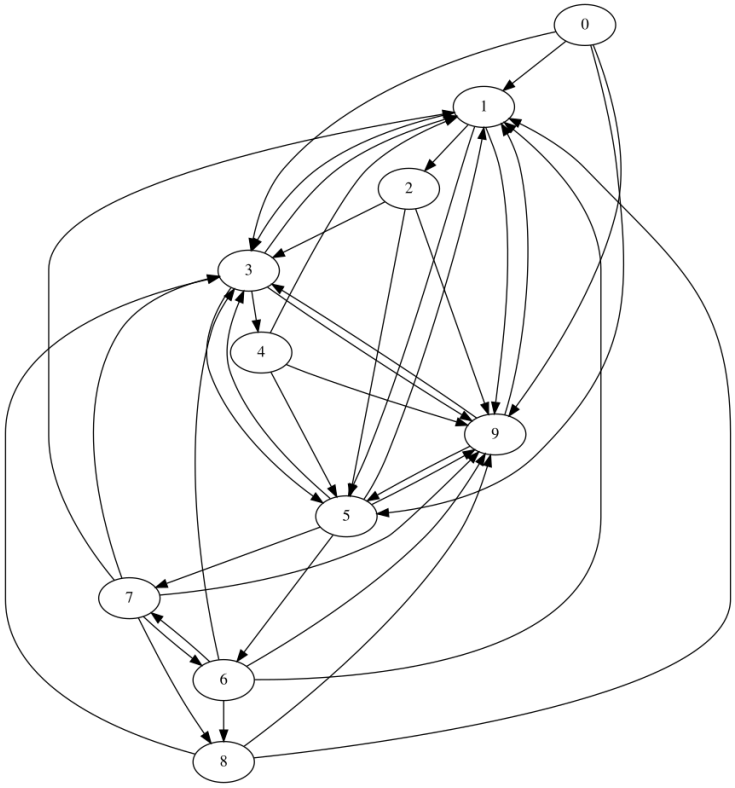
Boost.Graph Cookbook 1: Basics



Richèl J.C. Bilderbeek

May 14, 2021

0.1 Introduction

This is 'Boost.Graph Cookbook 1: Basics', version 3.3.

0.1.1 Why this tutorial

I needed this tutorial already in 2006, when I started experimenting with Boost.Graph. More specifically, I needed a tutorial that:

- Orders concepts chronologically
- Increases complexity gradually
- Shows complete pieces of code

What I had were the book [1] and the Boost.Graph website, both did not satisfy these requirements.

0.1.2 Tutorial style

Readable for beginners This tutorial is aimed at the beginner programmer. This tutorial is intended to take the reader to the level of understanding the book [1] and the Boost.Graph website require. It is about basic graph manipulation, not the more advanced graph algorithms.

High verbosity This tutorial is intended to be as verbose, such that a beginner should be able to follow every step, from reading the tutorial from beginning to end chronologically. Especially in the earlier chapters, the rationale behind the code presented is given, including references to the literature. Chapters marked with △ are optional, less verbose and bring no new information to the storyline.

Repetitiveness This tutorial is intended to be as repetitive, such that a beginner can spot the patterns in the code snippets their increasing complexity. Extending code from this tutorial should be as easy as extending the patterns.

Index In the index, I did first put all my long-named functions there literally, but this resulted in a very sloppy layout. Instead, the function 'do_something' can be found as 'Do something' in the index. On the other hand, STL and Boost functions like 'std::do_something' and 'boost::do_something' can be found as such in the index.

0.1.3 Coding style

Concept For every concept, I will show

- a function that achieves a goal, for example `create_empty_undirected_graph`
- a test case of that function, that demonstrates how to use the function, for example `create_empty_undirected_graph_test`
- Three

C++14 All coding snippets are taken from compiled and tested C++14 code. I chose to use C++14 because it was available to me on all local and remote computers. Next to this, it makes code even shorter then just C++11.

Coding standard I use the coding style from the Core C++ Guidelines. At the time of this writing, the Core C++ Guidelines were still in early development, so I can only hope the conventions I then chose to follow are still Good Ideas.

No comments in code It is important to add comments to code. In this tutorial, however, I have chosen not to put comments in code, as I already describe the function in the tutorial its text. This way, it prevents me from saying the same things twice.

Trade-off between generic code and readability It is good to write generic code. In this tutorial, however, I have chosen my functions to have no templated arguments for conciseness and readability. For example, a vertex name is std::string, the type for if a vertex is selected is a boolean, and the custom vertex type is of type `my_custom_vertex`. I think these choices are reasonable and that the resulting increase in readability is worth it.

Long function names I enjoy to show concepts by putting those in (long-named) functions. These functions sometimes border the trivial, by, for example, only calling a single Boost.Graph function. On the other hand, these functions have more English-sounding names, resulting in demonstration code that is readable. Additionally, they explicitly mention their return type (in a simpler way), which may be considered informative.

Long function names and readability Due to my long function names and the limitation of â50 characters per line, sometimes the code does get to look a bit awkward. I am sorry for this.

Use of auto I prefer to use the keyword auto over doubling the lines of code for using statements. Often the `do` functions return an explicit data type, these can be used for reference. Sometime I deduce the return type using decltype and a function with the same return type. When C++17 gets accessible, I will use `decltype(auto)` If you really want to know a type, you can use the `get_type_name` function (chapter **??**)

Explicit use of namespaces On the other hand, I am explicit in the namespaces of functions and classes I use, so to distinguish between types like `std::array` and `boost::array`. Some functions (for example, `get`) reside in the namespace of the graph to work on. In this tutorial, this is in the global namespace. Thus, I will write `get`, instead of `boost::get`, as the latter does not compile.

Use of STL algorithms I try to use STL algorithms wherever I can. Also you should prefer algorithm calls over hand-written for-loops ([2], chapter 18.12.1 and [3], item 43). Sometimes using these algorithms becomes a burden on the lines of code. This is because in C++11, a lambda function argument (use by the algorithm) must have its data type specified. It may take multiple lines of `using` statements being able to do so. In C++14 one can use `auto` there as well. So, only if it shortens the number of lines significantly, I use raw for-loops, even though you shouldn't.

Re-use of functions The functions I develop in this tutorial are re-used from that moment on. This improves to readability of the code and decreases the number of lines.

Tested to compile All functions in this tutorial are tested to compile using GitHub Actions in both debug and release mode.

Tested to work All functions in this tutorial are tested, using the Boost.Test library. GitHub Actions calls these tests after each push to the repository.

Availability The code, as well as this tutorial, can be downloaded from the GitHub at `www.github.com/richelbilderbeek/boost_graph_cookbook_1`.

0.1.4 License

This tutorial is licensed under Creative Commons license 4.0. All C++ code is licensed under GPL 3.0.

Figure 1: Creative Commons license 4.0

0.1.5 Feedback

This tutorial is not intended to be perfect yet. For that, I need help and feedback from the community. All referenced feedback is welcome, as well as any constructive feedback.

I have tried hard to strictly follow the style as described above. If you find I deviated from these decisions somewhere, I would be grateful if you'd let know. Next to this, there are some sections that need to be coded or have its code improved.

0.1.6 Acknowledgements

These are users that improved this tutorial and/or the code behind this tutorial, in chronological order:

- m-dudley, `http://stackoverflow.com/users/111327/m-dudley`
- E. Kawashima
- mat69, `https://www.reddit.com/user/mat69`
- danielhj, `https://www.reddit.com/user/danieljh`
- sehe, `http://stackoverflow.com/users/85371/sehe`

- cv_and_me, `http://stackoverflow.com/users/2417774/cv-and-he`
- mywtfmp3

0.1.7 Outline

The chapters of this tutorial are also like a well-connected graph. To allow for quicker learners to skim chapters, or for beginners looking to find the patterns.

The distinction between the chapter is in the type of edges and vertices. They can have:

- no properties: see chapter 0.2
- have a bundled property: see chapter **??**

Pivotal chapters are chapters like 'Finding the first vertex with ...', as this opens up the door to finding a vertex and manipulating it.

All chapters have a rather similar structure in themselves, as depicted in figure 2.

There are also some bonus chapters, that I have labeled with a $\triangle$. These chapters are added I needed these functions myself and adding them would not hurt. Just feel free to skip them, as there will be less theory explained.

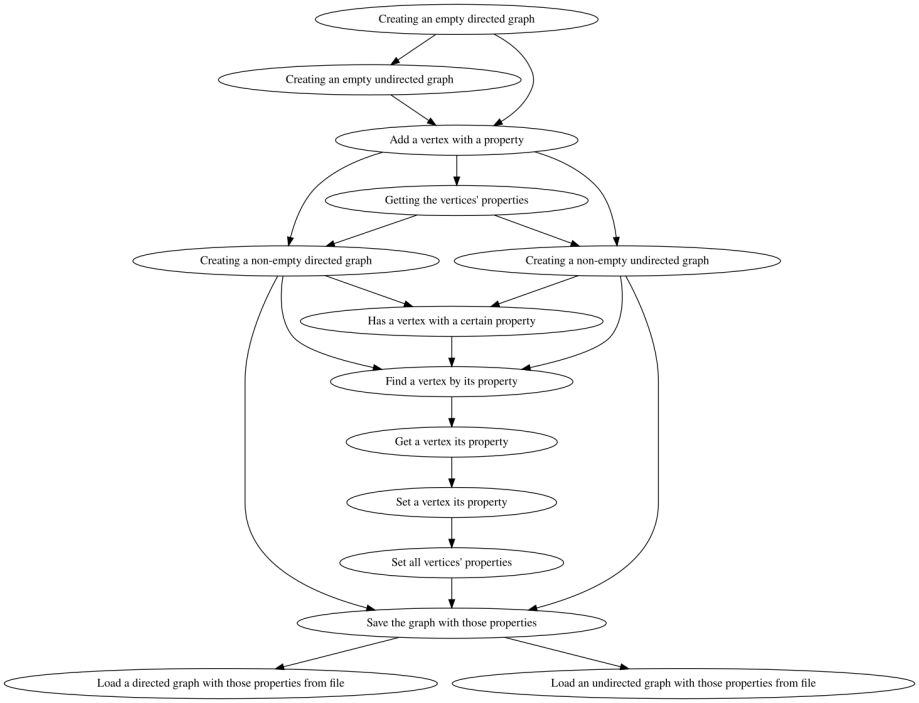


Figure 2: The relations between sub-chapters

0.2 Building graphs without properties

Boost.Graph is about creating graphs. In this chapter we create the simplest of graphs, in which edges and nodes have no properties (e.g. having a name).

Still, there are two types of graphs that can be constructed: undirected and directed graphs. The difference between directed and undirected graphs is in the edges: in an undirected graph, an edge connects two vertices without any directionality, as displayed in figure **??**.

In a directed graph, an edge goes from a certain vertex, its source, to another (which may actually be the same), its target. A directed graph is shown in figure **??**.

Chapter 1

Working on graphs without properties

Working on graphs without properties text.

Chapter 2

Building graphs with bundled vertices

Building graphs with bundled vertices text.

Chapter 3

Working on graphs with bundled vertices

Working on graphs with bundled vertices text.

Chapter 4

Building graphs with bundled edges and vertices

Building graphs with bundled edges and vertices text.

Chapter 5

Working on graphs with bundled edges and vertices

Working on graphs with bundled edges and vertices text.

Chapter 6

Building graphs with a graph name

Building graphs with a graph name text.

Chapter 7

Working on graphs with a graph name

Working on graphs with a graph name text.

Chapter 8

Other graph functions

Other graph functions text.

Chapter 9

Misc functions

Misc functions text.

Chapter 10

Errors

Errors text.

Bibliography

[1] Jeremy G Siek, Lie-Quan Lee, and Andrew Lumsdaine. *Boost Graph Library: User Guide and Reference Manual, The*. Pearson Education, 2001.

[2] Bjarne Stroustrup. *The C++ Programming Language (3rd edition)*. 1997. ISBN 0-201-88954-4.

[3] Scott Meyers. *Effective C++: 55 specific ways to improve your programs and designs*. Pearson Education, 2005.

Appendix A

Appendix

Appendix text.