Random Graphs

1.0

Generated by Doxygen 1.8.18

Chapter 1

Random Graphs API

1.1 Introduction

In this project, I create highly re-usable classes using class templates to build random graphs, which have the ability to store integers at the nodes, which themselves are labelled by integers. Additionally, the edges are weighted by randomly generated integers according to user-supplied range. Here, two components are random:

```
-# Graph Structure: Nodes are connected arbitarily.
-# Edge Weights: Edges are weighted arbitarily.
```

The Node, Edge, and Graph class templates provide high reusability by allowing to store a variety of data types.

1.2 Miscellaneous Notes

1.2.1 Class Hierarchy

A single Edge is composed of two Node objects. A RandomEdge object extends an Edge object. A Graph object is composed of zero or more Edge objects. Finally, a RandomGraph object extends a Graph object.

1.2.2 Template Instantiation

At the time of writing the implementation files, the compiler I used did not allow implicit instantiation of class templates. Therefore, at the end of several .cpp files, I explicitly instantiated the most common combinations of classes. Of course, this is far from ideal of a solution. It might also slow down compilation. You may uncomment those parts if you plan to re-use the code. Moreover, it may be possible to live without these lines if your compiler allows implicit instantiation.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Class Documentation

4.1 Edge< Weight, Label, Data > Class Template Reference

```
#include <Edge.h>
```

Public Member Functions

- Edge ()
- Edge (Weight edge_weight, Node< Label, Data > *end1, Node< Label, Data > *end2)
- virtual ~Edge ()
- void set_weight (Weight edge_weight)
- void reset_end (Node< Label, Data > *node, bool first_endpoint)
- Weight get_weight ()
- Node< Label, Data > * get_end (bool first_endpoint)

Protected Attributes

- Weight **weight**
- Node< Label, Data > * **first_endpoint**
- Node< Label, Data > * **second_endpoint**

4.1.1 Detailed Description

template<class Weight, class Label, class Data>
class Edge< Weight, Label, Data >

A weighted edge in a graph

Template Parameters

Weight	the class of the edge weight
Label	the class of an edge's node label
Data	the class of an edge's node data

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Edge() [1/2]

```
template<class Weight , class Label , class Data >
Edge< Weight, Label, Data >::Edge
```

Default Constructor

Generates an Edge with NULL Node pointers. The weight is set to 0.

Implementation of a weighted graph edge

Author

Ali Al-Musawi

4.1.2.2 Edge() [2/2]

```
template<class Weight , class Label , class Data >
Edge< Weight, Label, Data >::Edge (
             Weight edge_weight,
             Node< Label, Data > * end1,
             Node< Label, Data > * end2 )
```

Class Constructor

Parameters

edge_weight	the weight of this edge
end1	a pointer to a node on one end
end2	a pointer to a node on the other end

4.1.2.3 ∼Edge()

```
template<class Weight , class Label , class Data >
Edge< Weight, Label, Data >::∼Edge  [virtual]
```

Class Destructor

4.1.3 Member Function Documentation

4.1.3.1 get_end()

```
template<class Weight , class Label , class Data >
Node< Label, Data > * Edge< Weight, Label, Data >::get_end (
            bool first_endpoint )
```

Retrieves a node from one end of this edge

Parameters

first_endpoint	true if the node wanted is the first endpoint, otherwise the second endpoint is returned

Returns

a pointer to the node specified on this edge

4.1.3.2 get_weight()

```
template<class Weight , class Label , class Data >
Weight Edge< Weight, Label, Data >::get_weight
```

Retrieves the weight of this edge

Returns

the weight of this edge

4.1.3.3 reset_end()

```
template<class Weight , class Label , class Data >
void Edge< Weight, Label, Data >::reset_end (
            Node< Label, Data > * node,
            bool first_endpoint )
```

Sets a node to one end of this edge

Parameters

node	a reference to the Node to append to this edge
first_endpoint	true if node replaces the first_endpoint and false if the node replaces the second_endpoint

4.1.3.4 set_weight()

```
template<class Weight , class Label , class Data >
void Edge< Weight, Label, Data >::set_weight (
             Weight edge_weight )
```

Sets the weight of this edge

Parameters

edge_weight	the weight to set for this edge

The documentation for this class was generated from the following files:

- Edge.h
- Edge.cpp

4.2 Graph< Weight, Label, Data > Class Template Reference

```
#include <Graph.h>
```

Public Member Functions

- Graph ()
- virtual ∼Graph ()
- Node< Label, Data > ∗ get_node (Label node_label)
- std::vector< Node< Label, Data > ∗ > list_nodes ()
- unsigned int num_nodes ()
- Edge< Weight, Label, Data > ∗ get_edge (Node< Label, Data > ∗from, Node< Label, Data > ∗to)
- unsigned int num_edges ()
- bool is_adjacent (Node< Label, Data > ∗from, Node< Label, Data > ∗to)
- std::vector< Node< Label, Data > ∗ > neighbours (Node< Label, Data > ∗node)
- void print_graph ()
- void insert_node (Node< Label, Data > ∗node)
- void insert_edge (Edge< Weight, Label, Data > ∗edge)
- void delete_edge (Edge< Weight, Label, Data > ∗edge)

Protected Attributes

- unsigned int **nodes**
- unsigned int **edges**
- std::vector< Node< Label, Data > ∗ > **node_list**
- std::vector< std::vector< Edge< Weight, Label, Data > ∗ > > **adjacency_list**

Friends

- template<class W , class L , class D >
  bool node_exists (Node< L, D > ∗node, Graph< W, L, D > ∗g)
- template<class W , class L , class D >
  bool edge_exists (Node< L, D > ∗from, Node< L, D > ∗to, Graph< W, L, D > ∗g)

4.2.1 Detailed Description

template<class Weight, class Label, class Data>
class Graph< Weight, Label, Data >

A weighted graph

Template Parameters

Weight	the class of the Graph's Edge weight
Label	the class of the Graph's Node label
Data	the class of the Graph's Node data

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Graph()

```
template<class Weight , class Label , class Data >
Graph< Weight, Label, Data >::Graph
```

Default Constructor

Generates an empty graph

Implementation of a weighted graph

Author

Ali Al-Musawi

4.2.2.2 ∼Graph()

```
template<class Weight , class Label , class Data >
Graph< Weight, Label, Data >::∼Graph  [virtual]
```

Class Destructor

4.2.3 Member Function Documentation

4.2.3.1 delete_edge()

```
template<class Weight , class Label , class Data >
void Graph< Weight, Label, Data >::delete_edge (
             Edge< Weight, Label, Data > * edge )
```

Delete an Edge from this Graph

If the supplies Edge is not in this Graph, nothing changes.

Parameters

edge	a pointer to Edge of the Edge to delete

4.2.3.2 get_edge()

```
template<class Weight , class Label , class Data >
Edge< Weight, Label, Data > * Graph< Weight, Label, Data >::get_edge (
            Node< Label, Data > * from,
            Node< Label, Data > * to )
```

Retrieves an Edge in this Graph

Parameters

from	a pointer to a Node in this Graph
to	a pointer to a Node in this Graph

Returns

an Edge in this Graph connecting the supplied Nodes

Exceptions

GraphIllegalAccessException	if the Edge is not in the Graph

4.2.3.3 get_node()

```
template<class Weight , class Label , class Data >
Node< Label, Data > * Graph< Weight, Label, Data >::get_node (
            Label node_label )
```

Retrieve a Node from this Graph

Parameters

node_label	the label of the Node to get

Returns

a pointer to the Node

Exceptions

GraphIllegalAccessException	if the Node does not exist

4.2.3.4 insert_edge()

```
template<class Weight , class Label , class Data >
void Graph< Weight, Label, Data >::insert_edge (
             Edge< Weight, Label, Data > * edge )
```

Add an Edge to this Graph

Replaces an existing Edge if the added Edge's endpoints are already connected

Parameters

edge	a pointer to Edge of the inserted Edge

Exceptions

GraphInvalidInsertionException	if at least one Node on this Edge does not exist in the Graph

4.2.3.5 insert_node()

```
template<class Weight , class Label , class Data >
void Graph< Weight, Label, Data >::insert_node (
             Node< Label, Data > * node )
```

Adds a Node to this Graph

Replaces an existing Node if the Label is the same

Parameters

node	a pointer to Node of the inserted Node

4.2.3.6 is_adjacent()

```
template<class Weight , class Label , class Data >
bool Graph< Weight, Label, Data >::is_adjacent (
             Node< Label, Data > * from,
             Node< Label, Data > * to )
```

Checks if 2 Nodes are connected by an Edge in this Graph

Parameters

from	a pointer to Node in this Graph
to	a pointer to Node in this Graph

Returns

true if there is an Edge connecting the Nodes in this Graph

4.2.3.7 list_nodes()

```
template<class Weight , class Label , class Data >
std::vector< Node< Label, Data > * > Graph< Weight, Label, Data >::list_nodes
```

Retrieves a list of all Nodes in this Graph

Returns

a Vector of pointers to Node for all Nodes in this Graph

4.2.3.8 neighbours()

```
template<class Weight , class Label , class Data >
std::vector< Node< Label, Data > * > Graph< Weight, Label, Data >::neighbours (
            Node< Label, Data > * node )
```

Lists all Nodes that are neighbours of the given Node

Parameters

node	a pointer to Node whose neighbours are wanted

Returns

a Vector of pointers to Node neighbouring the given Node

Exceptions

GraphIllegalAccessException	if the given Node is not in the Graph

4.2.3.9 num_edges()

```
template<class Weight , class Label , class Data >
unsigned int Graph< Weight, Label, Data >::num_edges
```

The number of Edges in this Graph

Returns

the number of Edges in this Graph

4.2.3.10 num_nodes()

```
template<class Weight , class Label , class Data >
unsigned int Graph< Weight, Label, Data >::num_nodes
```

The size of this Graph

Returns

the number of Nodes in this Graph

4.2.3.11 print_graph()

```
template<class Weight , class Label , class Data >
void Graph< Weight, Label, Data >::print_graph
```

A text representation of the graph

Displays the adjacency matrix

4.2.4 Friends And Related Function Documentation

4.2.4.1 edge_exists

```
template<class Weight , class Label , class Data >
template<class W , class L , class D >
bool edge_exists (
            Node< L, D > * from,
            Node< L, D > * to,
            Graph< W, L, D > * g )  [friend]
```

Helper Method: Indicates whether an edge exists in a graph

Parameters

from	a pointer to Node forming one end point of the Edge to look for
to	a pointer to Node forming another end point of the Edge to look for
graph	a pointer to Graph to search in

Returns

true if edge is in this graph

4.2.4.2 node_exists

```
template<class Weight , class Label , class Data >
template<class W , class L , class D >
bool node_exists (
            Node< L, D > * node,
            Graph< W, L, D > * g )  [friend]
```

Helper Method: Indicates whether a node exists in a graph

Parameters

n	a pointer to Node to search for in this graph
graph	a pointer to Graph to search in

Returns

true if node is in this graph

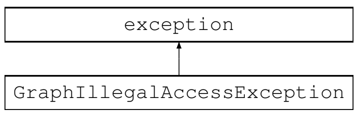
The documentation for this class was generated from the following files:

- Graph.h
- Graph.cpp

4.3 GraphIllegalAccessException Class Reference

```
#include <GraphIllegalAccessException.h>
```

Inheritance diagram for GraphIllegalAccessException:



Public Member Functions

- GraphIllegalAccessException ()
- virtual const char ∗ what () const throw ()

4.3.1 Detailed Description

An Edge Exception Class

Thrown if a Graph is accessed through non-existent Edge/Node

4.3.2 Constructor & Destructor Documentation

4.3.2.1 GraphIllegalAccessException()

```
GraphIllegalAccessException::GraphIllegalAccessException ( )
```

Default Constructor

4.3.3 Member Function Documentation

4.3.3.1 what()

```
const char * GraphIllegalAccessException::what ( ) const throw ( )   [virtual]
```

Generates Exception message

Returns

the message of the exception

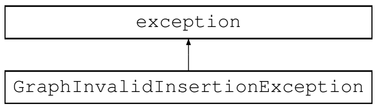
The documentation for this class was generated from the following files:

- GraphIllegalAccessException.h
- GraphIllegalAccessException.cpp

4.4 GraphInvalidInsertionException Class Reference

```
#include <GraphInvalidInsertionException.h>
```

Inheritance diagram for GraphInvalidInsertionException:



Public Member Functions

- GraphInvalidInsertionException ()
- virtual const char ∗ what () const throw ()

4.4.1 Detailed Description

An Edge Exception Class

Thrown if an Edge is inserted into a Graph and at least 1 Node on the Edge is not in the Graph

4.4.2 Constructor & Destructor Documentation

4.4.2.1 GraphInvalidInsertionException()

```
GraphInvalidInsertionException::GraphInvalidInsertionException ( )
```

Default Constructor

4.4.3 Member Function Documentation

4.4.3.1 what()

```
const char * GraphInvalidInsertionException::what ( ) const throw ( )   [virtual]
```

Generates Exception message

Returns

the message of the exception

The documentation for this class was generated from the following files:

- GraphInvalidInsertionException.h
- GraphInvalidInsertionException.cpp

4.5 Node< Label, Data > Class Template Reference

```
#include <Node.h>
```

Public Member Functions

- Node ()
- Node (Label node_label, Data node_data)
- virtual ~Node ()
- void set_label (Label node_label)
- void set_data (Data node_data)
- Label get_label ()
- Data get_data ()
- void print_node ()

4.5.1 Detailed Description

template<class Label, class Data>
class Node< Label, Data >

A labelled node in graph that stores data

Template Parameters

Label	the class of the node label
Data	the class of the node data

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Node() [1/2]

```
template<class Label , class Data >
Node< Label, Data >::Node
```

Default Constructor

The member fields are initialized to default value.

Implementation of a graph node

Author

Ali Al-Musawi

4.5.2.2 Node() [2/2]

```
template<class Label , class Data >
Node< Label, Data >::Node (
             Label node_label,
             Data node_data )
```

Class Constructor

Parameters

node_label	the name of the node
node_data	the data to store in the node

4.5.2.3 ∼Node()

```
template<class Label , class Data >
Node< Label, Data >::∼Node  [virtual]
```

Class Destructor

4.5.3 Member Function Documentation

4.5.3.1 get_data()

```
template<class Label , class Data >
Data Node< Label, Data >::get_data
```

A method to retrieve the node data

Returns

the data stored in this node

4.5.3.2 get_label()

```
template<class Label , class Data >
Label Node< Label, Data >::get_label
```

A method to retrieve the node label

Returns

the label of this node

4.5.3.3 print_node()

```
template<class Label , class Data >
void Node< Label, Data >::print_node
```

A text representation of the Node

4.5.3.4 set_data()

```
template<class Label , class Data >
void Node< Label, Data >::set_data (
             Data node_data )
```

Changes the data stored in the node

Parameters

node_data	the data to store in the node

4.5.3.5 set_label()

```
template<class Label , class Data >
void Node< Label, Data >::set_label (
             Label node_label )
```

Changes the label of the node

Parameters

node_label	the new label of the node

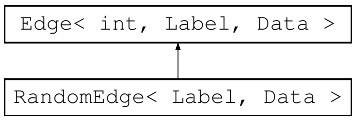
The documentation for this class was generated from the following files:

- Node.h
- Node.cpp

4.6 RandomEdge< Label, Data > Class Template Reference

```
#include <RandomEdge.h>
```

Inheritance diagram for RandomEdge< Label, Data >:



Public Member Functions

- RandomEdge ()
- RandomEdge (Node< Label, Data > ∗end1, Node< Label, Data > ∗end2, int min_weight, int max_weight, unsigned int seed)
- virtual ∼RandomEdge ()
- void set_random_weight (unsigned int seed)

Protected Attributes

- int **min_weight**
- int **max_weight**
- unsigned int **seed**

4.6.1 Detailed Description

template<class Label, class Data>
class RandomEdge< Label, Data >

A derived specialized template class

Edge weights are sampled from a distribution

4.6.2 Constructor & Destructor Documentation

4.6.2.1 RandomEdge() [1/2]

```
template<class Label , class Data >
RandomEdge< Label, Data >::RandomEdge
```

Default Constructor

Generates an edge randomly. The range of the weights is within the supported integers.

Implementation of a random weighted graph edge

Author

Ali Al-Musawi

4.6.2.2 RandomEdge() [2/2]

```
template<class Label , class Data >
RandomEdge< Label, Data >::RandomEdge (
            Node< Label, Data > * end1,
            Node< Label, Data > * end2,
            int min_weight,
            int max_weight,
            unsigned int seed )
```

Class Constructor

Parameters

end1	a pointer to a node on one end
end2	a pointer to a node on the other end
min_weight	a minimum bound on the generated weight
max_weight	a maximum bound on the generated weight
seed	a seed for reproducibility

4.6.2.3 ∼RandomEdge()

```
template<class Label , class Data >
RandomEdge< Label, Data >::∼RandomEdge  [virtual]
```

Class Destructors

4.6.3 Member Function Documentation

4.6.3.1 set_random_weight()

```
template<class Label , class Data >
void RandomEdge< Label, Data >::set_random_weight (
             unsigned int seed )
```

Resets the weight of this edge randomly

The range of the weight is established during construction

Parameters

seed	a seed for reproducibility

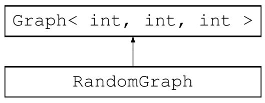
The documentation for this class was generated from the following files:

- RandomEdge.h
- RandomEdge.cpp

4.7 RandomGraph Class Reference

```
#include <RandomGraph.h>
```

Inheritance diagram for RandomGraph:



Public Member Functions

- RandomGraph ()
- RandomGraph (unsigned int size, double density, int min_weight, int max_weight, bool directed, bool loops)
- virtual ∼RandomGraph ()

Public Attributes

- const unsigned int **DEFAULT_NODES** = 50

Protected Attributes

- double **density**

4.7.1 Detailed Description

A derived specialized template class

Edges are randomly generated according to a density between 0 and 1. If density = 1, a clique is generated. If density = 0, no edge is in the graph. Nodes are labelled sequentially by integers, and their data are initialized to MAX_INT in <limits>.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 RandomGraph() `[1/2]`

```
RandomGraph::RandomGraph ( )
```

Default Constructor

Generates an undirected graph with the minimum (50) number of nodes, sets the graph density randomly, and the edge weights range between 0 and MAX_INT in <limits>

4.7.2.2 RandomGraph() `[2/2]`

```
RandomGraph::RandomGraph (
             unsigned int size,
             double density,
             int min_weight,
             int max_weight,
             bool directed,
             bool loops )
```

Class Constructor

Parameters

size	the number of nodes
density	the ratio of the number of edges to the maximum possible number of edges in a simple graph
min_weight	the minimum weight of an edge
max_weight	the maximum weight of an edge
directed	true if the generated graph has directed edges
loops	true if the generated graph can contain self-loops

4.7.2.3 ∼RandomGraph()

```
RandomGraph::∼RandomGraph ( )  [virtual]
```

Class Destructor

The documentation for this class was generated from the following files:

- RandomGraph.h
- RandomGraph.cpp