your code.

Input consists of files that represent graphs. They are formatted as follows:

```
<Graph implementation type>
<Number of vertices>
<vertex number>: <list of incident vertices>
<vertex number>: <list of incident vertices>
<vertex number>: <list of incident vertices>
…
<commands to run>
```

The list of adjacent vertices is not guaranteed to be in sorted order!
<Graph implementation type> is either “matrix” or “list”. It tells you what graph implementation to use. Test each graph with both implementations.
<commands to run> is a list of commands. The commands for Part 1 and Part 2 are given below.

Part 1:
medianDegree
giantComponent
is5Clique <list of 5 vertices>
has6DegreeRootN
has6Degree6

Part 2:
maxInDegree
maxOutDegree
hasOneCycle
numEdgeTriangles
numTriangles
vertexClusterCoeff
globalClusterCoeff

For example, a file representing a complete undirected graph on 5 vertices that executes all five Part 1 methods using an adjacency matrix representation would contain:

```
matrix
5
0: 1 2 3 4
1: 0 2 3 4
2: 1 0 3 4
3: 0 1 2 4
4: 3 2 1 0
medianDegree
giantComponent
is5Clique 0 1 2 3 4
has6DegreeRootN
has6Degree6
```

Its expected output would be:

```
4.0
[0, 1, 2, 3, 4]
true
[]
[]
```

For the same example with the part 2 problems the input would be:

```
<same graph>
maxInDegree
maxOutDegree
hasOneCycle
numEdgeTriangles 0 1
numTriangles
vertexClusterCoeff 0
globalClusterCoeff
```

Its expected output would be:

```
4
4
[]
3
10
1.0
1.0
```

Files are passed to the program through standard input. You can either manually enter the graph or you can use input redirection to work with files. Also note that to represent an edge (u,v) in an undirected graph, we add an edge from u to v and an edge from v to u.

You can run the program by compiling it with javac *.java (or your IDE's equivalent) and typing

```
java Project5 < input.txt
```

If you are interested in using graph visualization tools, you may want to explore:

- graphviz - http://www.graphviz.org/Home.php
- Cytoscape - http://www.cytoscape.org/

## Analysis Questions

**Important:** All analysis questions must be answered using the graph files we provide for Part 2 (i.e., Graph.java, MatrixGraph.java, ListGraph.java, LinkedList.java). Even if your implementations may result in a better run-time, your answers need to use our implementations. Answers based on your implementation will not be graded. Look carefully at the implementation we provide as some details may be different from yours and may impact performance.

Your **typed** report must answer each question briefly and clearly. All material must be readable and clear and prepared in a professional manner.

### Analytical Questions

For each method listed below, provide the asymptotic running time of the method for using an adjacency matrix and for using adjacency lists. Give the asymptotic worst-case running time of each method using the graph files we posted. Express bounds in terms of n and m. Provide a brief (1-3 sentences) explanation for each answer. You should thus provide the run times + explanation of 2*10 = 20 implementations.

1. removeEdge
2. hasEdge
3. getAdjacentVertices
4. medianDegree
5. hasGiantConnectedComponent
6. is5Clique
7. has6DegreeRootN
8. has6Degree6
9. maxInDegree
10. hasOneCycle

## Bonus: Extended Analysis

For 10 points of extra credit, find asymptotic running time bounds your clustering coefficient algorithms (single vertex and global) and give a detailed explanation as to why the stated bounds are achieved by your implementation. Do the analysis only for the Adjacency list implementation. Bonus points will be awarded based on efficiency of your algorithm and correctness of your analysis. If you choose to do the bonus question put it at the end of your written report.

## Summary

Part 1 and Part 2 contain the following:

Part 1:

- addEdge(u,v) - Adds an edge from u to v
- removeEdge(u,v) - Removes a directed edge from u to v, if it exists
- hasEdge(u,v) - Returns true if the graph contains an edge from u to v
- getAdjacentVertices(v) - Returns a list of vertices adjacent to v

- double medianDegree(Graph G)
- List<Integer> hasGiantConnectedComponent(Graph G)
- boolean is5Clique(Graph G, List<int> vertices)
- List<Integer> has6DegreeRootN(Graph G)
- List<Integer> has6Degree6(Graph G)

Part 2:

- int maxInDegree(Graph G)
- int maxOutDegree(Graph G)
- list<Integer> hasOneCycle(Graph G)
- int numEdgeTriangles(Graph G, int u, int v)
- int numTrianlges(Graph G)
- double vertexClusterCoeff(Graph G, int v)
- double globalClusterCoeff(Graph G)

- Report

## Testing your project

Several test files are provided in sampletests/. You should note that these are only the graph files; to run them you will need to add commands to the end. Each test case arbitrarily starts with either “list” or “matrix”, to test both versions of your graph you can change the first line of the test file. The five graphs represent the following:

- complete5.txt: A complete graph on 5 vertices
- directed.txt: A small directed graph
- cluster.txt: The graph used in the clustering coefficient example
- random.txt: A randomly generated graph
- smallworld: A large graph

As always, these sample test cases are not representative of all possible cases that can happen. You should write your own test cases to test your projects thoroughly.

## Grading

Project 5 is worth 100 points. The bonus question is worth 10 points of extra credit.

- **10** Graph representation
- **25** Undirected graph properties
- **20** Directed graph properties and triangles
- **20** Clustering coefficients
- **20** Report
- **5** Code Quality

## Submission Instructions

**You are responsible for ensuring your submission meets the requirements. If not, points will be deducted from code quality allocation.** You project will not be graded before the deadline. Your score on Vocareum will remain 0 until we grade your project.

### Code

- Submit only the source files of your implementation. This means .java files only.
  - No .jar files or .class files should be submitted.
  - If a precompiled binary file is submitted without the .java file, you will receive a 0.
- In each of the `.java` files, you must correctly fill your name, login ID, your project completion date, and your PSO section number.
- Your code must compile to be graded by the autograder. Be sure to test your code before submitting.
- You should submit
  - Part 1

        - Graph.java, MatrixGraph.java, ListGraph.java, and Part1.java
        - Any extra .java classes you wrote
    - Part 2
        - Part2.java
        - Any extra .java classes you wrote
        - Your report in one .pdf file. Maximum size allowed is 10MB.
- If you submit Project5.java it will be removed and replaced during grading

## Report

- File name: `<username>.pdf`
- Your report must be typed.
    - Use LaTeX, OpenOffice Write, or Microsoft Word.
    - State your name on top of the report.
    - Handwritten reports, including digitized versions, will not be accepted.
    - Figures and diagrams that **cannot** be easily generated may be hand drawn and digitized for inclusion into the .pdf file representing your report. Note that tables and graphs that can easily be generated by Excel or similar **cannot** be hand drawn!
- Your report must be submitted as a PDF.
    - Other file formats will not be accepted.
- Your report must be submitted using Vocareum.
    - Submissions via any other means, to any other venue, will not be accepted.

All work must be submitted in Vocareum following the submission instructions at Vocareum Submission Guidelines. Please be aware of the following:

- Only your final submission will be graded.
- A submission **after** the deadline is considered late. DON'T wait until the last second to submit.
- If you are submitting late and want to use your slip days, you need to select “use slip days” when submitting on Vocareum.
- See top of project description of allowed late/slip days for Part 1 and Part 2