COSC 2123/1285 Algorithms and Analysis
Semester 1, 2019

Assignment 1: Closest Associates

Due date: 11:59pm Sunday, 14th of April, 2019
(Check Canvas and annoucements for latest due dates)
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

Facebook, Twitter and LinkedIn are some examples of social networks that we commonly use everyday. Indirect networks, such as email communications between people, can also reveal inherent relationships. There are many interesting questions that can be asked from these social-related networks, e.g., who are my closest friends, who is influential in spreading news (or gossip) and what are my relationship groups and who belongs to them. All these questions can be studied when we abstract and represent these social networks as graphs and perform analysis using graph operations and techniques from social network analysis and network science.

When we represent these networks as a graph, the vertices in such a graph represent people and edges can represent relationships or communications. There are many types of social-related networks, in this assignment, we are interested in communication graphs. In a communication graph, the edges are typically *directed* and represent one person communicating with another. They may also be weighted, representing the level of communication between two people and implicitly, how close or significant their relationship are. This can be determined via finding the k-nearest neighbours on such graphs, which is described in more details below.

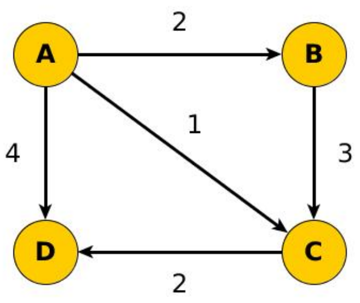
In class, we studied three methods to represent the graph, the adjacency list, adjacency matrix and vertex/edge list representations. There is a fourth type of representation called *incident matrix* (see below for details). The performance of each representation varies depending on the characteristics of the graph. In this assignment, we will implement the adjacency list and incident matrix representations, and evaluate on how well they perform when representing a communication graph. This will assist with your understanding of the tradeoffs between data structure representations and its effect when implementing operations or solving problems using algorithms and data structures.

2.1 Incidence Matrix Representation

The ***incidence matrix*** represents a graph as a set of vertices and list of edges incident to each vertex as a 2D array/matrix. Incidence matrices are not generally used to represent weighted graphs, but in this assignment we will use the following convention to do so. More formally, let the incidence matrix of a *directed* graph is an `n x m` matrix A with `n` and `m` are the number of vertices and edges of the graph respectively such that:

- $A_{i,k} = w_k$ (entry in this matrix) if edge e_k is a directed edge from v_i to v_j (meaning v_i is the source of the edge) and e_k has weight of w_k.
- $A_{j,k} = -w_k$ (entry in this matrix) if edge e_k is a directed edge from v_i to v_j (meaning v_j is the target of the edge) and e_k has weight of w_k.
- $A_{i,k} = 0$ for all other non-incident edges.

For example, the following graph:



has its incidence matrix as below:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{ccccc} AB & AC & AD & BC & CD \end{array} \\ \begin{pmatrix} 2 & 1 & 4 & 0 & 0 \\ -2 & 0 & 0 & 3 & 0 \\ 0 & -1 & 0 & -3 & 2 \\ 0 & 0 & -4 & 0 & -2 \end{pmatrix} \end{array}$$

Some extra resources about incidence matrix:

- `https://en.wikipedia.org/wiki/Incidence_matrix` (note we use the opposite sign convention to wikipedia for directed graphs)
- `http://mathonline.wikidot.com/incidence-matrices` (note in their directed graph example, there is a typo in columns 3 and 4 and column 7 doesn't need to exist)

2.2 K-nearest Neighbours

In class, we discussed the concept of a neighbours of a vertex – all the vertices that are incident to it. For directed graphs, recall there are two types, *in-neighbours* and *out-neighbours*. In-neighbours of a vertex v_j are all vertices that are connected (incident) to an edge coming into the vertex v_j. Out-neighbours of a vertex v_i are all vertices that are connected (incident) to an edge coming out from vertex v_i. Because the graphs are weighted, we can also have the concept of nearest in/out-neighbours, where we order all the in/out neighbours and choose the k ones with highest edge weights.

As an example, in the example above for incidence matrix, we have the following neighbours:

- In-neighbourhood of C is {A, B}.
- Out-neighbourhood of C is {D}.

- In-neighbourhood of A is { } (empty set or no in-neighbours).
- Out-neighbourhood of A is {D, C, B}.
- 1-nearest in-neighbourhood of C is {B}. ?
- 2-nearest out-neighbourhood of C is {D} (the number of neighbours less than k, so we return all).
- 2-nearest out-neighbourhood of A is {D, B}.

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 *directed*, *weighted* graph using the *adjacency list* and *incidence matrix* representations. Each representations will be implemented by a data structure. Your implementation should support the following operations:

- Create an empty directed graph (implemented as a constructor that takes zero arguments).
- Add a vertex to the graph.
- Add an edge to the graph.
- Get the weight of an edge in the graph.
- Update weight of edge in the graph.
- Delete a vertex from the graph.
- Compute the k-nearest in-neighbours of a vertex in the graph.
- Compute the k-nearest out-neighbours of a vertex in the graph.
- Print out the set of vertices of the graph.
- Print out the set of edges and their weights of 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 list, using an array of linked lists.
- Incidence matrix, using a 2D array (an array of arrays).

For the above data structures, you must program your own implementation, and not use the LinkedList or Matrix type of data structures in java.utils or any other libraries. You must implement your own nodes and methods to handle the operations. If you use java.utils or other implementation from libraries, this will be considered as an invalid implementation and attract 0 marks for that data structure. The only exception is if you choose to implement a map of vertex labels to a row or column index for the incidence matrix, you may use one of the existing Map classes to do this.

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, W, U, RV, IN, ON, PV, PE, 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> <weight>` – add an edge with source vertex 'srcLabel', target vertex 'tarLabel' and edge weight 'weight' into the graph.
- `W <srcLabel> <tarLabel>` – return weight of edge. If edge doesn't exist, return -1.
- `U <srcLabel> <tarLabel> <newWeight>` – Update the weight of edge ('srcLabel', 'tarLabel) to 'newWeight' value. If 'newWeight' = 0, then delete the edge.
- `RV <vertLabel>` – remove vertex 'vertLabel' from the graph.
- `IN <k> <vertLabel>` – Return a set of k nearest in-neighbours for vertex 'vertLabel'. The ordering of the neighbours does not matter. If k = -1, then all neighbours should be returned. See below for the required format.
- `ON <k> <vertLabel>` – Return a set of k nearest out-neighbours for vertex 'vertLabel'. The ordering of the neighbours does not matter. If k = -1, then all neighbours should be returned. See below for the required format.
- `PV` – prints the vertex set of the graph. See below for the required format. The vertices can be printed in any order.
- `PE` – prints the edge set of the graph. See below for the required format. The edges can be printed in any order.
- `Q` – quits the program.

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

```
A <(neighbour1,weight1) (neighbour2, weight2) ...>
```

Each neighbour has its associated edge weight printed with it, e.g, neighbour1 and weight 1. 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> <weight>
```

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 1
AE C B 1
AE B D 1
AE A E 3
AE D C 5
AE F A 2
ON −1 A
IN −1 F
W C B
W B C
W A D
U C B 4
U A B 0
RV D
AV G
PV
PE
Q
```

The output from the two neighbour operations (‘ON -1 A’, ‘IN -1 F’) should be:

```
A (B,2) (E,3)
F
```

The output from operations to retrieve edge weights (‘W C B’, ‘W B C’, ’W D C’) should be:

```
1
−1
5
```

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

```
A B C E F G
```

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

```
A E 3
C B 4
F A 2
```

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).

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.

file	description
`GraphEval.java`	Code that reads in operation commands from stdin then executes those on the selected graph implementation. Also will format the output as required. *No need to modify this file*.
`AssociationGraph.java`	Interface class for the graph representations. It contains the common interface/methods that you'll need to implement for the various representations. *No need to modify this file*.
`AbstractAssocGraph.java`	Abstract class, that you can use to implement common functionality among the representations.
`AdjList.java`	Code that implements the adjacency list implementation of a graph. Complete the implementation (implement parts labelled "Implement me!").
`IncidenceMatrix.java`	Code that implements the incidence matrix implementation of a graph. Complete the implementation (implement parts labelled "Implement me!").
`MyPair.java`	Code that implements a pair class, as javafx is not available on the core teaching server. *No need to modify this file*.

Table 1: Table of Java files.

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 GraphEval.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 GraphEval.java.

Instructions on how the python script runs are available within the header of the script, on Canvas and discussed in lectures.

Notes

- We will run the supplied test script on your implementation on the university's core teaching servers. 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 1.8**.

Test Data

For the next part, we provide an email graph of about 2000 vertices in a file called "assocGraph.csv". This is real, network of the amount of emial correspondence between people but to protect privacy we do not state what organisation nor have the people names.

Task B: Evaluate your Data Structures (7 marks)

In this second task, you will evaluate your two 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/communication network.

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 **8 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[1] on these scenarios. There are many possibilities, but for this assignment, consider the following scenarios:

Scenario 1 Shrinking graph (Removals): Although not often occurring for association graph, people do defriend/de-associate with each other. In this scenario, you are to evaluate the performance of your implementations in terms of:

- vertex removal
- edge removal

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.

Scenario 2 Nearest Neighbours: In this scenario, the graph is not changing, but important operations such as neighbourhood are requested.

Assume the graph that you start with is the one that we provided you with. You are to evaluate the performance of the the nearest neighbourhood implementations, for both in and out neighbourhoods, as the density of the initial graph and k are varied.

Scenario 3 Changing associations (Update edge weights): In this scenario, associates between people are fluctuating and the corresponding edge weights are changing. In this scenario, you are to evaluate the performance of your implementations in terms of:

- edge weight changes, both increases and decreases (but never enough to cause a weight to be 0 or less and subsequently be deleted).

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

Data Generation

When generating the vertices and edges to remove, find neighbourhood and update edge weights 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.,

[1]Density $= \frac{\text{number of edges}}{\text{number of vertices}^2}$

0 to max vertex index of your graph when generating the vertices and edges for removing, nearest neighbourhoods and updating weights, and a different range when adding vertices (we do not want to repeatingly add vertices that are in the graph already).

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 from the one we supplied, then evaluate for the appropriate scenario. Alternatively, you can consider writing a data generator within Java to insert directly into the data structures. Or can use one of the many great graph generators[2] 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.

Analysis

In your analysis, you should evaluate each of your representations and data structures in terms of the different scenarios outlined above.

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.

[2]See https://networkx.github.io/documentation/stable/reference/generators.html, under heading Random Graphs, select one of them. Note this is in Python.

- 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.

Implementation (8/15):

- You implementation will be assessed on whether they are adjacency lists and incidence matrices, 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.

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 (see Canvas for time and location details). In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Demonstrator. We will also be posting common questions on the project 1 Q&A section on Canvas and we encourage you 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.