Boost.Graph tutorial

Richel Bilderbeek

December 4, 2015

1 Introduction

1.1 Coding style used

I prefer not to use the keyword auto, but to explicitly mention the type instead. I think this is beneficial to beginners. When using Boost.Graph in production code, I do prefer to use auto.

2 Creating graphs

Boost.Graph is about creating graphs. In this chapter we create graphs, starting from simple to more complex.

2.1 Creating an empty graph

Let's create a trivial empty graph:

Algorithm 1 Creating an empty graph

```
#include "create_empty_graph.h"

boost::adjacency_list<>
create_empty_graph() noexcept
{
  return boost::adjacency_list<>();
}
```

Congratulations, you've just created a boost::adjacency_list in which:

- The out edges are stored in a std::vector
- The vertices are stored in a std::vector
- The graph is directed

- Vertices, edges and graph have no properties
- Edges are stored in a std::list

The boost::adjacency_list is the most commonly used graph type, the other is the boost::adjacency_matrix.

2.2 Creating K_2, a fully connected graph with two vertices

To create a fully connected graph with two vertices (also called K_2), one needs two vertices and one (undirected) edge, as depicted in figure 1.

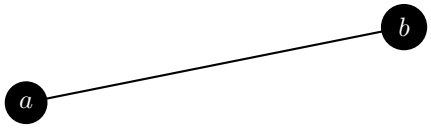

Figure 1: K_2: a fully connected graph with two vertices named a and b

To create K_2, the following code can be used:

```
#include "create_k2_graph.h"

boost::adjacency_list<
  boost::vecS,
  boost::vecS,
  boost::undirectedS
>
create_k2_graph() noexcept
{
  using graph = boost::adjacency_list<
    boost::vecS,
    boost::vecS,
    boost::undirectedS
  >;
  using vertex_descriptor
    = typename boost::graph_traits<graph>::
      vertex_descriptor;
  using edge_descriptor
    = typename boost::graph_traits<graph>::
      edge_descriptor;
  using edge_insertion_result
    = std::pair<edge_descriptor,bool>;

  graph g;
  const vertex_descriptor va = boost::add_vertex(g);
  const vertex_descriptor vb = boost::add_vertex(g);
```

```
  const edge_insertion_result ea = boost::add_edge(va, vb
      , g);
  assert(ea.second);
  return g;
}
```

Note that this code has more lines of using statements than actual code! In this code, the third template argument of boost::adjacency_list is boost::undirectedS, to select (that is what the S means) for an undirected graph. Adding a vertex with boost::add_vertex results in a vertex descriptor, which is a handle to the vertex added to the graph. Two vertex descriptors are then used to add an edge to the graph. Adding an edge using boost::add_edge returns two things: an edge descriptor and a boolean indicating success. In the code example, we assume insertion is successfull.

Note that the graph lacks all properties: nodes do not have names, nor do edges.