Requirements:

In this project, you will implement one class implementing the algorithm:

`Dijkstra's algorithm`

A skeleton header file for this class and its associated class is provided. You may modify these files for the project, though you must **add your own comments**. Do not copy text from the specifications. Testing tools for some of the classes provided in the associated testing directory.

You are required to submit the following header file(s):

- `Weighted_graph.h`

You may add any other header files (of your own authorship) you need for your project to work.

You may use all standard libraries available. This includes the Standard Template Library (STL), `cstring`, etc.

Runtime

Ten percent of the grade will be attributed to run-time relative to other students in your class. The class average and standard deviation will be calculated (with outliers removed), and those achieving the class average will get 80%. Other grades will be based on 10 marks per standard deviation in either direction.

Memory usage

There are no constraints on memory use in this project.

sample test

The sample test code must be such that passing the sample test code will guarantee a grade of 85 % on the project. That is, it will test that member functions are named correctly, and that the basic functionality is present. Students are expected to do their own testing, though testing tools and some minimal tests will be provided.

What You May Do

You may:

- Make multiple submissions—only the last is kept.
- Declare and define other classes and other member functions not specified in the project description; however, if these appear in other files, those files must be included with your submission.
- Reuse classes from previous projects; however, you must include those files with your submission.
- If you reuse previous projects, you are, of course, allowed to make corrections to those projects.
- You may even, though you must be very cautious, overload the given member functions or add additional parameters to those member functions so long as those extra parameters have default values and the functions as specified in the project may be called with no issues or ambiguities.

Compiler and Testing

Source code which compiles using a Windows IDE but does not compile under `g++` will receive a mark of 0.

Late Projects

If you were late for your submission, you can still submit the project two days after the deadline. You will receive a maximum grade of 90% in the project. Suppose you are an engineer working on a project that has specified deadline with penalties in place if you are late. Is the party contracting with you going to relieve you of your obligations under the contract? In the same way, you should consider these projects your contractual obligations with this course.

Purpose

The purpose of the projects is to help you learn the course material and to help you begin to implement your own personal library of tools. Many of the subsequent projects will rely on previous ones (for example, you may be asked to specifically use your linked list classes to implement more complex data structures).

Program Documentation and Style

The following programming style is required for all projects.

Your name and ID must appear at the top of all files which you have created or modified.

Write clear and understandable code. Improve the clarity of your code by using vertical and horizontal spacing, meaningful variable names, proper indentation and comments.

Precede each function with comments indicating:

- What it does
- What each parameter is used for
- Assumptions that it makes
- How it handles errors

Interface

You are allowed to add whatever private or public class functions you feel are necessary.

TERM PROJECT:

Dijkstra's algorithm

You may use all standard libraries available. This includes the Standard Template Library (STL), `cstring`, etc. However, if you want to compete for the fastest implementations that get a prize, you should restrict yourself to using operations like `std::swap` and other reasonably straight-forward features. Do not use any of the containers if you want to compete.

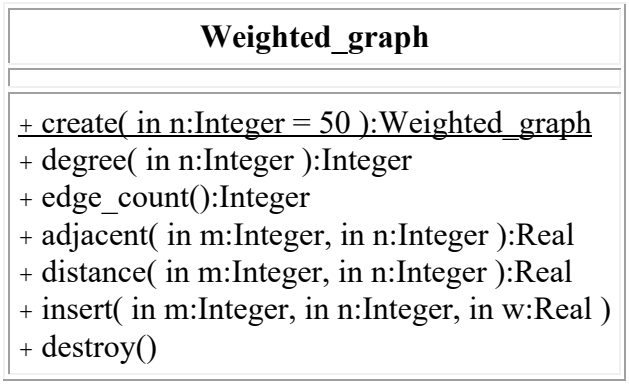
Requirements

In this sub-project, you will implement one class:

1. A weighted graph: `Weighted_graph`.

Class Specification

UML Class Diagram



Description

This class allows the user to create and destroy an undirected weighted graph. You will be able to find the shortest distance between two connected vertices. The vertices are numbered 0 through $n - 1$ where n is the argument to the constructor.

Member Variables

You may define whatever member variables you wish.

Member Functions

Constructors

```
Weighted_graph( int n = 50 )
```

Construct an undirected graph with n vertices (by default, 50). If $n \leq 0$, use $n = 1$.

Destructor

Clean up any allocated memory.

Accessors

This class has three accessors:

`int degree( int n ) const`
: Returns the degree of the vertex *n*. Throw an illegal argument exception if the argument does not correspond to an existing vertex. (**O**(1))

`int edge_count() const`
: Returns the number of edges in the graph. (**O**(1))

`double adjacent( int m, int n ) const`
: Returns the weight of the edge connecting vertices *m* and *n*. If the vertices are the same, return 0. If the vertices are not adjacent, return infinity. Throw an illegal argument exception if the arguments do not correspond to existing vertices.

Mutators

This class has two mutators:

`void insert( int m, int n, double w )`
: If the weight $w \leq 0$ if it is infinity, throw an illegal argument exception. If the weight $w > 0$, add an edge between vertices *m* and *n*. If an edge already exists, replace the weight of the edge with the new weight. If the vertices do not exist or are equal, throw an illegal argument exception.

`double distance( int m, int n )`
: Return the shortest distance between vertices *m* and *n*. Throw an illegal argument exception if the arguments do not correspond to existing vertices. The distance between a vertex and itself is 0.0. The distance between vertices that are not connected is infinity.