Graph and Search Library Design Document

Andrew Grant
amg2215@columbia.edu

Anton Igorevich
ain2108@columbia.edu

Somya Vasudevan
sv2500@columbia.edu

4/28/2017

1 Repository Information

The complete project is located at `https://github.com/andyg7/Graph-Library`. The core graph library code is located at `https://github.com/andyg7/Graph-Library/src`

2 Introduction

This is a graph library, built in C++, that makes it easy to create and use graphs. The library is designed to make it really easy for users to use their own vertex and edge data structures. The user will define his/her Vertex and Edge types and then will immediately be able to start using the library and running algorithms on graphs. Currently the library supports the following graphs: DG (directed graph), DAG (directed acyclic graph), DT (directed tree), Matrix(undirected). The idea is that users define their own vertex and edge data types, and then the library handles everything else under the hood.

Conceptually, a graph is made up of a bunch of vertices and edges. At a minimum there must be some way to distinguish between vertices, distinguish between edges, and define edges as made up of two vertices. Nontheless, users often want to embed extra information in these ADTs. For example, a user may want a graph representing cities and the highways between them. The user may have a City class; cities must of course have some unique id (e.g. city name), but they may also have extra information such as population, GDP, etc. The same goes for edges; maybe a Road class is used, and the class also has miles, age of road etc. This library makes it easy for users to provide their own ADTs, and immediately start creating graphs, and run algoritms on them.

3 Concepts

Reference: `http://www.stroustrup.com/good_concepts.pdf`

Concepts are essentially compile time predicates. That is, they are requirements on the types that are passed into functions. If an argument doesn't satisfy the concept, the compiler will immediately report an error. This makes debugging much easier when using templates. Without concepts, debugging can be very tricky when using templates, as it's often late in the compilation process that the compiler realizes a type is no good. As a result error messages can be extremely long, making dubugging tricky.

In terms of this library, the idea is that user defined vertices and edges must satisfy certain properties; concepts are used to enforce these. For example, an Edge must point to two vertices; an Edge must be comparable; a Vertex must have a unique identifier. We also use concepts to support function overloading. The names of the functions that work on graphs are all the same; so the user can call the same function whether he or she is working with a matrix or adjacency list. Concepts are used to determine which function should be called.

4 Conclusion