COSC 2123/1285 Algorithms and Analysis
Semester 1, 2018

Assignment 1: Six Degrees of Separation

Due date: 11:59pm Thursday, 12^{th} of April, 2018 **Weight:** 15%
Pair (Group of 2) Assignment

1 Objectives

There are three key objectives for this assignment:

- Understand how a real problem can be mapped to graphs and its operations.
- Use a number of fundamental data structures to implement the *graph* abstract data type.
- Evaluate and contrast the performance of the data structures with respect to different usage scenarios and input data.

2 Background

You may of heard of the phrase, "six degrees of separation", but did you know where it came from? It is based on the idea that anyone in the world is more or less separated from anyone else in the world by 6 direct acquaintances, i.e., you know someone who knows someone who knows someone ... Although the phrase is often incorrectly attributed to American psychologist Milgram and a few others, nevertheless, it is of interest to understand the set of experiments that popularised the phrase. In 1967, Milgram setup an experiment to test the small world phenomena (everyone is linked to everyone else via a short chain of acquaintances). He gave around 300 volunteers a name of someone in the US and instructions to send the letter to the person if they know them, or if they don't, to send the letter an acquaintance that they think would know this person. Before sending on the letter, they should record their name in a roster and send this with the letter. If the letter get to the target person, this person is asked to send the roster, now containing the chain of persons, back to Milgram. Analysing the rosters that came back, he found on average, it took a chain of about 6 persons to reach the target people, which lead to the term of "six degrees of separation".

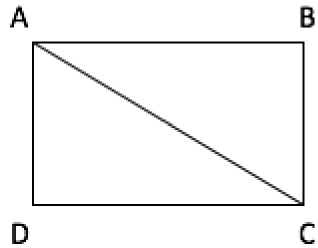
This problem can be studied in terms of graphs, specifically social networks modelled as graphs. The vertices in such a graph represent people and edges represent relationships. There are many types of social networks, in this assignment, we are interested in friendship graphs. In a friendship graph, the edges are *undirected*[1] and represent friendship. The separation between a pair of persons can be measured by the shortest path distance between their respective vertices, and we can replicate Milgram's experiment by calculating the average shortest path distance between all pairs of vertices (representing people).

In class, we studied two methods to represent a graph, ***adjacency list*** and ***adjacency matrix***. Besides these two representations, a graph can also be represented using ***incidence matrix***.

The ***incidence matrix*** of a graph is an `m x n` matrix A with `m` and `n` are the number of vertices and edges of the graph respectively, such that $A_{i,j} = 1$ if vertex v_i and edge e_j are incident, i.e. v_i is an endpoint of e_j, and 0 otherwise.

For example, the following graph:

[1]We assume reciprocal friendship in this assignment, but there is a model where this assumption is not made.



has its incidence matrix as below:

$$\begin{array}{c c} & \begin{array}{ccccc} AB & AC & AD & BC & CD \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Some extra resources about incidence matrix:

- `https://en.wikipedia.org/wiki/Incidence_matrix`
- `http://mathonline.wikidot.com/incidence-matrices`

The performance of each representation varies depending on the characteristics of the graph. In this assignment, we will implement ***adjacency matrix*** and ***incidence matrix*** representations, and evaluate how well they perform when representing a friendship graph and computing the average separation between a pair of people.

3 Tasks

The assignment is broken up into a number of tasks, to help you progressively complete the project.

Task A: Implement the Graph Representations and their Operations (8 marks)

In this task, you will implement the undirected graph using *adjacency matrix* and *incidence matrix* representations. Each representations will be implemented by a data structure. Your implementation should support the following operations:

- Create an empty undirected graph (implemented as a constructor that takes zero arguments).
- Add a vertex to the graph.
- Add an edge to the graph.
- Neighbours of a vertex in the graph.
- Remove a vertex from the graph.
- Remove an edge from the graph.
- Print out the set of vertices of the graph.
- Print out the set of edges of the graph.
- Compute the shortest path between a pair of vertices in the graph.

Data Structure Details

Graphs can be implemented using a number of data structures. You are to implement the graph abstract data type using the following data structures:

- Adjacency matrix, using a 2D array (an array of arrays).
- Incidence matrix, also using a 2D array (an array of arrays).

Operations Details

Operations to perform on the implemented graph abstract data type are specified on the command line. They are in the following format:

```
<operation> [arguments]
```

where operation is one of {AV, AE, N, RV, RE, V, E, S, Q} and arguments is for optional arguments of some of the operations. The operations take the following form:

- `AV <vertLabel>` – add a vertex with label 'vertLabel' into the graph.
- `AE <srcLabel> <tarLabel>` – add an edge with source vertex 'srcLabel' and target vertex 'tarLabel' into the graph.
- `N <vertLabel>` – Return a set of neighbours for vertex 'vertLabel'. The ordering of the neighbours does not matter. See below for the required format.
- `RV <vertLabel>` – remove vertex 'vertLabel' from the graph.
- `RE <srcLabel> <tarLabel>` – remove edge with source vertex 'srcLabel' and target vertex 'tarLabel' from the graph.
- `V` – prints the vertex set of the graph. See below for the required format. The vertices can be printed in any order.
- `E` – prints the edge set of the graph. See below for the required format. The edges can be printed in any order.
- `S <vertLabel1> <vertLabel2>` – compute the shortest path distance between vertex 'vertLabel1' and vertex 'vertLabel2'. If there is no path between the two vertices, then their distance is -1. See below for the required format.
- `Q` – quits the program.

The format of the output of a neighbour operation for vertex 'A' should take the form:

```
A <list of neighbouring vertices>
```

If a vertex has no neighbours, then the neighbour list should be empty.

The print vertex operation output the vertices in the graph in a single line. The line should specifies all the valid vertex (indices) in the graph.

```
<vertex1> <vertex2> <vertex3> ...
```

The print edge operation output the edges in the graph in over a number of lines. Each line specifies an edge in the graph, and should be in the following format:

`<srcVertex> <tarVertex>`

The shortest path distance operation outputs the distance between two vertices, and should be in the following format:

`<srcVertex> <tarVertex> <distance>`

As an example of the operations, consider the output from the following list of operations:

AV A
AV B
AV C
AV D
AV E
AV F
AE A B
AE C B
AE B D
AE A E
AE D C
N A
N F
S E B
S E C
S B F
RV D
RE A B
AV G
V
E
Q

The output from the two neighbour operations ('N A', 'N F') should be:

A B,E
F

The output from the three distance operations ('S E B', 'S E C', 'S B F') should be:

E B 2
E C 3
B F −1

The output from the print vertices operation (V) could be (remember that the order doesn't matter):

A B C E F G

The output from the print edges operation (E) could be (remember that the order doesn't matter):

A E
B C
E A
C B

Testing Framework

We provide Java skeleton code (see Table 1) to help you get started and automate the correctness testing. You may add your own Java files to your final submission, but please ensure that they work with the supplied Python testing script (see below).

file	description
`GraphTester.java`	Code that reads in operation commands from stdin then executes those on the selected graph implementation. *Do not modify this file.*
`FriendshipGraph.java`	Abstract class for graph. All implement ions should extend the FriendshipGraph class defined in this file. *Do not modify this file.*
`AdjMatrix.java`	Code that implements the adjacency matrix implementation of a graph. Complete the implementation (implement parts labelled "Implement me!").
`IndMatrix.java`	Code that implements the incidence matrix implementation of a graph. Complete the implementation (implement parts labelled "Implement me!").

Table 1: Table of Java files.

In addition, we provide a Python script that automates testing, based on input files of operations (such as example above). These are fed into the Java framework which calls your implementations. The outputs resulting from any print operations are stored, then compared with the expected output. We have provided two sample input and expected files for your testing and examination.

For our evaluation of the correctness of your implementation, we will use the same Python script and input/expected files that are in the same format as the provide examples. To avoid unexpected failures, please do not change the Python script nor GraphTester.java. If you wish to use the script for your timing evaluation, make a copy and use the unaltered script to test the correctness of your implementations, and modify the copy for your timing evaluation. Same suggestion applies for GraphTester.java.

As the instructions for the assignment is getting too lengthy, instructions on how the python script runs are available within the header of the script.

Notes

- Use the output of the provided *sample* implementation (sample is the actual name of the implementation) to help you determine the right output format for the operations. If you correctly implement the "Implement me!" parts, you in fact do not need to do anything else to get the correct output formatting. GraphTester.java will handle this.

- We will run the supplied test script on your implementation on the university's core teaching servers, e.g., `titan.csit.rmit.edu.au`, `jupiter.csit.rmit.edu.au`, `saturn.csit.rmit.edu.au`. If you develop on your own machines, please ensure your code compiles and runs on these machines. You don't want to find out last minute that your code doesn't compile on these machines. If your code doesn't run on these machines, we unfortunately do not have the resources to debug each one and cannot award marks for testing.

- All submissions should compile with no warnings on **Oracle Java 8**.

Test Data

We provide a friendship graph of about 4000 vertices in a file called "facebook_combined.txt". This is real, sampled part of the Facebook friendship graph[2], with the users anonymised. The following is not part of your assessment, but for your own interest and learning, evaluate if the average distance between any two vertices is around 6. Be careful of the -1 distances.

Task B: Evaluate your Data Structures (7 marks)

In this second task, you will evaluate your three implemented structures in terms of their time complexities for the different operations and different use case scenarios. Scenarios arise from the possible use cases of a social network (friendship graph).

Write a report on your analysis and evaluation of the different implementations. Consider and recommend in which scenarios each type of implementation would be most appropriate. The report should be **6 pages or less**, in font size 12. See the assessment rubric (Appendix A) for the criteria we are seeking in the report.

Use Case Scenarios

Typically, you use real usage data to evaluate your data structures. However, for this assignment, you will write data generators to enable testing over different scenarios of interest. We are also interested in the effect of the density of the graph[3] on these scenarios. There are many possibilities, but for this assignment, consider the following scenarios:

Scenario 1 Growing friendship graph (Additions): In this scenario, people are joining the social network and the friendship graph is growing in terms of vertices and edges. In this scenario, you are to evaluate the performance of your implementations in terms of:

- vertex addition
- edge addition

Assume the graph that you start with is the facebook_combined.txt one. You are to evaluate the performance the vertex and edge addition operations as the density of the initial graph is varied.

Scenario 2 Neighbourhoods and shortest paths': In this scenario, the graph is not changing, but important operations such as neighbourhood and shortest path distances are requested.

Assume the graph that you start with is the facebook_combined.txt one. You are to evaluate the performance of the the neighbourhood and shortest path implementations as the density of the initial graph is varied.

Scenario 3 Shrinking friendship graph (Removals): Although not often occurring for a friendship graph, people do leave the social network or defriend others. In this scenario, you are to evaluate the performance of your implementations in terms of:

- vertex removal
- edge removal

---

[2] Data was obtained from `https://snap.stanford.edu/data/`

[3] Recall density = $\frac{\text{number of edges}}{\text{number of vertices}^2}$

Assume the graph that you start with is the one that we provided you with. You are to evaluate the performance the vertex and removal operations as the density of the initial graph is varied.

When generating the vertices and edges to add, remove, find neighbourhood and shortest path distances for, the distribution of these elements, compared to what is in the graph already, will have an effect on the timing performance. However, without the usage and query data, it is difficult to specify what this distributions might be. Instead, in this assignment, uniformly sample from a fixed range, e.g., 0 to 4038 when generating the vertices and edges for edge addition, removals, neighbourhoods and shortest path distances (4038 is the largest vertex label), and a different range when adding vertices (we do not want to repeatingly add vertices that are in the graph already).

Analysis

In your analysis, you should evaluate each of your representations and data structures in terms of the different scenarios outlined above. For generating graphs with different initial densities, you may want to either generate a series of add edge operations ('AE') to grow the graph to the desired density, then evaluate for the appropriate scenario. Alternatively, you can consider writing a data generator within Java to insert directly into the data structures. Whichever method you decide to use, remember to generate graphs of different densities to evaluate on. Due to the randomness of the data, you may wish to generate a few datasets with the same parameters settings (same graph density and a scenario) and take the average across a number of runs.

> Note, you may be generating and evaluating a significant number of datasets, hence we advise you to get started on this part relatively early.

4 Report Structure

As a guide, the report could contain the following sections:

- Explain your data generation and experimental setup. Things to include are (brief) explanations of the generated data you decide to evaluate on, the density parameters you tested on, describe how the scenarios were generated (a paragraph and perhaps a figure or high level pseudo code suffice), which approach(es) you decide to use for measuring the timing results, and briefly describe the fixed set(s) you used to generate the elements for vertex addition.
- Evaluation of the data structures using the generated data. Analyse, compare and discuss your results across different densities, representations and scenarios. Provide your explanation on why you think the results are as you observed. You may consider using the known theoretical time complexities of the operations of each data structure to help in your explanation.
- Summarise your analysis as recommendations, e.g., for this certain data scenario of this density, I recommend to use this data structure because... We suggest you refer to your previous analysis to help.

5 Submission

The final submission will consist of three parts:

- Your **Java source code** of your implementations. Your source code should be placed into in a flat structure, i.e., all the files should be in the same directory/folder, and that directory/folder should be named as `Assign1-<partner 1 student number>-<partner 2 student number>`.

Specifically, if your student numbers are s12345 and s67890, then all the source code files should be in folder Assign1-s12345-s67890.

- All folder (and files within) should be zipped up and named as `Assign1-<partner 1 student number>-<partner 2 student number>.zip`. E.g., if your student numbers are s12345 and s67890, then your submission file should be called `Assign1-s12345-s67890.zip`, and when we unzip that zip file, then all the submission files should be in the folder Assign1-s12345-s67890.
- Your **written report for part B** in PDF format, called "assign1.pdf". Place this pdf within the Java source file directory/folder, e.g., Assign1-s12345-s67890.
- Your **data generation code**. Create a sub-directory/sub-folder called "generation" within the Java source file directory/folder. Place your generation code within that folder. We will not run the code, but will examine their contents.
- Your group's **contribution sheet**. See the following 'Team Structure' section for more details. This sheet should also be placed within your source file folder.

Note: **submission of the report and code will be done via Canvas**. More detailed instructions will be provided closer to submission date.

6 Assessment

The project will be marked out of 15. Late submissions will incur a deduction of 1.5 marks per day, and no submissions will be accepted 7 days beyond the due date.

The assessment in this project will be broken down into two parts. The following criteria will be considered when allocating marks.

6.1 Implementation (8/15)

- You implementations will be assessed on whether they are adjacency matrix and incidence matrix, respectively, and on the number of tests it passes in our automated testing.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence these factors will contribute towards your marks.

6.1.1 Checkpoint (2 marks)

As part of the implementation work and to help you progress, you are asked to demo your implementation of the *Adjacency Matrix* representation and its operations as listed in task A during the lab session of week 5 (Lab 04). One member of the group should at least attend your allocated lab, where your Lab Assistant will assess your progress on the implementations. We will run the provided test script on some inputs to test if your *Adjancency Matrix* representation compiles, runs and passes a number of basic tests. This does not guarantee your code is bug free and will pass all of our final tests. Please conduct your own further testing.

6.2 Report (7/15)

The marking sheet in Appendix A outlines the criteria that will be used to guide the marking of your evaluation report. Use the criteria and the suggested report structure (Section 4) to inform you of how to write the report.

7 Team Structure

This project should be done in **pairs** (group of two). If you have difficulty in finding a partner, post on the discussion forum or contact your lecturer. If there are issues with work division and workload in your group, please contact your lecture as soon as possible.

In addition, please submit what percentage each partner made to the assignment (a contribution sheet will be made available for you to fill in), and submit this sheet in your submission. The contributions of your group should add up to 100%. If the contribution percentages are not 50-50, the partner with less than 50% will have their marks reduced. Let student A has contribution X%, and student B has contribution Y%, and $X > Y$. The group is given a group mark of M. Student A will get M for assignment 1, but student B will get $\frac{M}{\frac{X}{Y}}$.

8 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted project work in this subject is to be the work of you and your partner. It should not be shared with other groups. **Multiple automated similarity checking software will be used to compare submissions**. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the students concerned. Plagiarism of any form may result in zero marks being given for this assessment and result in disciplinary action.

For more details, please see the policy at `http://www1.rmit.edu.au/students/academic-integrity`.

9 Getting Help

There are multiple venues to get help. There are weekly consultation hours with the Lecturer and the Head Tutor(see Canvas for time and location details).

In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Assistant. We will also be posting common questions on the assignment's Q&A section in Canvas. You are encouraged to check and participate in the discussion forum on Canvas. However, please **refrain from posting solutions**.

A Marking Guide for the Report

Design of Evaluation (Maximum = 1.5 marks)	**Analysis of Results** (Maximum = 4 marks)	**Report Clarity and Structure** (Maximum = 1.5 marks)
1.5 marks Data generation is well designed, systematic and well explained. All suggested scenarios, data structures and a reasonable range of densities were evaluated. Each type of test was run over a number of runs and results were averaged.	4 marks Analysis is thorough and demonstrates understanding and critical analysis. Well-reasoned explanations and comparisons are provided for all the data structures, scenarios and densities. All analysis, comparisons and conclusions are supported by empirical evidence and possibly theoretical complexities. Well reasoned recommendations are given.	1.5 marks Very clear, well structured and accessible report, an undergraduate student can pick up the report and understand it with no difficulty.
1 marks Data generation is reasonably designed, systematic and explained. There are at least one obvious missing suggested scenarios, data structures or reasonable densities. Each type of test was run over a number of runs and results were averaged.	3 marks Analysis is reasonable and demonstrates good understanding and critical analysis. Adequate comparisons and explanations are made and illustrated with most of the suggested scenarios and densities. Most analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Reasonable recommendations are given.	1 marks Clear and structured for the most part, with a few unclear minor sections.
0.5 mark Data generation is somewhat adequately designed, systematic and explained. There are several obvious missing suggested scenarios, data structures or reasonable densities. Each type of test may only have been run once.	2 marks Analysis is adequate and demonstrates some understanding and critical analysis. Some explanations and comparisons are given and illustrated with one or two scenarios and densities. A portion of analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Adequate recommendations are given.	0.5 mark Generally clear and well structured, but there are notable gaps and/or unclear sections.
0 marks Data generation is poorly designed, systematic and explained. There are many obvious missing suggested scenarios, data structures or reasonable densities. Each type of test has only have been run once.	1 mark Analysis is poor and demonstrates minimal understanding and critical analysis. Few explanations or comparisons are made and illustrated with one scenario and density setting. Little analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Poor or no recommendations are given.	0 marks The report is unclear on the whole and the reader has to work hard to understand.