Graphs and Algorithms

Martin Blom[*] and Jonatan Langlet[†]

Karlstad University, Sweden

November 24, 2019

1 Introduction

Graph theory has many applications that range from linguistics and social sciences to physics, chemistry, and computer science. For example, graphs can be used to study lexical semantics, gossip, molecules, websites, and computer networks [1]. Formally, a graph $G = \{V, E\}$ is defined as a set of vertices V and a set of edges E, where $(a, b) \in E$ represents a relationship between two vertices $a, b \in V$. Depending on the type of graph, edges may be (un)directed and associated with numeric numbers (so called weights). As an example of an edge in a weighted directed graph, $(a, b, 385)$ could represent the price of a bus ticket from location a to location b.

[*]martin.blom@kau.se

[†]jonatan.langlet@gmail.com

1.1 Learning Goals and Preparation

Throughout the lab you will learn how a graph can be represented by an adjacency list, implementing it from scratch based on its basic *principles*. You will also get hands-on experience working with four different graph algorithms: Prim, Dijkstra, Floyd, and Warshall. We recommend that you try these algorithms using pen and paper *before* implementing them: both to deepen your understanding and to get a few test-cases on your own.

2 Task

2.1 Getting Started

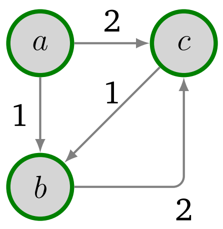
The start-code is available on Canvas. Alternatively, you can clone it from the terminal: `git clone https://git.cs.kau.se/rasmoste/dvgb03.git`. Next, navigate to `lab3-graphs` and run `./bin/main --help` to show available options. For example, the default mode is to use a directed graph but it can be changed as follows: `./bin/main -m undirected`. Note: we recommend directed mode until the adjacency list is fully implemented.

2.2 Adjacency List

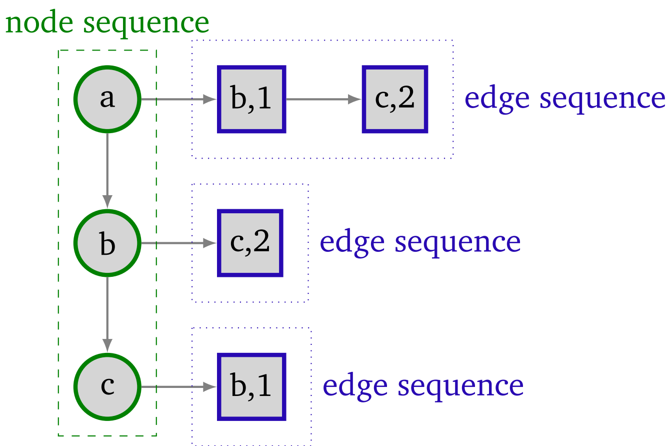
As shown in Figure 1, an adjacency list can be viewed as a sequence of nodes where each node tracks its neighbors as an edge sequence. For example, node a is the source of two edges that goes towards b on weight one and c on weight two. Therefore, the edge sequence of node a is $(b, 1)$ followed by $(c, 2)$. In the start code, you can think of the node sequence as the `AdjacencyList` class and the respective edge sequences as instances of the `Edge class`. Your task is to implement these classes based on the provided skeletons, maintaining the adjacency list in *lexicographical order* at all times. To get a quick overview, you may list all methods that are TODO-marked as follows: `cat src/adjlist.py | grep TODO`. Note that the expected behavior of each method is documented in the source code. Further, you are encouraged to add helper methods where appropriate.

2.3 Algorithms

Your next task is to use your adjacency list to implement four different graph algorithms: Dijkstra, Prim, Warshall, and Floyd. You will find further



(a) Graph



(b) Adjacency list

Figure 1: A graph viewed as an adjacency list.

documentation regarding excepted input and output in `src/algorithm.py`. Hint: you may need to add helper methods to (un)mark nodes and such.

2.4 Strict Implementation Criteria

- Any method that is already defined in the start-code must not be changed in terms of name and expected input/output.
- You need not import any additional libraries. If, for some reason, you still want to do it though: please discuss it with a lab supervisor first.

2.5 Points Awarded

The lab is automatically corrected using the unit tests that are provided in the `test` directory. You are encouraged to run them to verify that your solution is correct *before submission*.[1] If all tests pass for a given task, you will be awarded points as described in Table 1. It should be noted that we only use test-cases that have well-defined behavior. For example, Dijkstra will not be tested on a graph that has multiple alternative shortest paths.

[1]Further instructions can be found in the `README.md` file.

Table 1: Score overview; points are awarded iff all automatic tests pass.

Task	Points
Adjacency list	2
Warshall	1
Floyd	1
Dijkstra	1
Prim	1

3 Submission

You may work alone or in pairs of two. Use the provided start-code and follow all criteria as described in this specification and the provided source code. Submit the following on Canvas no later than **January 14, 2020**:

- Names of all group members in the comment field.
- `adjlist.py` and `algorithm.py` without any zipping.

Acknowledgements

This lab was initially created by Donald F. Ross. The current start-code and specification was repackaged by Rasmus Dahlberg and Martin Blom.

References

[1] Wikipedia contributors. Graph theory — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Graph_theory&oldid=926124335, 2019. accessed 2019-11-17.