The Little Grasshopper

The Graceful Tree Conjecture

Overview

This post is a collection of notes about an interesting unproven problem in combinatorics. The Kotzig-Ringel conjecture, better known as the Graceful Tree Conjecture (GTC), claims that:

> All trees are graceful.

This is a simple statement but mathematicians don't know if it's true or not! At the time of this writing, nobody has come up with a counterexample and there is no proof that one exists.

Before discussing trees, let's consider the broader category of *simple* graphs. Simple graphs are finite undirected graphs with no self-adjacent vertices or duplicated edges. The *labeling* of a simple graph assigns an integer to each of its vertices and edges. (The British spelling is *labelling*.)

In a *graceful* labeling, each edge label (E_{ij}) is the absolute difference between the labels of its adjoining vertices (V_i and V_j). All edge labels must be unique, and all vertex labels must be unique.

> If a graceful graph has *n* edges, then labels V_i and E_{ij} can be assigned to its vertices and edges such that:
>
> - $V_i \in \{0, 1, \dots, n\}$
> - $V_i \neq V_j$ iff $i \neq j$
> - $\{ E_{ij} \} = \{1, \dots, n\}$
> - $E_{ij} = | V_i - V_j |$

Figure 1 depicts an example of a graph with graceful labels.

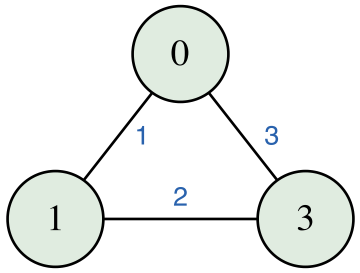


Figure 1. Graceful graph with three edges.

A *complementary* graceful labeling can be generated by subtracting each vertex label from *n*, as shown in the next figure. This shows that a given graph might permit more than one graceful labeling.

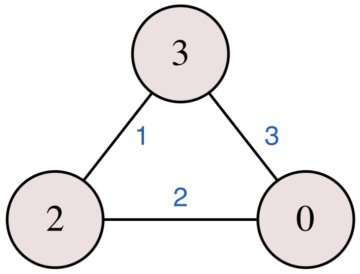


Figure 2. The graceful complement of Figure 1.

Graceful trees

Graphs that are *trees* are simple, connected, and acyclic. This implies that trees exhibit the property **V=E+1** where **V** is the number of vertices and **E** is the number of edges. Therefore, if you only consider graceful graphs that happen to be trees, you must use each vertex label in {0, 1, … , n} exactly once, and you must use each edge label in {1, … , n} exactly once.

(Once way of remembering V=E+1 is to apply Euler's Polyhedron Formula with only one face. Since trees have no cycles, they do not divide the plane.)

Figure 3 depicts a graceful tree. The edge labels are easy to compute so they are left as an exercise to the reader. While this happens to be a rooted tree, please note that the conjecture applies to all trees, even if they are unrooted.

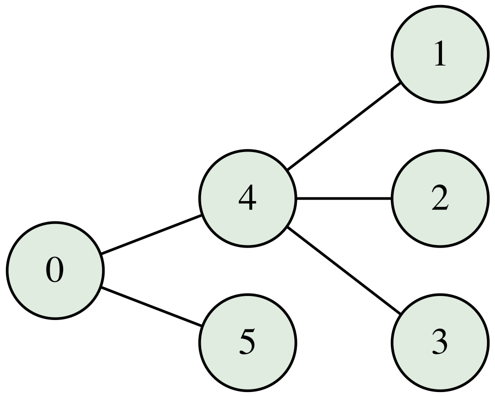


Figure 3. Graceful graph with five edges.

Representing graceful graphs

One neat thing about a graceful labeling is that you don't need to draw out the graph in order to uniquely identify it. Every graceful labeling can be represented with a sequence of positive integers that has the following property.

For a positive integer m, the sequence of integers ($j_1, j_2, j_3, \ldots, j_m$) is a *labeling sequence* if $0 \le j_i \le m-i$ for all $i \in [1,m]$.

To construct a graph from an integer sequence with *m* terms, follow this two-step process:

1. Assign the integers in [0,m] to isolated vertices.
2. For each term j_i in the sequence, draw an edge between j_i and $j_i + i$.

The integer sequences for the graphs in Figures 1 through 3 are shown below.

1. **(0, 1, 0)**
2. **(2, 0, 0)**
3. **(3, 2, 1, 0, 0)**

To build a graph, simply add the elements in the sequence with the natural numbers, like so:

1. **(0, 1, 0)** + (1, 2, 3) = **(1, 3, 3)**
2. **(2, 0, 0)** + (1, 2, 3) = **(3, 2, 3)**
3. **(3, 2, 1, 0, 0)** + (1, 2, 3, 4, 5) = **(4, 4, 4, 4, 5)**

The three graphs can be rendered from the above list by connecting each term in the left boldface sequence with its corresponding term in the right boldface sequence.

Conversely, to construct a sequence from a labeled graph, do this:

1. Go through the edges in order from 1 to *n*.
2. For each edge, pick the smaller of its two endpoint labels, and append it to the sequence.

These sequences were first demonstrated by David Sheppard in 1974. If you can prove that every acyclic connected graph (i.e. every tree) has a corresponding Sheppard sequence, then you can prove the conjecture and make history.

Counting graceful graphs

The mapping between those special integer sequences and graceful graphs implies that there exist *n!* graceful graphs that have *n* edges. Another way of coming to this conclusion is to consider each of the edge values, one by one, starting with the greatest edge value.

For example, if edge labels are in [1,5] and vertex labels are in [0,5], then:

The edge with label 5 must be between the vertex with label 0 and the vertex with label 5.

The above statement is always true in a graceful graph with 5 edges, because those two vertices are the only ones that are sufficiently far apart. Similarly:

The edge with label 4 must either (a) lie between vertex 0 and vertex 4, or (b) lie between vertex 1 and vertex 5.

So, there is 1 possibility for placing the "5" edge, and 2 possibilities for placing the "4" edge. If you continue this line of thought, a pattern emerges as the number of possible edge placements increases: 1 possibility, times 2 possibilities, times 3 possibilities…

The *n!* conclusion is all well and good for general graphs, but the more constrained problem of trees seems to have eluded mathematicians. I don't think anybody has come up with a closed form solution for "number of graceful trees with n edges".

Another integer sequence

However, there *is* a closed form solution for "number of labeled (graceful or otherwise) trees with *n* edges". This becomes apparent after encountering another interesting integer sequence called a *Prüfer sequence* (sometimes known as the *Prüfer code*). These sequences apply only to labeled trees, not graphs in general. However they need not have a graceful labeling. Instead, the only constraint is that the vertices are numbered {0, … , n} where *n* is the number of edges. (They work with 1-based sequences too, but I'm using 0-based sequences for consistency.)

Another important difference between Prüfer and Sheppard sequences: for a graceful tree with *n* edges, its Prüfer sequence has *n-1* terms while its Sheppard sequence has *n* terms.

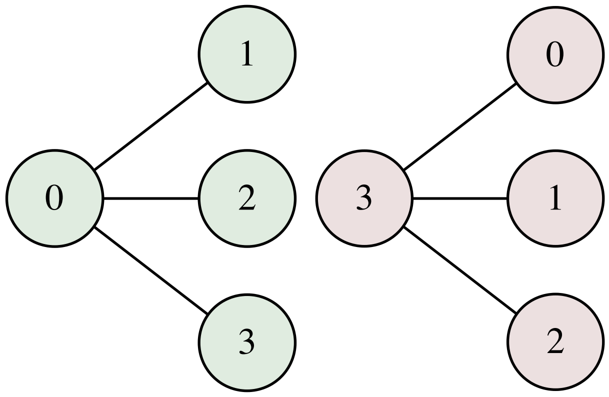
I won't go into detail about how to construct a Prüfer sequence, it's a fairly simple process that involves peeling off the leaf nodes, one by one. Constructing a tree from a sequence is also simple; consult wikipedia for details. The important thing to note is that a Prüfer sequence can be used to uniquely identify any labeled tree.

Gallery of graceful trees

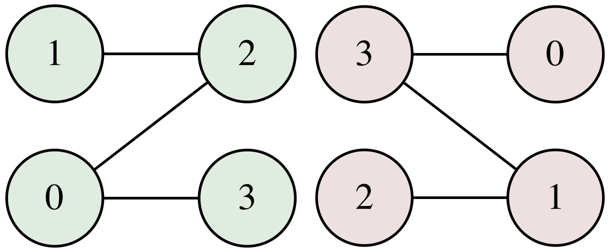
Just for fun, the entire remainder of this post is filled with SVG diagrams of graceful trees along with their corresponding Sheppard and Prüfer sequences. The graceful complement of each labeling is shown on the right. The gallery is limited to trees with 3 edges, 4 edges, and 5 edges.

The physical arrangement of vertices in all the following diagrams does not matter, but I find the non-determinism somewhat distracting, so I also created a large table of "sundial" diagrams on this page, which are a more deterministic way of drawing trees with graceful labeling.

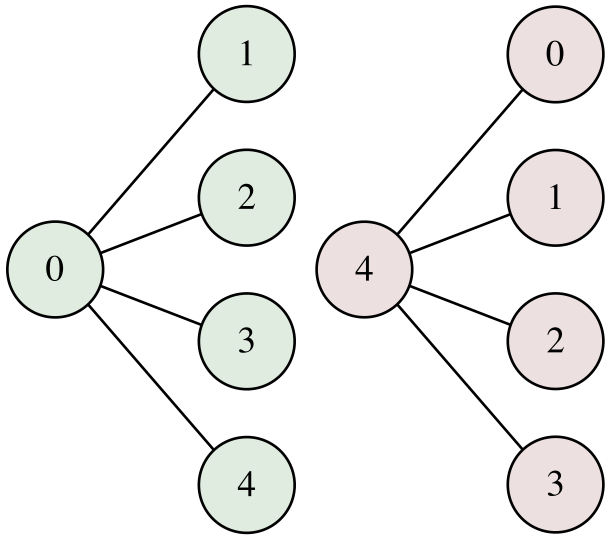
Sheppard: **(0, 0, 0)** and **(2, 1, 0)**
Prüfer: **(0, 0)** and **(3, 3)**



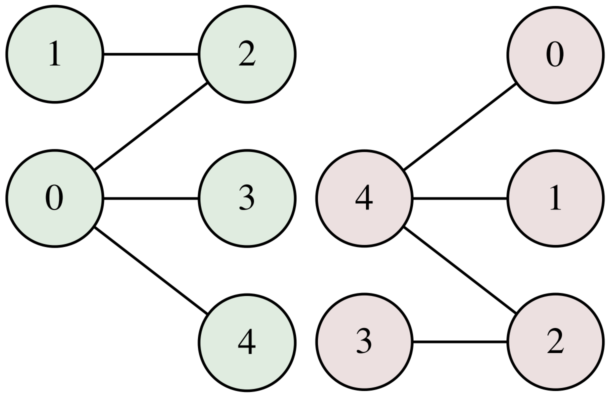
Sheppard: **(1, 0, 0)** and **(1, 1, 0)**
Prüfer: **(2, 0)** and **(3, 1)**



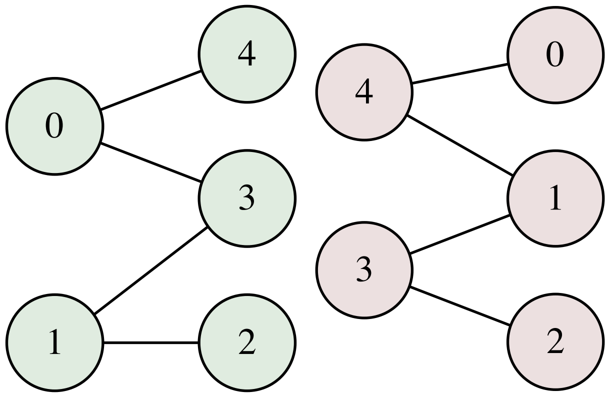
Sheppard: **(0, 0, 0, 0)** and **(3, 2, 1, 0)**
Prüfer: **(0, 0, 0)** and **(4, 4, 4)**



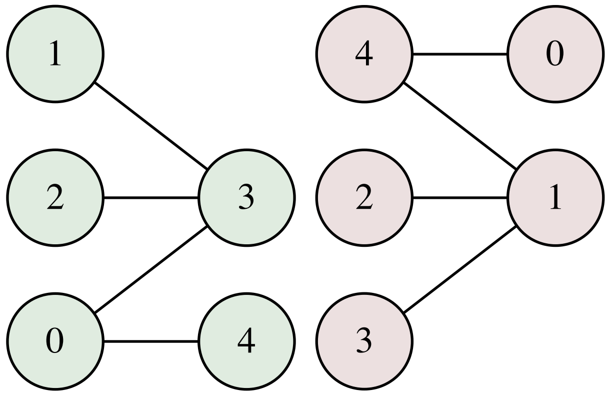
Sheppard: **(1, 0, 0, 0)** and **(2, 2, 1, 0)**
Prüfer: **(2, 0, 0)** and **(4, 4, 2)**



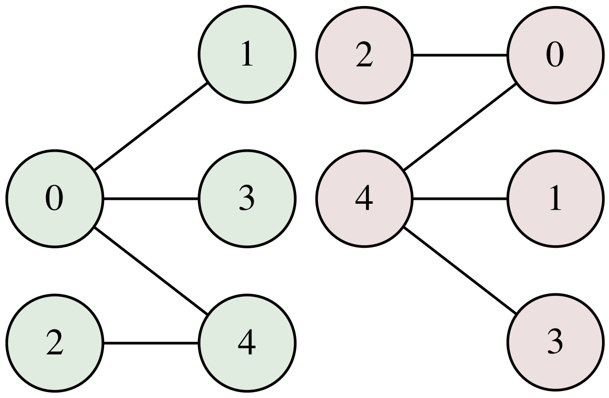
Sheppard: **(1, 1, 0, 0)** and **(2, 1, 1, 0)**
Prüfer: **(1, 3, 0)** and **(4, 3, 1)**



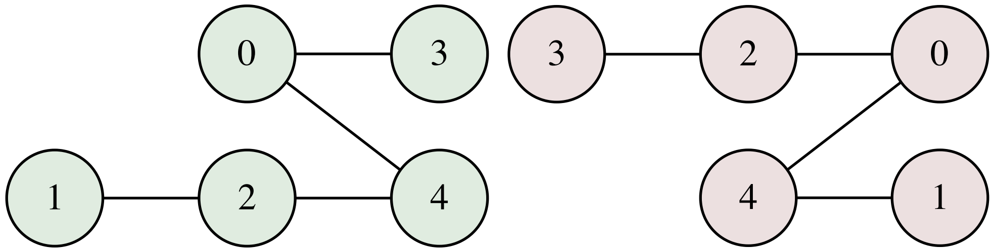
Sheppard: **(2, 1, 0, 0)** and **(1, 1, 1, 0)**
Prüfer: **(3, 3, 0)** and **(4, 1, 1)**



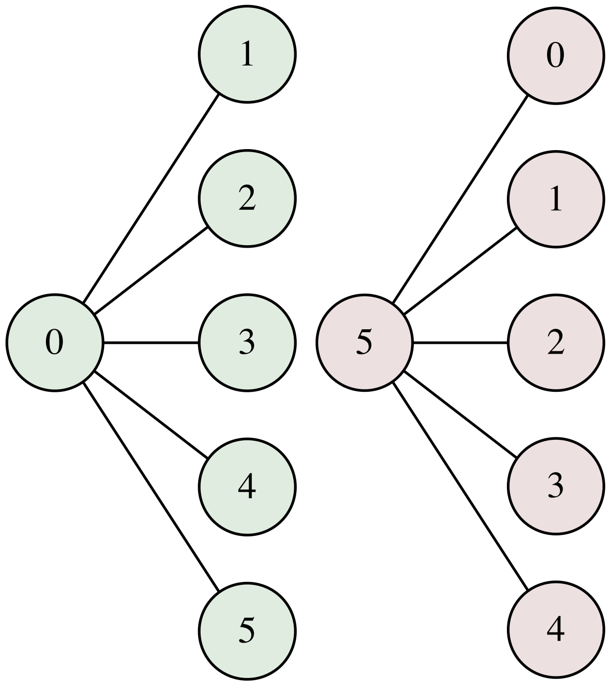
Sheppard: **(0, 2, 0, 0)** and **(3, 0, 1, 0)**
Prüfer: **(0, 4, 0)** and **(4, 0, 4)**



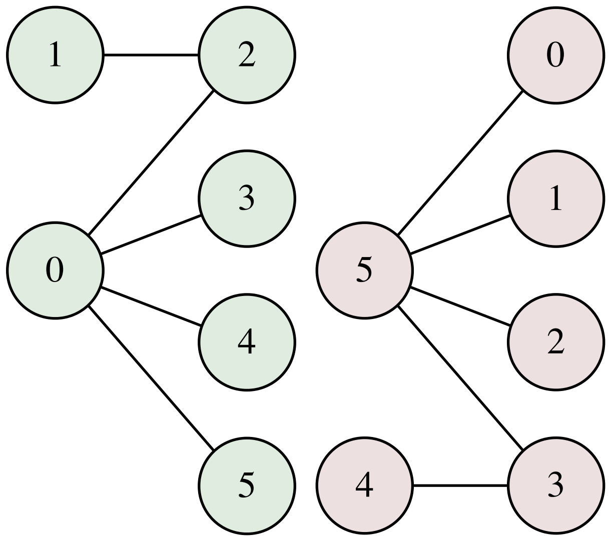
Sheppard: **(1, 2, 0, 0)** and **(2, 0, 1, 0)**
Prüfer: **(2, 4, 0)** and **(4, 2, 0)**



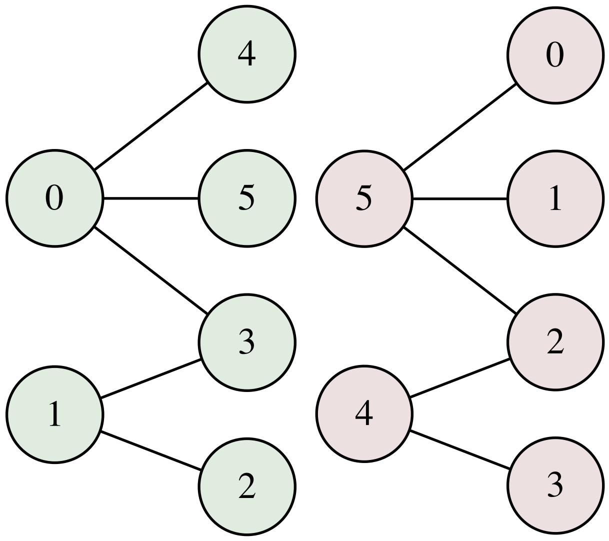
Sheppard: **(0, 0, 0, 0, 0)** and **(4, 3, 2, 1, 0)**
Prüfer: **(0, 0, 0, 0)** and **(5, 5, 5, 5)**



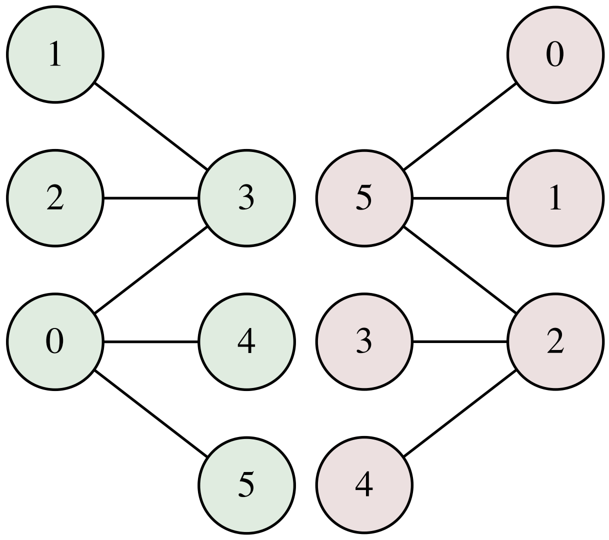
Sheppard: **(1, 0, 0, 0, 0)** and **(3, 3, 2, 1, 0)**
Prüfer: **(2, 0, 0, 0)** and **(5, 5, 5, 3)**



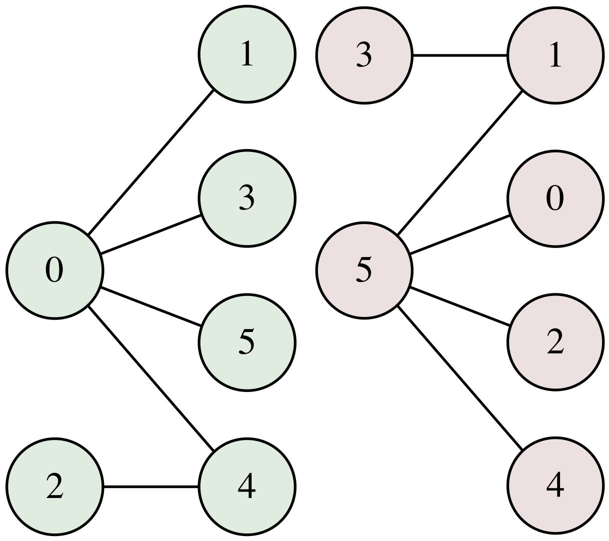
Sheppard: **(1, 1, 0, 0, 0)** and **(3, 2, 2, 1, 0)**
Prüfer: **(1, 3, 0, 0)** and **(5, 5, 4, 2)**



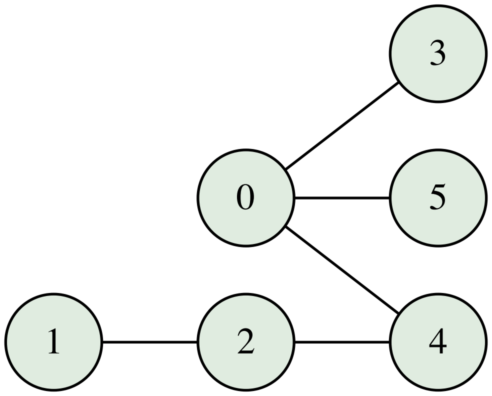
Sheppard: **(2, 1, 0, 0, 0)** and **(2, 2, 2, 1, 0)**
Prüfer: **(3, 3, 0, 0)** and **(5, 5, 2, 2)**

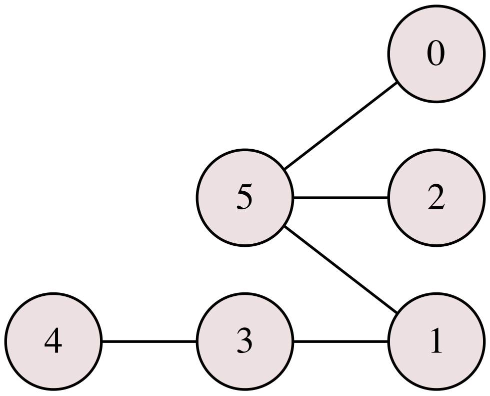


Sheppard: **(0, 2, 0, 0, 0)** and **(4, 1, 2, 1, 0)**
Prüfer: **(0, 4, 0, 0)** and **(5, 5, 1, 5)**

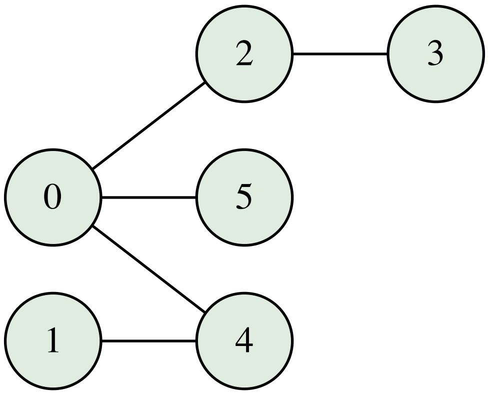


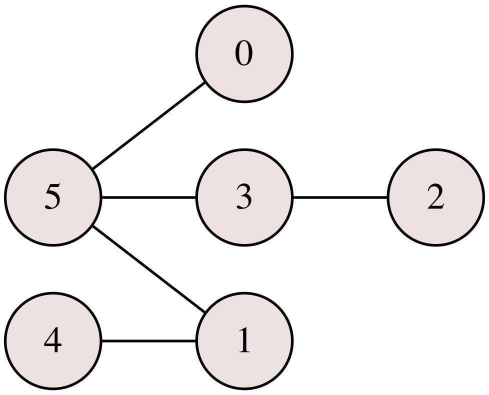
Sheppard: **(1, 2, 0, 0, 0)** and **(3, 1, 2, 1, 0)**
Prüfer: **(2, 4, 0, 0)** and **(5, 5, 3, 1)**



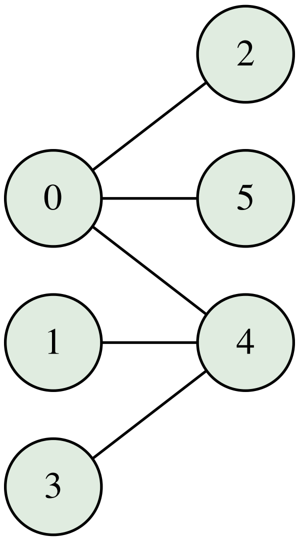


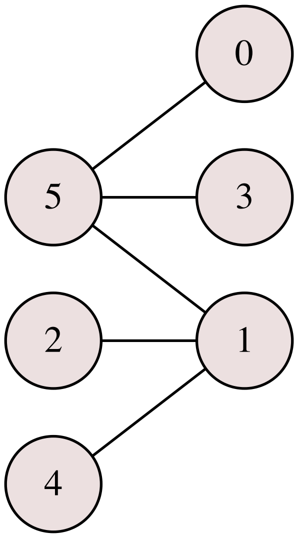
Sheppard: **(2, 0, 1, 0, 0)** and **(2, 3, 1, 1, 0)**
Prüfer: **(4, 2, 0, 0)** and **(5, 3, 5, 1)**



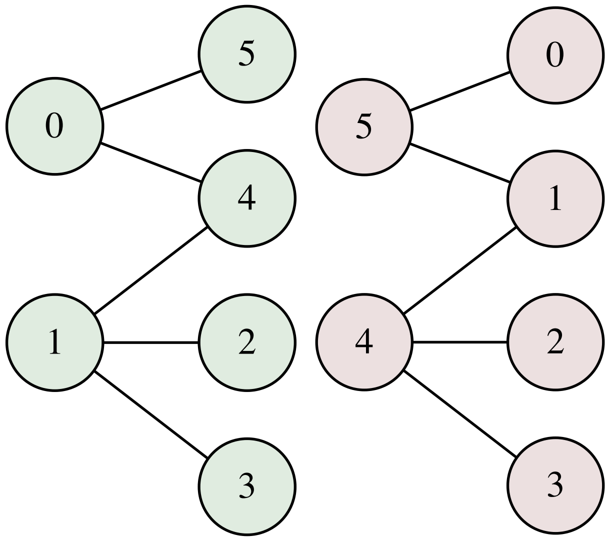


Sheppard: **(3, 0, 1, 0, 0)** and **(1, 3, 1, 1, 0)**
Prüfer: **(4, 0, 4, 0)** and **(5, 1, 5, 1)**

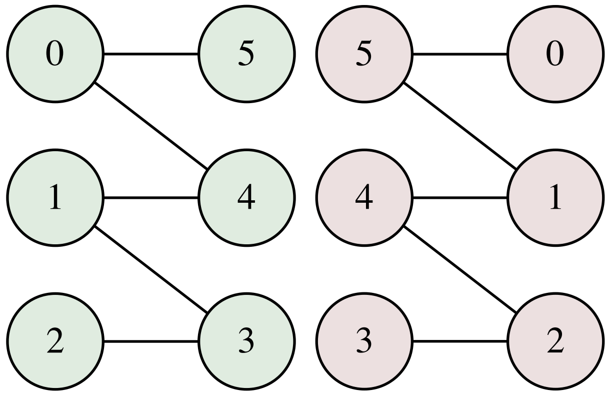




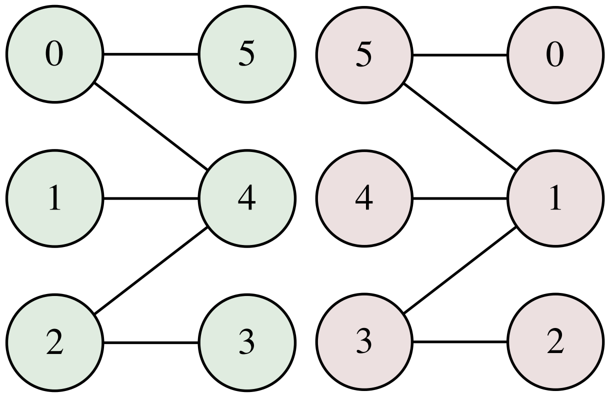
Sheppard: **(1, 1, 1, 0, 0)** and **(3, 2, 1, 1, 0)**
Prüfer: **(1, 1, 4, 0)** and **(5, 4, 4, 1)**



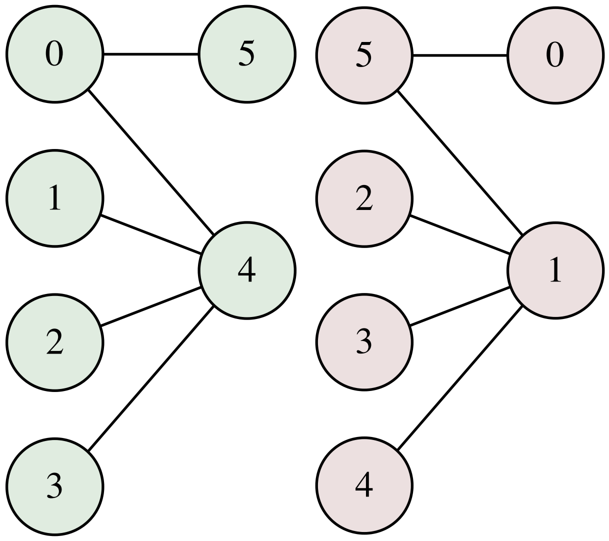
Sheppard: **(2, 1, 1, 0, 0)** and **(2, 2, 1, 1, 0)**
Prüfer: **(3, 1, 4, 0)** and **(5, 2, 4, 1)**



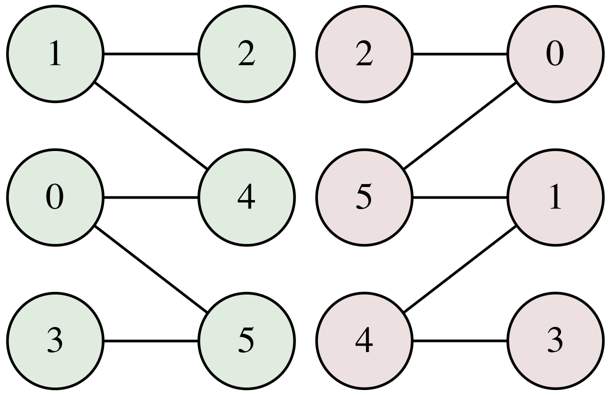
Sheppard: **(2, 2, 1, 0, 0)** and **(2, 1, 1, 1, 0)**
Prüfer: **(4, 2, 4, 0)** and **(5, 3, 1, 1)**



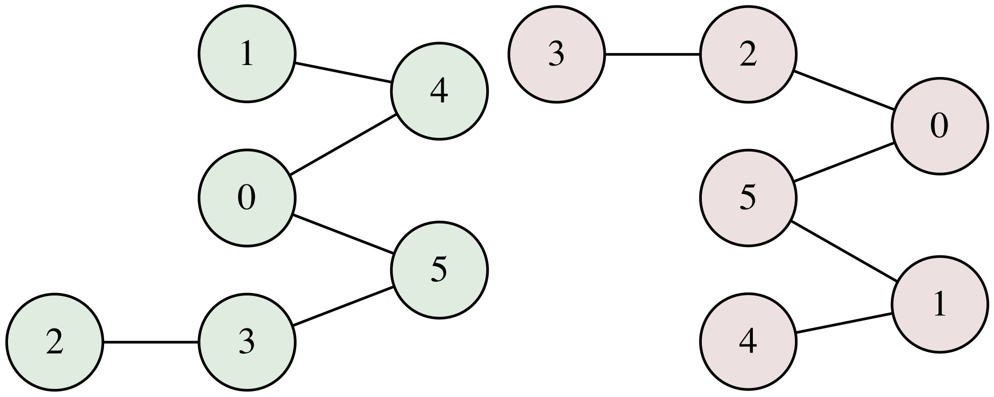
Sheppard: **(3, 2, 1, 0, 0)** and **(1, 1, 1, 1, 0)**
Prüfer: **(4, 4, 4, 0)** and **(5, 1, 1, 1)**



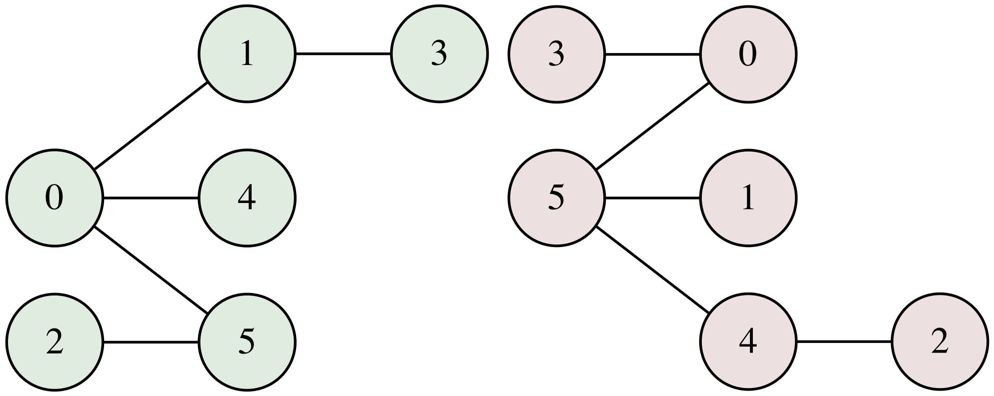
Sheppard: **(1, 3, 1, 0, 0)** and **(3, 0, 1, 1, 0)**
Prüfer: **(1, 4, 5, 0)** and **(0, 5, 4, 1)**



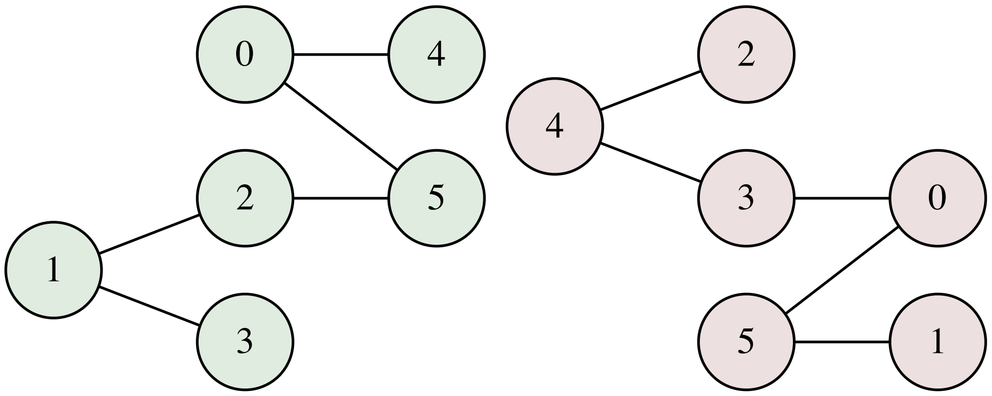
Sheppard: **(2, 3, 1, 0, 0)** and **(2, 0, 1, 1, 0)**
Prüfer: **(4, 3, 5, 0)** and **(2, 0, 5, 1)**



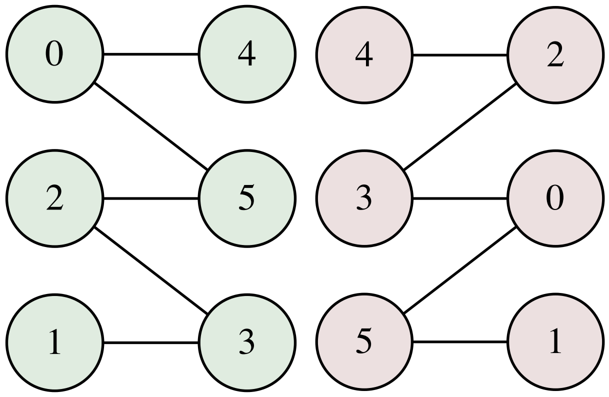
Sheppard: **(0, 1, 2, 0, 0)** and **(4, 2, 0, 1, 0)**
Prüfer: **(5, 1, 0, 0)** and **(5, 4, 0, 5)**



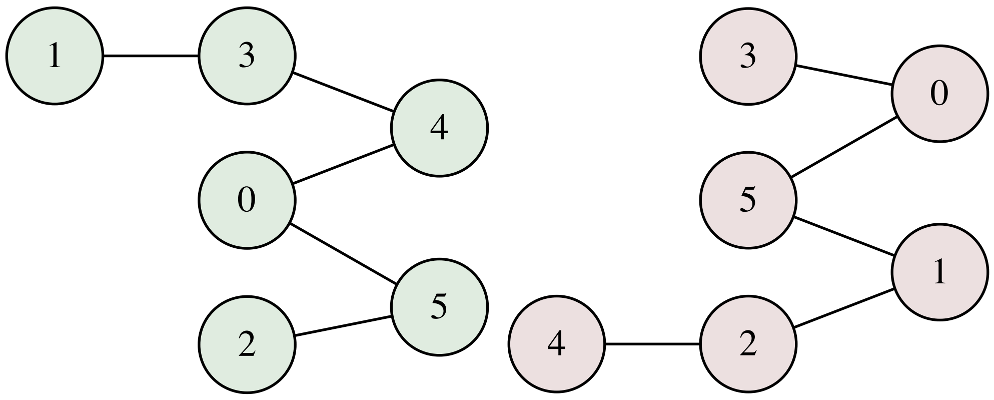
Sheppard: **(1, 1, 2, 0, 0)** and **(3, 2, 0, 1, 0)**
Prüfer: **(1, 2, 5, 0)** and **(5, 4, 3, 0)**



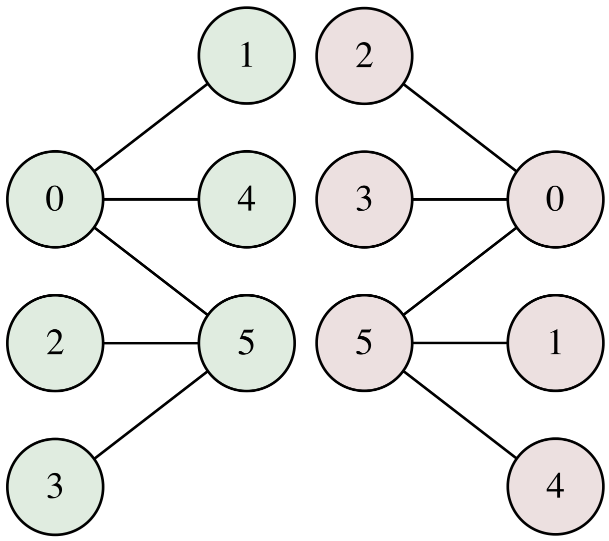
Sheppard: **(2, 1, 2, 0, 0)** and **(2, 2, 0, 1, 0)**
Prüfer: **(3, 2, 5, 0)** and **(5, 2, 3, 0)**



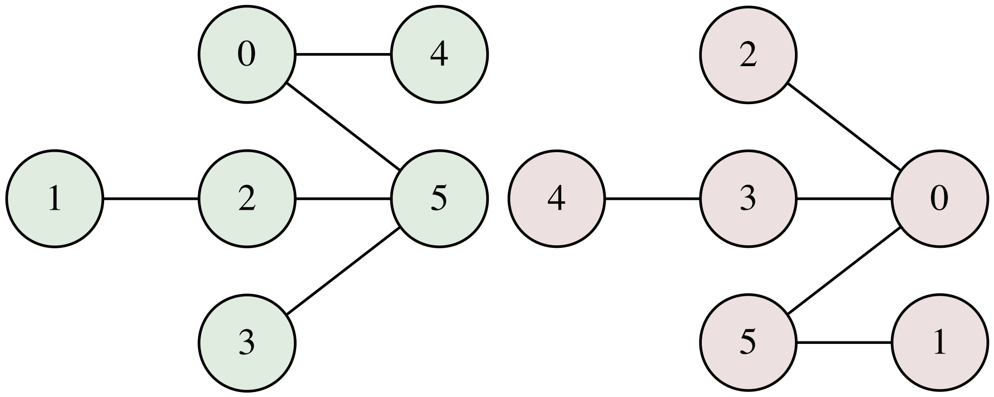
Sheppard: **(3, 1, 2, 0, 0)** and **(1, 2, 0, 1, 0)**
Prüfer: **(3, 5, 4, 0)** and **(0, 5, 2, 1)**



Sheppard: **(0, 3, 2, 0, 0)** and **(4, 0, 0, 1, 0)**
Prüfer: **(0, 5, 5, 0)** and **(5, 0, 0, 5)**



Sheppard: **(1, 3, 2, 0, 0)** and **(3, 0, 0, 1, 0)**
Prüfer: **(2, 5, 5, 0)** and **(5, 0, 3, 0)**



This post was written by Philip Rideout in March of 2020. If you want, you can download a pdf file.