# Generics Design

A screenshot of a computer

Description automatically generated

Worth noting decoupling

# Graph Representations List

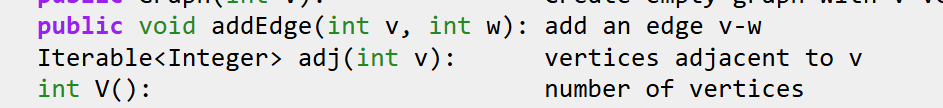
Integer Vertices

Make a ( dict ) Map and pair IDs with string Labels

Adjacency Matrix

Edge Sets

## Integer Vertices



This Iterable<Integer> behaves like a generator

Look

A screenshot of a computer program

Description automatically generated

## Adj Matrix

A screenshot of a computer

Description automatically generated

## Edge Sets

A screenshot of a computer

Description automatically generated

Prob too slow, not worth trying

## Adjacency Lists – most popular for sparse graphs

A screenshot of a computer

Description automatically generated

## Not sparse graph?

Buckets?

# Bones

Base API

Specific Implementation – adj matrix, adj list

Storage type

Add Edge, View Adj List, Subtract Edge

Algorithms

Print out a traversal

isConnected?

Shortest Paths

Test Cases

# Heap

Watch the project videos – nvm I got the basic idea

Check the Software Engineering lectures for notes on Design Documents

Once I know what the design document should look like, I’ll go do it and the checkpoint

Done

Read **requirements more carefully and take notes**

Do the checkpoint

Then start testing out reading in data

Complete the Graph merge

And then the small details on the design document

# Bucket

Princeton Implementations were not too informative…

Baeldung Implementation done

Create a Adjacency Matrix version of a graph

It would be ideal but I don’t know if I have time anymore – I still have algorithms to test out and there’s the actual project

Next step should be to merge the Graphs into one comprehensive one, and implement the changes listed below

Possibly far better version:

<https://www.baeldung.com/java-graphs>

# Design Thoughts

## Structure

GraphBase extended by Storage Method

Adj List

Adj List that stores LinkedList of Nodes ( for if Edge is bidirectional )

Consider if each should be a node

Adj Matrix

GraphBase interface

addVertex

removeVertex

addEdge

removeEdge

Track neighbors – done as adjacency list

Soon: track bidirectional edge

Optional: visualize

Optional: output as CSV file

Optional: track unconnected vertices

Track whatever would be slow to Find via an algorithm, I think

Algorithms:

Track or Find sinks and sources

Find traversals : depth first, etc

Find shortest path from A to B node

Future: convert to minimum spanning tree?