Data Structures and Algorithms

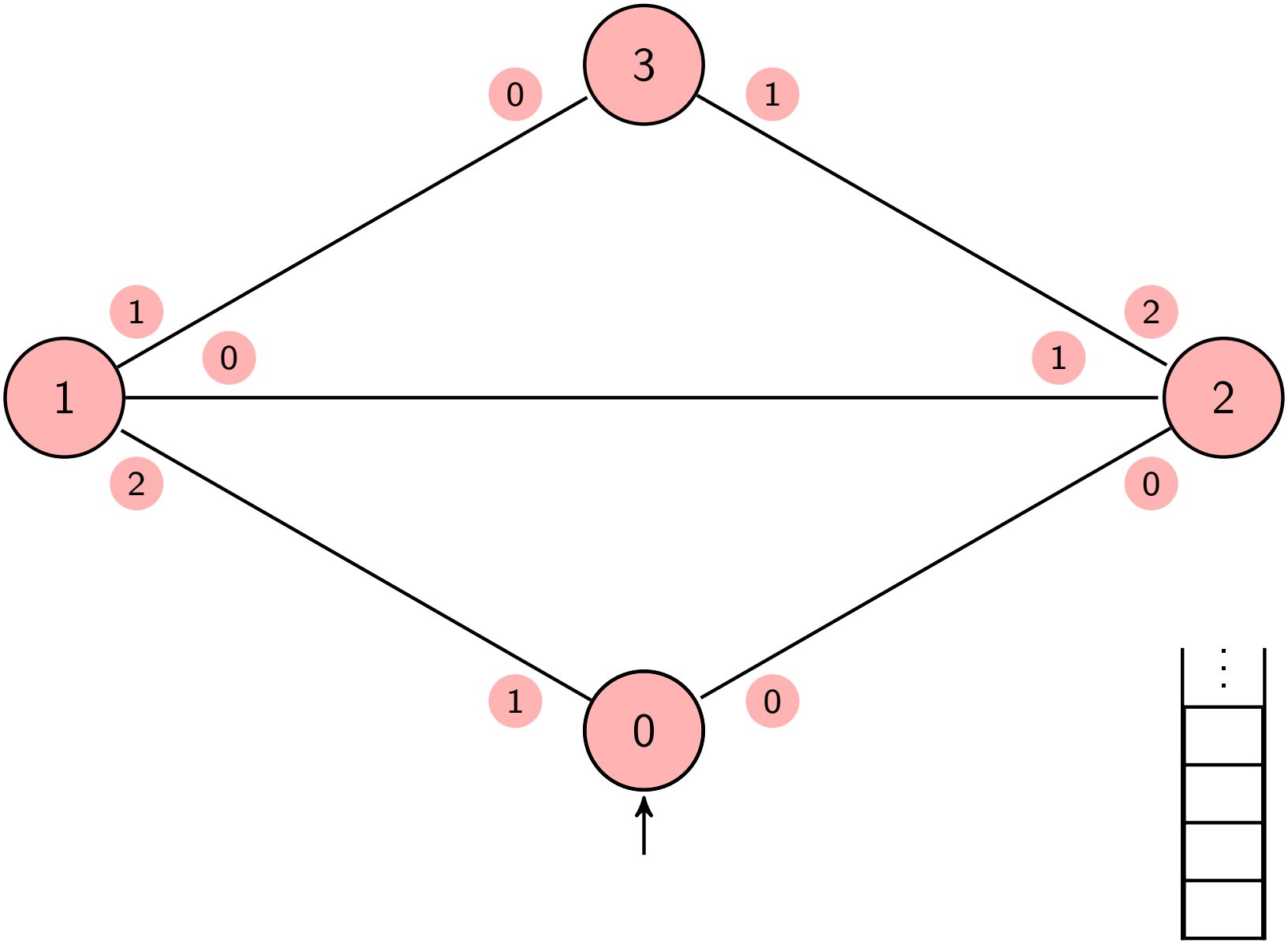
Assignment 2

Alan P. Sexton

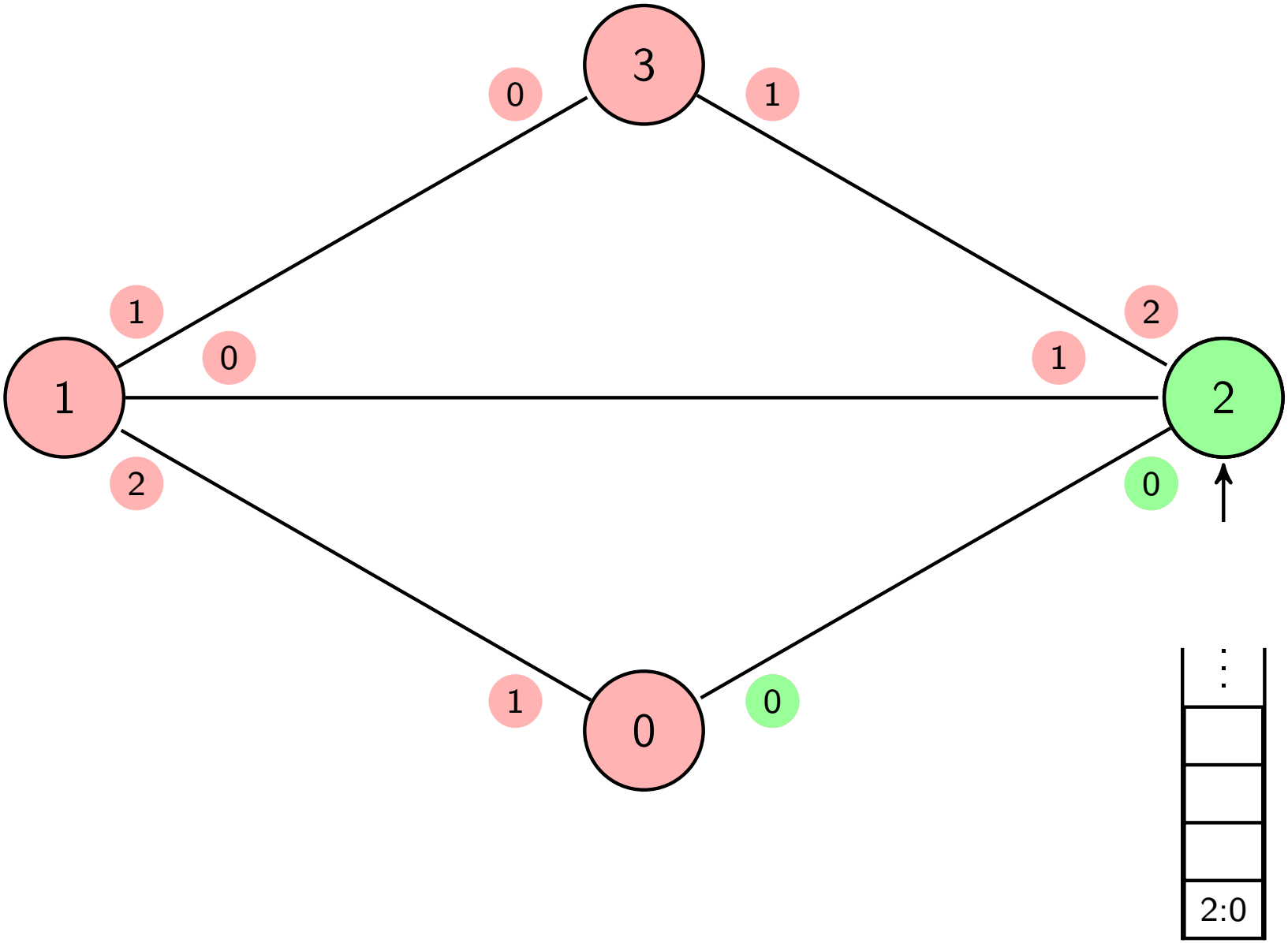
University of Birmingham

Spring 2019

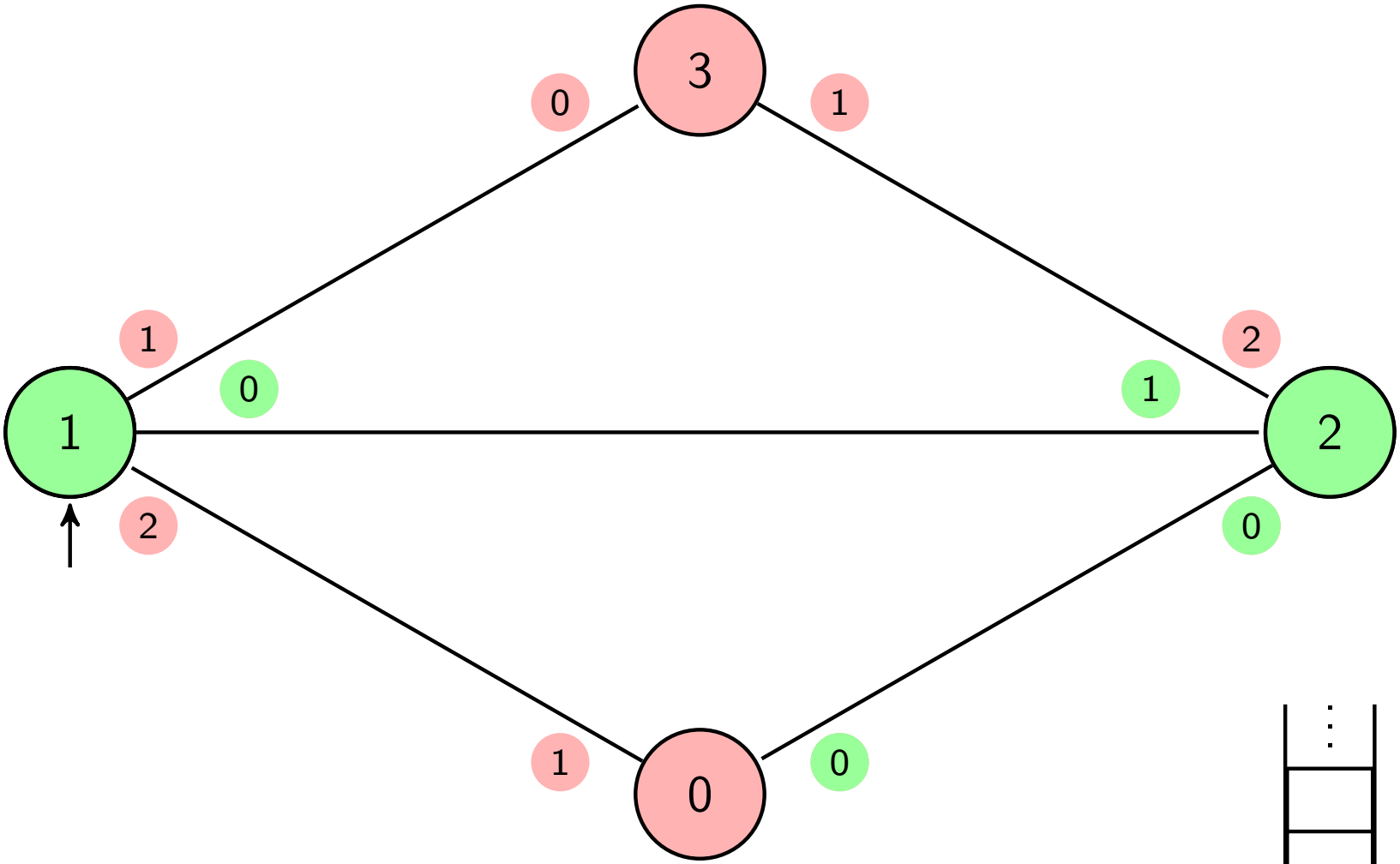
Example Search Execution



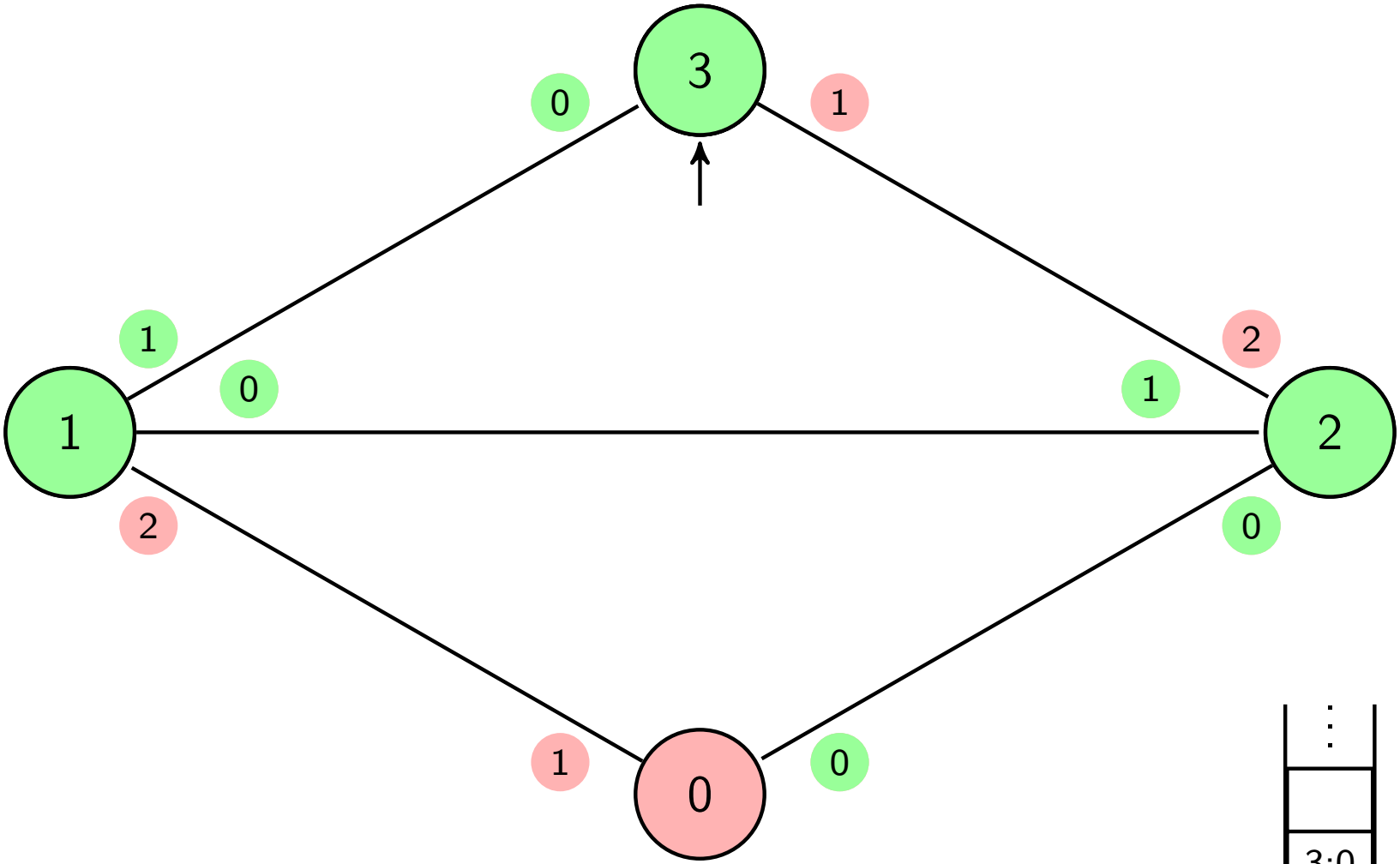
Example Search Execution



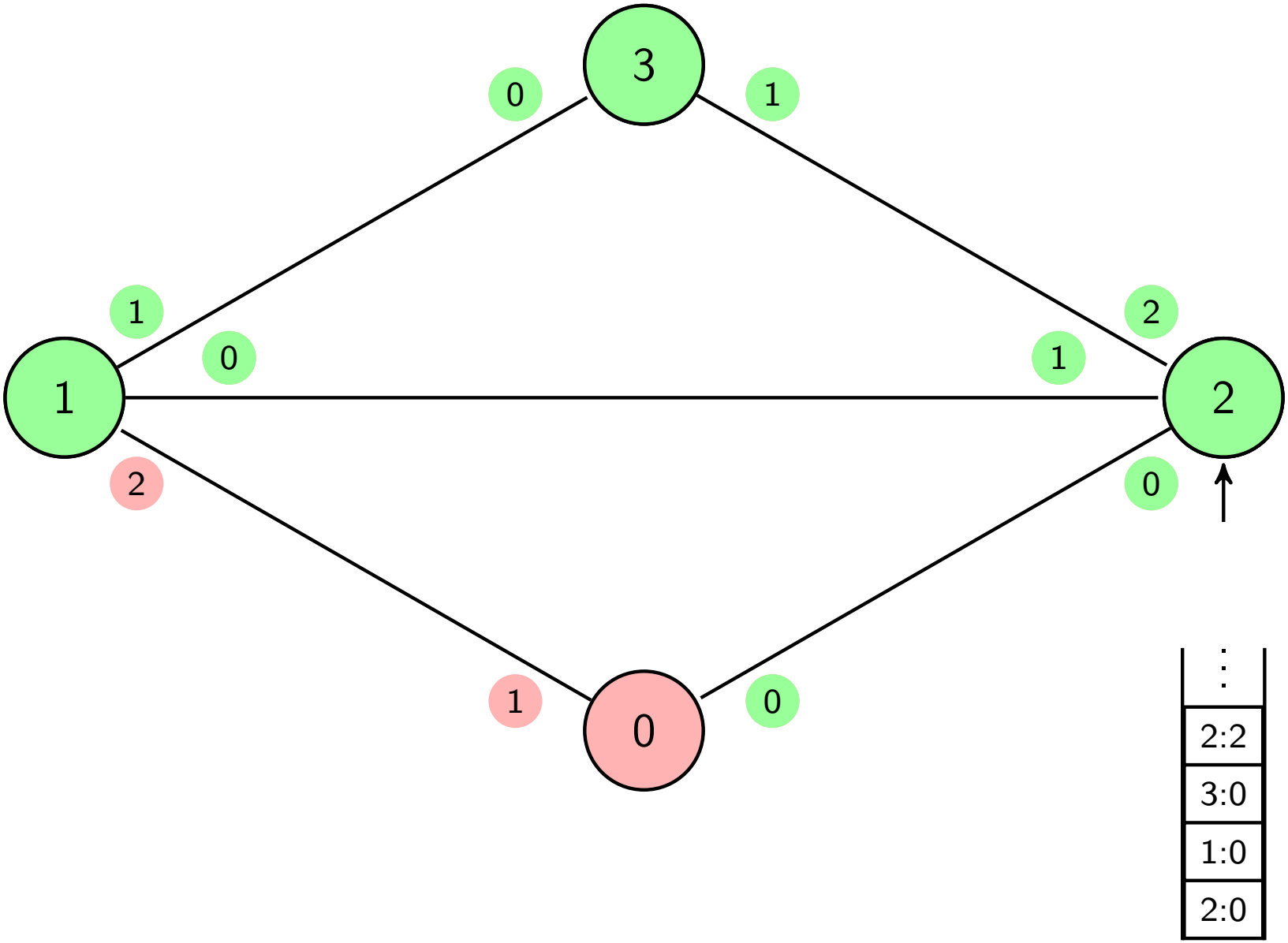
Example Search Execution



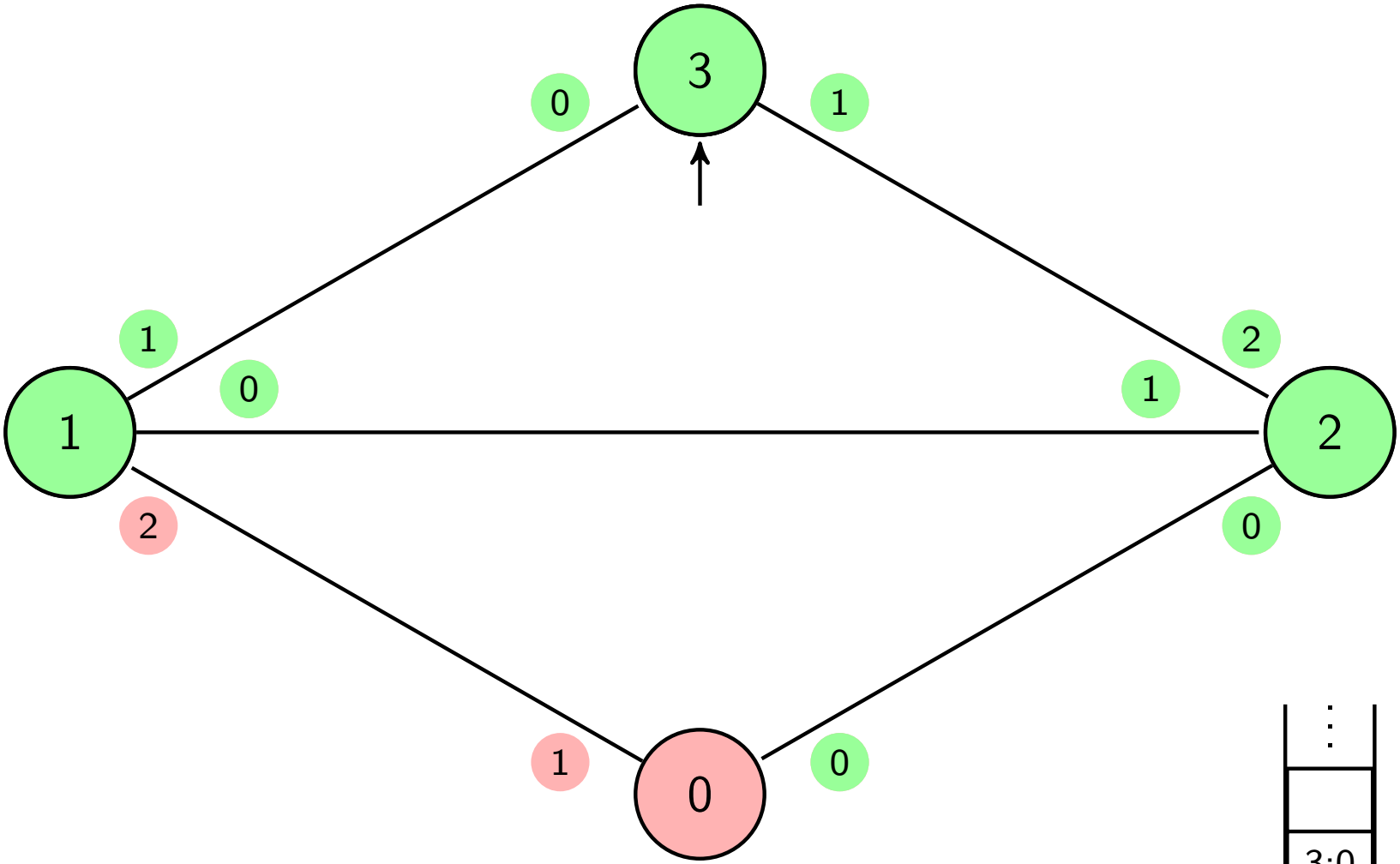
Example Search Execution



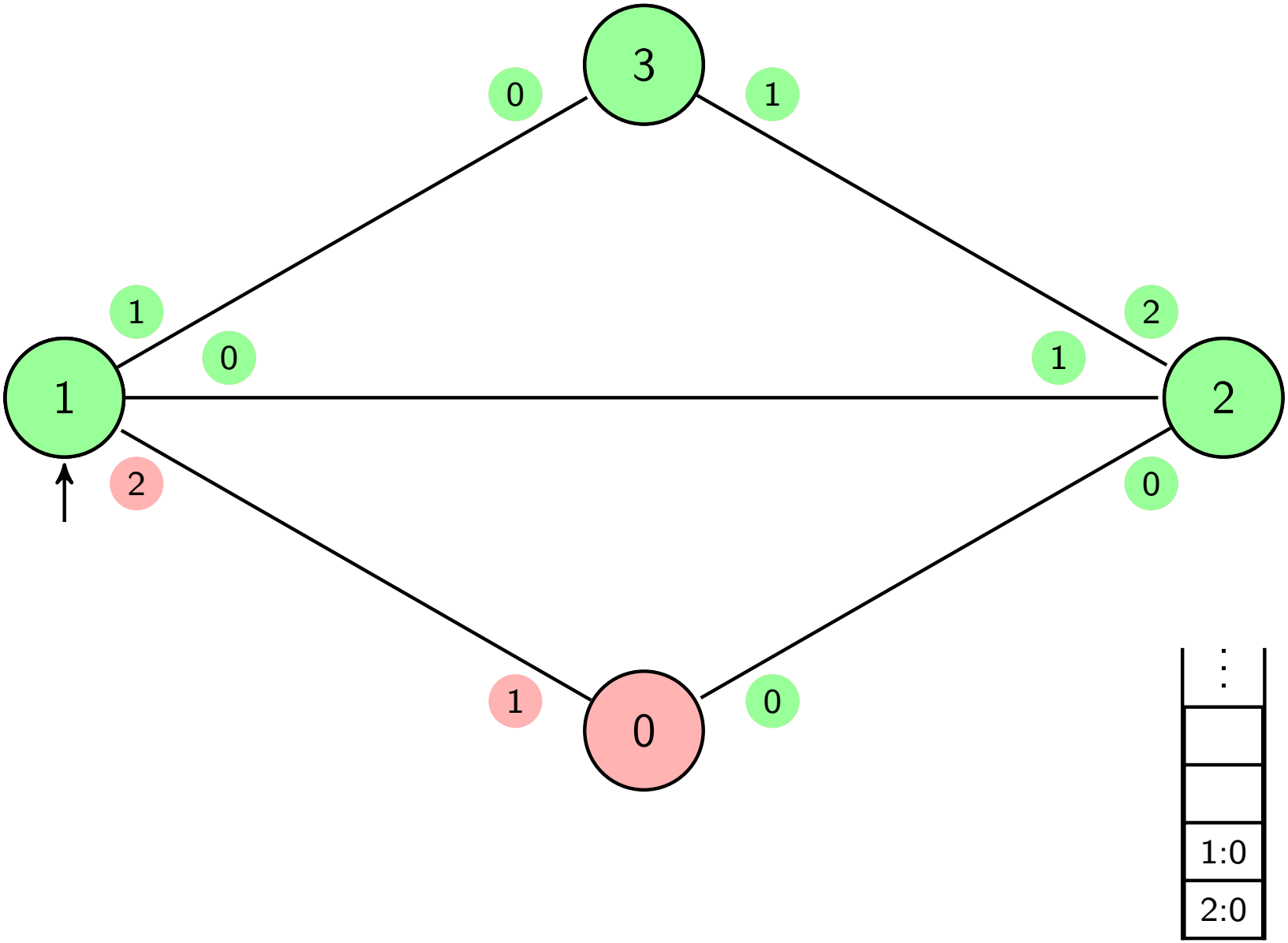
Example Search Execution



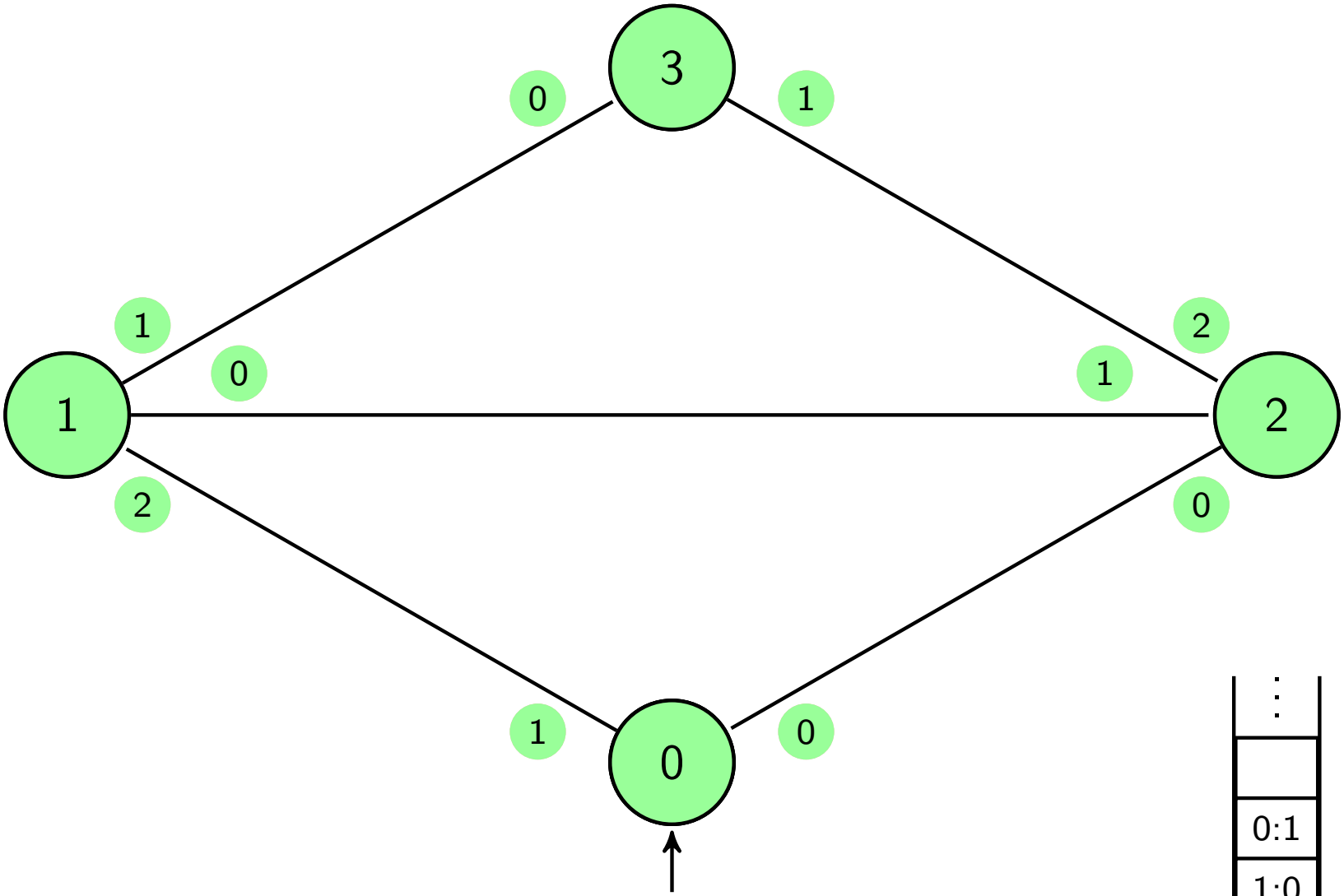
Example Search Execution



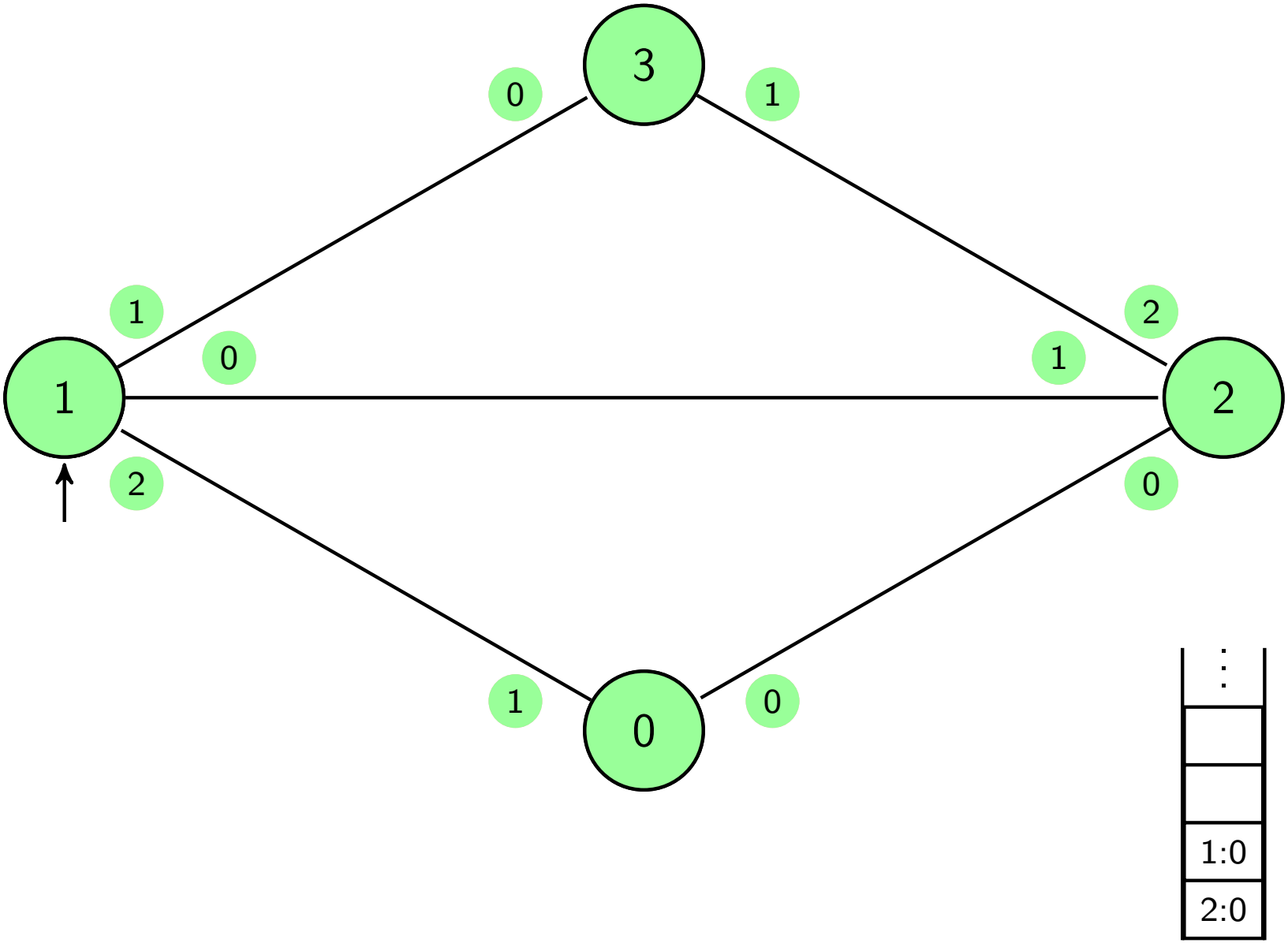
Example Search Execution



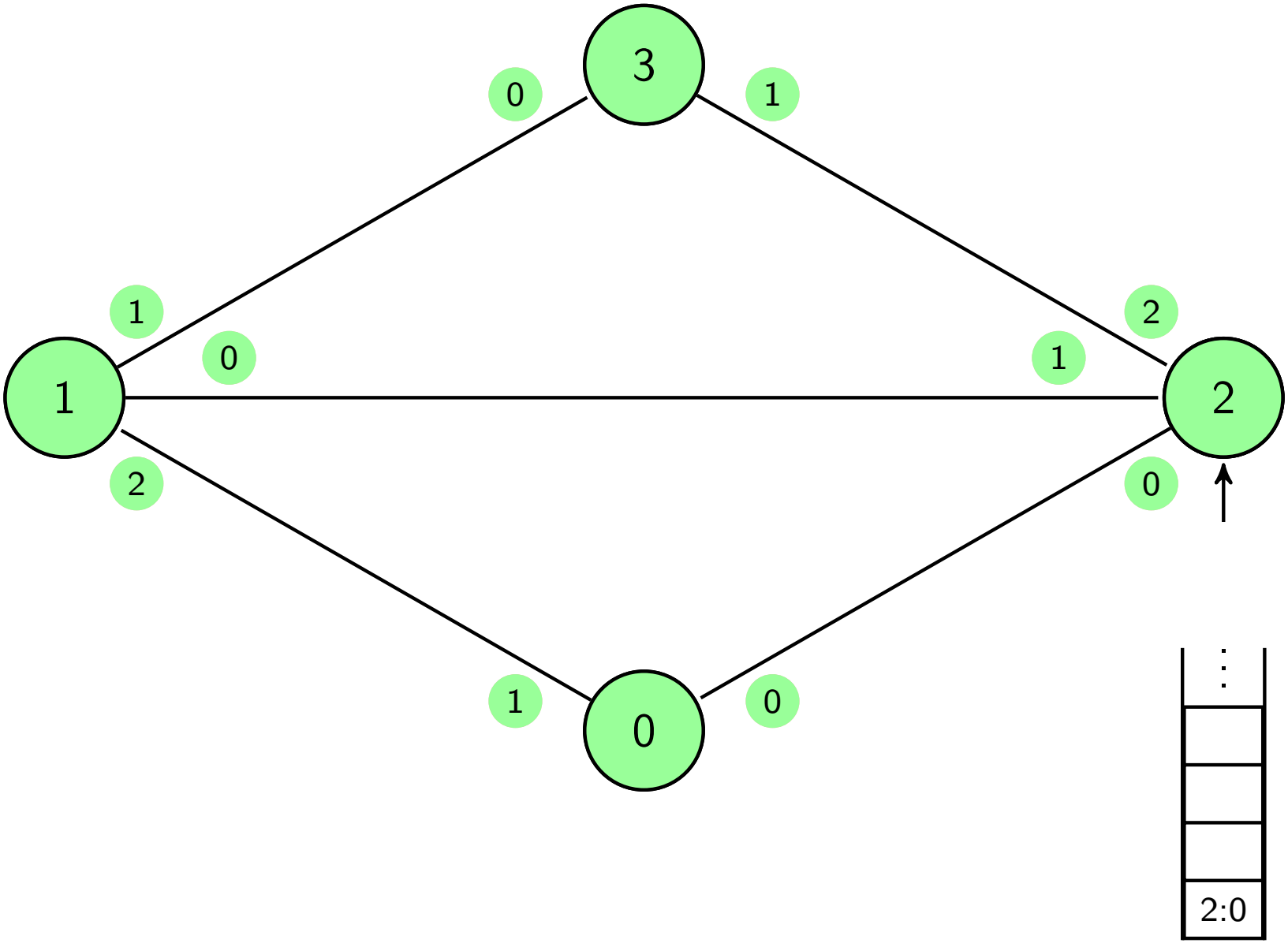
Example Search Execution



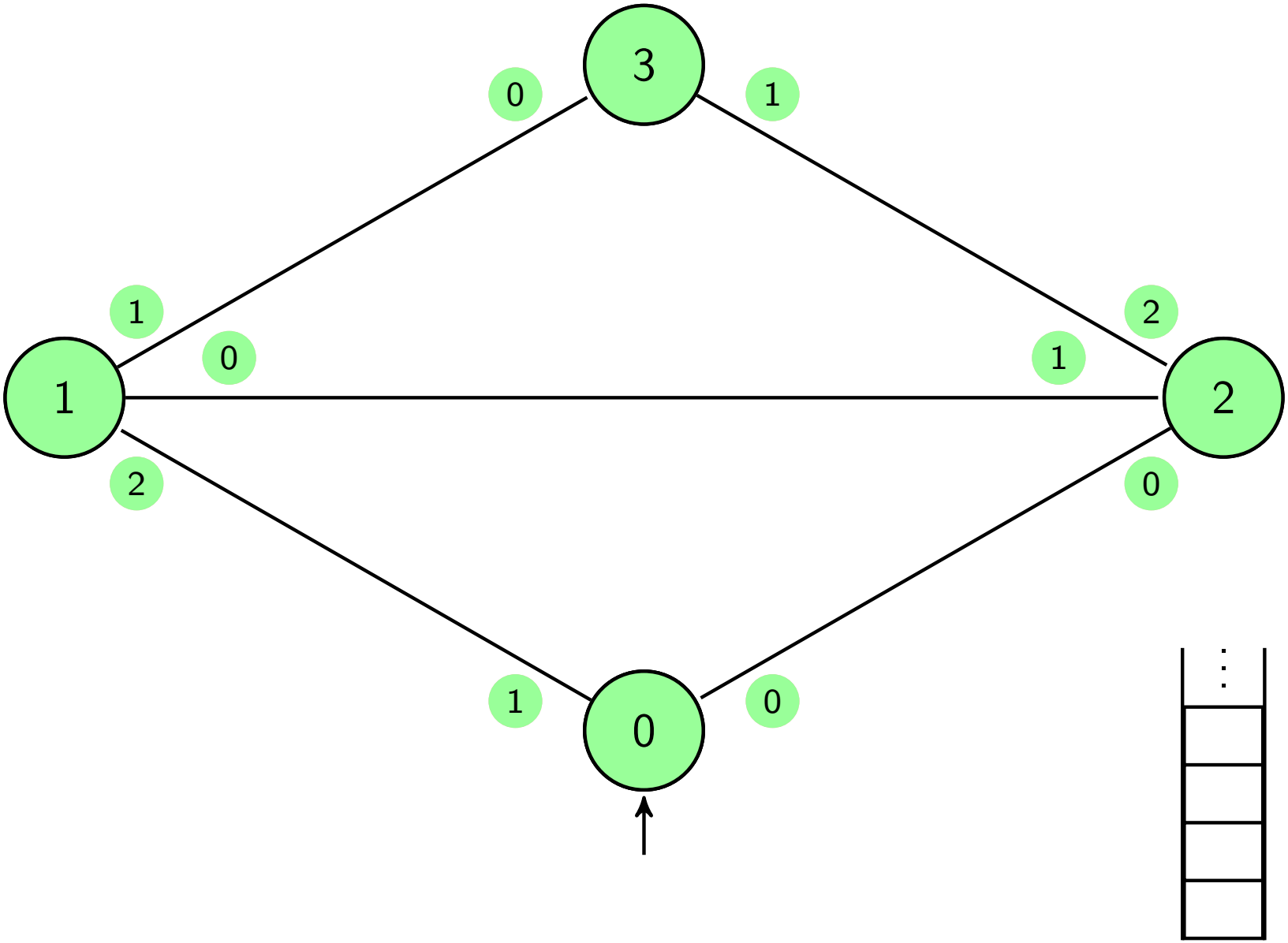
Example Search Execution



Example Search Execution



Example Search Execution



Choosing Data Structures

When programming in Java and a mechanism for holding, searching within, and manipulating collections of objects of some kind is needed, beginner programmers often reach for an `ArrayList<>`

- Very easy to understand
- Easy to manipulate `add(index, element)`, `get(index)`, `remove(index)`, `contains(object)`, etc.

However, while `ArrayList<>` is good at many things, there are other collection classes that are much better for some tasks

- Better means: enables shorter, simpler, more efficient code

When you find yourself reaching for `ArrayList<>`, **FIRST** consider whether your problem would be better dealt with by a:

- `List<>`, `Deque<>`, `Set<>`, `Map<>`, or one of their variants

Choosing Data Structures

When choosing a data structure, first identify the operations that your problem requires of the collection of objects you are working with

- This does not mean *"can I implement what I need using* `ArrayList<>`*"* (the answer is usually yes, with sometimes a lot of effort), but rather *"what operations need to work in the end, whether I have to implement them or the data structure provides them for free"*
- With this question answered, you are in a position to select a data structure, while also paying attention to the complexity costs of the operations required using your selection

Data Structures for Assignment 02

- The stack for handling back-tracking is already provided:
  - `Deque<Portal> visitStack`
- You need a data structure to keep track of which portals you have already traversed
  - Use it when in a chamber to choose an unused portal to traverse, or, if there is none, to decide to back-track
  - You can use a `ArrayList<>` for this, but you have to do more programming, and more complicated programming, to make it work than necessary
  - Look through other options, and see what was used in the rest of the program (`Maze` and `DroneTest` classes) and see what would make the programming shorter and easier
- Whatever data structure you use, read through the API and make sure you are familiar with the different methods available to them, paying particular attention to their return values
- Don't forget to look at the utility static methods in the `Collections` and `Arrays` classes