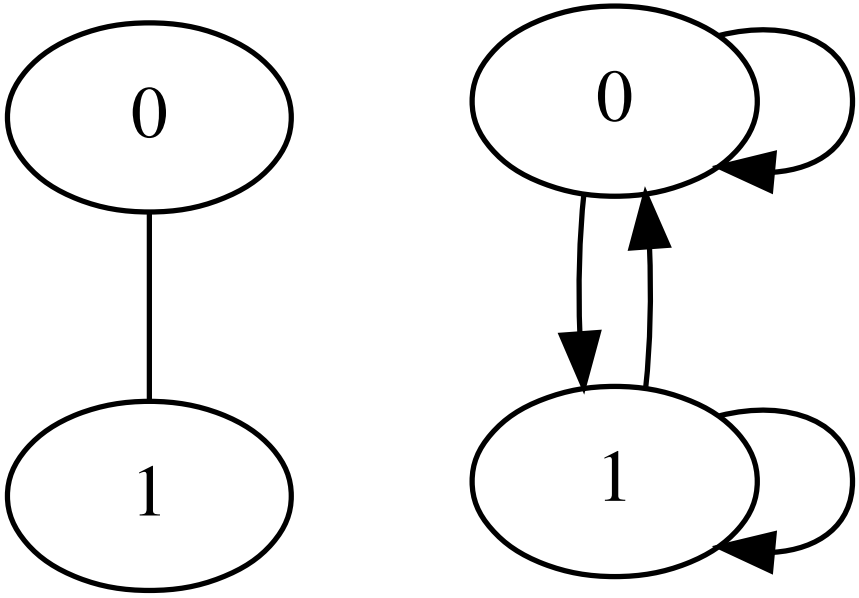
Boost.Graph

Richel Bilderbeek

GitHub

Graphs

- Vertices/nodes and edges
- Directed or undirected

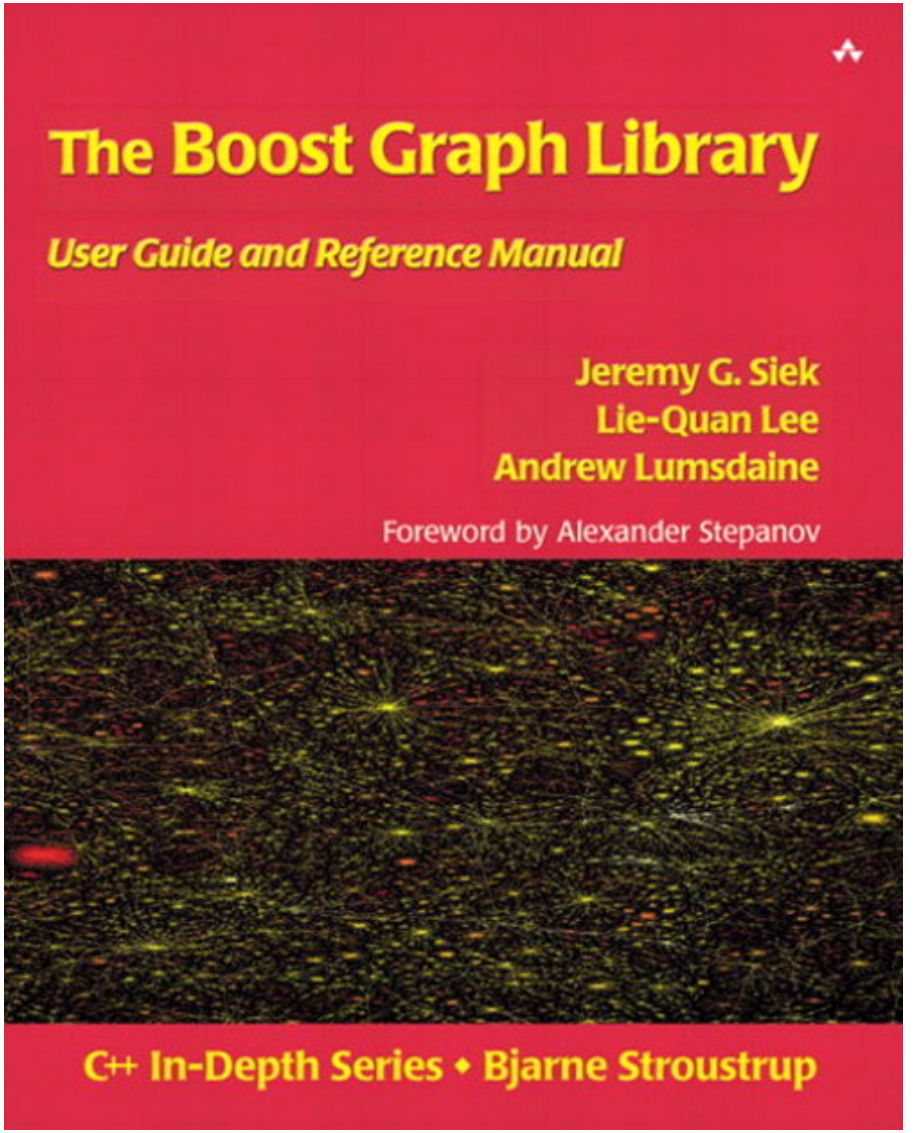


Application of graphs

- Fundamental mathematical concept
- Neural networks
- Reinforcement learning
- Path-finding
- Order of complex tasks

Boost.Graph

- Part of the Boost libraries
- Flexibility by templates
- Steep learning curve



Scott Meyers. Effective C++ (3rd edition). Item 54: Familiarize yourself with Boost

Problem

- Compile errors are unreadable
- Documentation:
  - Is not ordered non-chronologically
  - Does not increase gradually in complexity
  - Offers few complete code snippets

CC BY SA

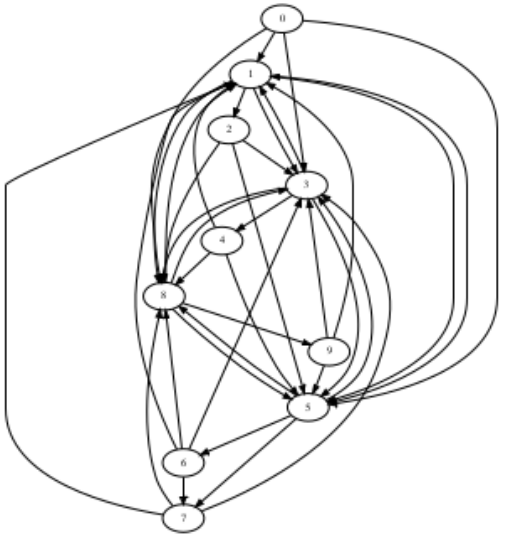
My work

- Chronologically ordered
- Starts simple
- Short and complete code snippets
- Following the C++ Core Guidelines
- Well received by the community
- >300 pages

A well-connected C++11 Boost.Graph tutorial

Richèl Bilderbeek

December 30, 2015



Contents

GitHub

Unwatch 10 Star 60 Fork 4

Steps

- Create an empty graph
- Add a vertex
- Add an edge
- Check if graphs are isomorphisms

Create an empty directed graph

```
#include <boost/graph/adjacency_list.hpp>

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

Create an empty directed graph

```
#include "create_empty_directed_graph.h"

void create_empty_directed_graph_demo() noexcept
{
  const auto g = create_empty_directed_graph();
}
```

Create an empty undirected graph

```
#include <boost/graph/adjacency_list.hpp>

boost::adjacency_list<
  boost::vecS,
  boost::vecS,
  boost::undirectedS
>
create_empty_undirected_graph() noexcept
{
  return {};
}
```

Create an empty undirected graph

```
#include "create_empty_undirected_graph.h"

void create_empty_undirected_graph_demo() noexcept
{
  const auto g
  = create_empty_undirected_graph();
}
```

Recap

- Creating an empty graph = defining its type
- C++11 '{}' and 'auto' comes in handy!
- boost::adjaceny_list has six template arguments
  - How to store the vertices, out edges, edges
  - Properties of vertices, edges and graph

Add a vertex

```
#include <type_traits>
#include <boost/graph/adjacency_list.hpp>

template <typename graph>
typename boost::graph_traits<
  graph
>::vertex_descriptor
add_vertex(graph& g) noexcept
{
  static_assert(!std::is_const<graph>::value,
    "graph cannot be const"
  );
  const auto vd = boost::add_vertex(g);
  return vd;
}
```

Add a vertex

```
#include "add_vertex.h"
#include "create_empty_directed_graph.h"
#include "create_empty_undirected_graph.h"

void add_vertex_demo() noexcept
{
  auto g = create_empty_undirected_graph();
  add_vertex(g);
  assert(boost::num_vertices(g) == 1);

  auto h = create_empty_directed_graph();
  add_vertex(h);
  assert(boost::num_vertices(h) == 1);
}
```

Recap

- Vertices can be added to any type of graph
  - All graphs have vertices!
- Vertex (or edge) descriptors → work on graph

Add an edge

```
#include <cassert>
#include <type_traits>
#include <boost/graph/adjacency_list.hpp>

template <typename graph>
typename boost::graph_traits<graph>::edge_descriptor
add_edge(graph& g) noexcept
{
  static_assert(!std::is_const<graph>::value,
    "graph cannot be const"
  );
  const auto vd_a = boost::add_vertex(g);
  const auto vd_b = boost::add_vertex(g);
  const auto aer = boost::add_edge(
    vd_a, vd_b, g
  );
  assert(aer.second);
  return aer.first;
}
```

Add an edge

```
#include "add_edge.h"
#include "create_empty_directed_graph.h"
#include "create_empty_undirected_graph.h"

void add_edge_demo() noexcept
{
  auto g = create_empty_undirected_graph();
  add_edge(g);
  assert(boost::num_vertices(g) == 2);
  assert(boost::num_edges(g) == 1);

  auto h = create_empty_directed_graph();
  add_edge(h);
  assert(boost::num_vertices(h) == 2);
  assert(boost::num_edges(h) == 1);
}
```

Recap

- To add an edge one needs vertex descriptors

Check for isomorphism

```
#include <boost/graph/isomorphism.hpp>

template <typename graph1, typename graph2>
bool is_isomorphic(
  const graph1 g,
  const graph2 h
) noexcept
{
  return boost::isomorphism(g,h);
}
```

Check for isomorphism

```
#include <cassert>
#include "create_path_graph.h"
#include "create_k3_graph.h"
#include "is_isomorphic.h"

void is_isomorphic_demo() noexcept
{
  const auto g = create_path_graph(3);
  const auto h = create_k3_graph();
  assert( is_isomorphic(g,g));
  assert(!is_isomorphic(g,h));
}
```

Conclusion

- Simple things are already template-heavy
- Complex things are available

- Complex things do not fit on a slide

Discussion

- Still unknown how to save and load a graph with a name