# Abstract

# Introduction

# Applications/Graph Theory

# Algorithms

# Implementation

The java implementation has been constructed using object-oriented programming (OOP). The variables for each class are private with setter and getters to incorporate encapsulation.

The first stage of the implementation was to create an edge weighted graph as a framework for the program. An edge weighted graph consists of two main components which are represented in the java implementation by an Edge class and a Vertex class. These classes create the foundation for the Edge Weighted Graph class that can theoretically convey any graph. In order to traverse the edge weighted graph both classes need to retain information which pertains to the other class.

A vertex or node is a point within the graph that is often labelled. These can be connected to each using edges. These ideas translate into the following variable:

* String label – Allow for the vertex to be labelled
* HashSet<Edge> edges - A set of all edges that are attached to this vertex

Each new Vertex class is instantiated with a String variable as an argument. The constructor uses the String as the label for the class then creates a new HashSet<Edge>. A string has been used for the label to allow any character to denote the vertex. The HashSet<Edge> has been implemented as it is a dynamic data structure that will prevent duplicates of each vertex classes in the set. An addition benefit of this data structure it that the search time for an Edge will be constant. This will help reduce the time taken for each traversal which is important as the method that will be used to check for loops when generating a new hypothesis and within Kruskal’s algorithm is the depth first traversal.

An edge is used to connect to vertices within a graph. For an edge weighted graph the edge will have a weight linked to itself. These concepts translate into the following variables:

* Vertex Vertex1 – A vertex class which the edge is connected to
* Vertex Vertex2 – The other vertex class which the edge is connected to
* double Weight – A double that represents the weight attached to the edge

An array has been used to store the two Vertex classes as the maximum number of vertex is predetermined to be two.

This information is passed to the class each time a new Edge class is instantiated.

The class constructor has been overloaded to allow for two constructors. This allows for the representation of regular graphs and for weighted graphs. As such the first constructor takes two Vertex classes and sets the weight to 0 to express a regular graph. The second constructor takes two Vertex classes and a numerical value (double) for the weight to express the weighted graph.

A private method named initialize() has been created which takes two vertex classes as an argument. The method calls the add() method of each vertex class given to add the newly created edge (this) to the edge sets of each vertex class. This method has been created private to enforce encapsulation as the initialize() method is only required within the edge class.

The Edge class has been given a special getter method to help with the traversal of the graph. The method is called getOther() and takes a vertex as an argument. The method will then check if the vertex is connected to the node. If the vertex is connected to the node, then method will return the vertex that is connected to the over side of the edge. If the vertex is not connected to the node then with method returns null.

For the following example the vertex class variables will be named V1 and V2 respectively:

## (Code)

# Testing and Errors

Test Driven Development (TDD)

Requirements for Edge Weighed Graph

* The ability to construct regular and edge weighted graphs
* The graph should not allow vertices to be connected to themselves (no self-loops)
* There must be no duplicate connections between vertices
* The label of each vertex must be unique

Requirements for Edges:

* Must be connected on both sides as a vertex cannot be connected to nothing
* The graph will not contain any vertices that self-loop
* The edge must be able to return both of the vertices connected for traversal

Requirements for Vertex:

* The ability to add and remove edges to allow for construction of graphs
* Each vertex must know how many edges are connected to it at any given time
* Each vertex must be able to return specific edges for traversal

Error when preventing duplicate edges caused by vector.equals(otherVector) not recognising when the vectors are equal. This was due to the creation of a new vertex within the test. The fix was to check the labels (String) variables of each vector against each other. This has overall made the program more robust as it will now be able to recognise if newly created vertex and edge classes carry the same information as an old class.

# Methodology

Waterfall

# Research

# Results

# Future Development

# Conclusion