Introduction

Definition 1.1: An undirected, simple graph $G = \{V, E\}$ is a set of vertices V, and a set of edges, E, where E is a set of unordered pairs of vertices from V such that for all $u \in V$, $\{u, u\} \notin E$.

Example 1.1: GRAPHS

Remark: Graphs may have any number of vertices. However, unless otherwise specified, suppose any graph presented in this paper has a finite vertex set.

Definition 1.2: For a graph $G = \{V, E\}$, if $\{u, v\} \in E$, we say u and v are *adjacent* or *neighbors*.

Definition 1.3: Let u be a vertex in a graph G. The *neighborhood of* u in G, denoted $\mathcal{V}(u)$ is the set of neighbors of u. The cardinality of $\mathcal{V}(u)$ is the *degree* of the vertex u.

Example 1.2: adjacent and non-adjacent vertices.

Definition 1.4: Two graphs, $G = \{V, E\}$ and $G' = \{V', E'\}$, are *isomorphic*, denoted $G \cong G'$, if there exists a bijection $f : V \to V'$ such that for every $u, v \in V$, $\{u, v\} \in E$ if and only if $\{f(u), f(v)\} \in E'$. If such a function f exists, we call f a *graph isomorphism* from G to G'.

Example 1.3: One way to think about two graphs being isomorphc is that two graphs are isomorphic if the vertices of one graph can be rearrenged without adding or breaking any edges to produce the other graph. See Figures 1.1 and 1.2.

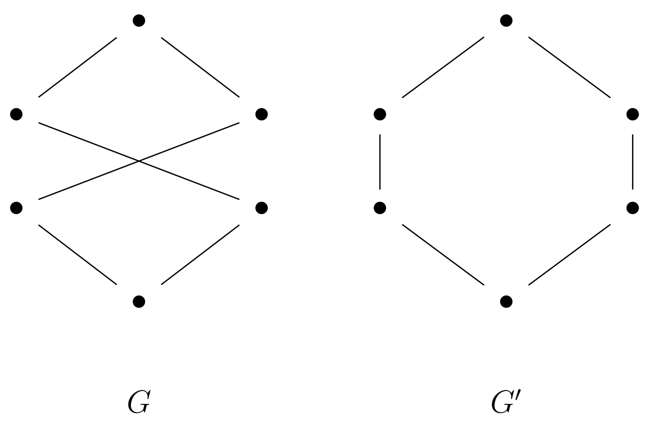


Figure 1.1: $G \cong G'$

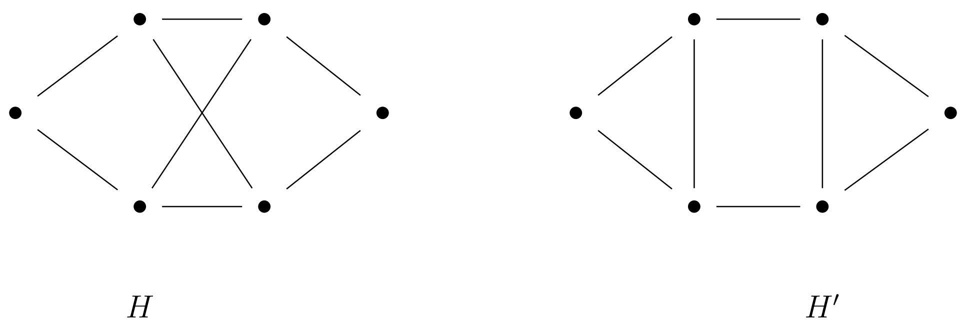
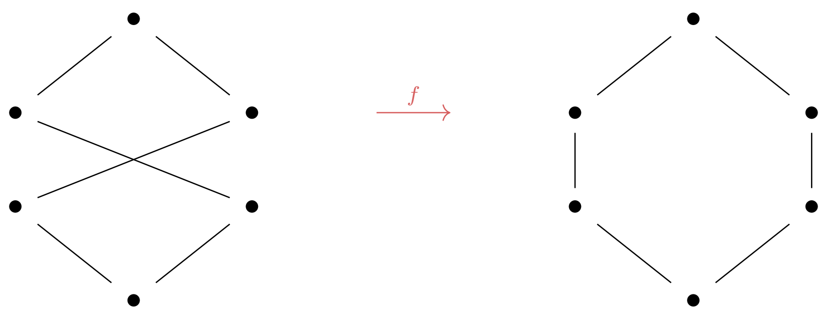


Figure 1.2: H is not isomorphic to H'

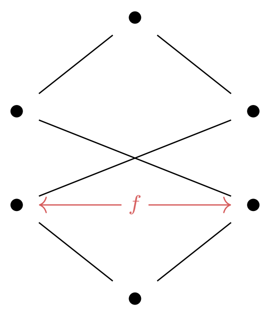
1.1 The Graph Isomorphism Problem

Definition 1.5: Given two graphs $G = \{V, E\}$ and $G' = \{V', E'\}$, the *graph isomorphism problem* is to determine if there exists a graph isomorphism from G to G'.

Example 1.4: The function f is a graph isomorphism.



One possible defintion for f is shown below, where only two vertices are swapped.



The graph isomorphism problem (GI) has a straightforward statement. Given any two graphs, $G = \{V, E\}$ and $G' = \{V', E'\}$, solving the problem requires proving the existence of a graph isomorphism from V onto V' satisfying the conditions in Definition 1.3 or showing no such function exists. There are are some instances where solving the GI is as simple as the statement of the problem. If $V = V$ and $E = E'$, then the identity map is a graph isomorphism. On the other hand, there are many scenarios where it is easy to tell that a graph isomorphism between two graphs cannot exist. Since it is impossible to produce a bijection between two sets of different finite sizes, if $|V| \neq |V'|$ then G is not isomorphic to

G'. One way to think of a graph isomorphism is that it is a bijection between two graphs that preserves neighborhood relations. Thus, if G is a graph with two vertices of degree 3 while G' only has a single vertex of degree 3, G cannot be isomorphic to G'. Nearly all graph properties we care about must be shared between two graphs for there to exist a graph isomorphism between them. To understand another property of graphs that must be preserved by a graph isomorphism, the following definition is useful.

Definition 1.6: A *walk* is a list $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges such that for $1 \leq i \leq k$, the edge e_i has endpoints v_{i-1} and v_i. (West, 2002)

A $u, v-$*path* in a graph $G = \{V, E\}$ is a walk from vertex u to vertex v that has no repeated vertices. A *n-cycle* is a path with n edges that starts and ends at the same vertex. A graph is *connected* if there exists a $u, v-$path for all $u, v \in V$. If G is connected and G' is not connected, then G and G' are not isomorphic. The length of a $u, v-$path is the number of edges it contains. Thus, the *longest* $u, v-$path in a graph is the one containing the most edges where u and v can be any vertices in G. If the longest path in G is not the same length as that of G', then G and G' are not isomorphic. Another example is if G has a 3-cycle while G' does not, then G and G' are not isomorphic. When the given graphs share common properties (e.g.,connected) and standard measures (e.g., longest path), solving the GI solution is more difficult. In fact, the computational complexity of GI is currently unsettled. The most recent advance is Babai (2018), which established a *quasi-polynomial time*, $n^{(\log n)^{\mathcal{O}(1)}}$, isomorphism test, where n denotes the number of vertices of the input graphs. Although GI is thought to be of polynomial time complexity, Babai's bound is the best established. Luks (1982) established a divide and conquer method, which laid the groundwork for Babai's (2018) result. Luks' paper shows that GI is polynomial time reducible to the *color automorphism problem.*

On the other hand, there are tools for efficiently solving GI for two graphs, even if they have thousands of vertices (see Nauty). There are also algorithms that are useful for giving a sense as to how similar two graphs are, even if they are not isomorphic. The next section is dedicated to such an algorithm, the Weisfeler Lehman Algorithm (WL). WL is also successful on nearly all types of graphs at determining that two graphs are not isomorphic. The ideas WL uses are present at the core of nearly every practical grpah isomorphism solver as well as in Babai's breakthrough algorithm.

Weisfeler Lehman Algorithm

The Weisfeler-Lehman Algorithm (WL) is an isomorphism test that makes use of a method created by Weisfeler and Leman (1968) for reducing a graph to a canonical form, where canonical refers to a canonical labelling of the vertices of the graph.

Definition 2.7: Given a graph $G = \{V, E\}$, a *canonical form* of G is a partial ordering of V such that for all $u, v \in V$, if u and v are incomparable, then there exists a graph automorphism moving u to v. (Weisfeler, Leman 1968)

The reason canonical form is useful is that first, representing a graph in canonical form preserves the structure of the graph. That is, if graphs are not isomorphic when each is reduced to canonical form, then they were not isomorphic to begin with. Second, it is straightforward to check that two graphs are not isomorphic once they are both in canonical form, which will become clear shortly. To reduce a graph to canonical form, Weisfeler and Leman (1968) propose the following procedure iterative procedure.
First, each vertex u is labelled by the number of neighbors it has. These neighborhood labels determine the initial class for each vertex. Note that there is a natural order on these classes, as they are integers. Next, generate a characteristic vector for each vertex v by $v_u = (l, v_{u1}, v_{u2}, \dots)$, where v_{uk} is the number of neighbors of vertex u from class k and l is the number of the class which contains vertex u (Weisfeler, Leman 1968). Next, divide the vertices into classes once again based on their new characteristic vectors, ordered lexicographically. Iterate this process until all vertices belong to different classes or the classes stabilize, meaning subsequent iterations do not change the classes. In either case, a canonical labelling is produced. We formalize this in the following algorithm.

WL Canonical Labeling

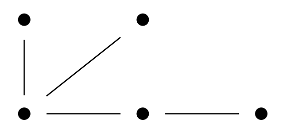
Input: $G = \{V, E\}$

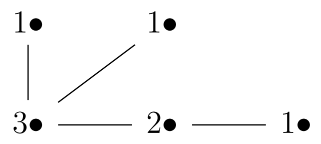
Initialize: Label vertices using degree

Iterate: Produce characteristic vectors, sort, relabel

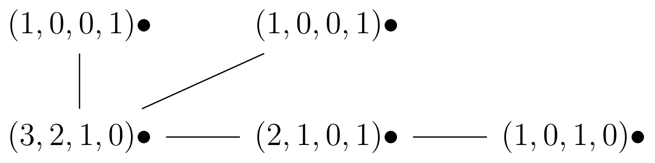
Output: Relabeled graph $G$with a canonical labelling on the labels of the vertex set

Example 2.5: Given the graph G below, we follow WL canonical labelling until the classes stabilize.

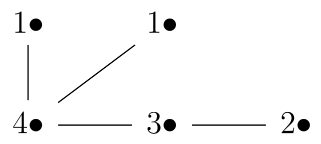




We now generate the characteristic vector v_u for each vertex u.



Next, each vertex is relabeled based on the class of its characteristic vector, ordered lexicographically.



We now iterate, producing v_u for each vertex u.

$$\begin{tikzcd}
(1,0,0,0,1)\bullet \arrow[d, dash] & (1,0,0,0,1)\bullet & \\
(4,2,0,1,0)\bullet \arrow[r, dash] \arrow[ur, dash] & (3,0,1,0,1)\bullet \arrow[r, dash] & (2,0,0,1,0)\bullet
\end{tikzcd}$$

Last, we relabel based on the characteristic vectors.

$$\begin{tikzcd}
1\bullet \arrow[d, dash] & 1\bullet & \\
4\bullet \arrow[r, dash] \arrow[ur, dash] & 3\bullet \arrow[r, dash] & 2\bullet
\end{tikzcd}$$

Notice that this graph is equivalent to the relabeled graph from the previous iteration. The classes have stabilized, so the algorithm terminates. Weisfeler and Leman prove in their (1968) that if two vertices $u, v \in V$ are in the same class when the algorithm terminates, then there exists an automorphism sending u to v. In this case, the partial ordering of V given by such classes is a canonical form by Definition 2.7. On the other hand, if each vertex has a unique class when the algorithm terminates, then the partial ordering of V given by the classes leaves no incomparable vertices. In this case, the ordering on V given by the classes is a canonical form as well, satisfying Definition 2.7 trivially.

Remark: Since WL iteratively produces partitions of vertex sets, it is also called *color refinement*. The color of a vertex is simply its class at any given iteration.

Without much work, this canonical labelling algorithm can be transformed into an isomorphism test, call it the *Naive Weisfeler Leman Isomorphism Test*. This test takes two graphs as input, then computes a canonical labelling of the graphs at the same time, pooling the class count. If the graphs have different canonical labellings, then the graphs are not isomorphic.

Naive WL Isomorphism Test

Input: Graphs $G = \{V, E\},\ G' = \{V', E'\}$

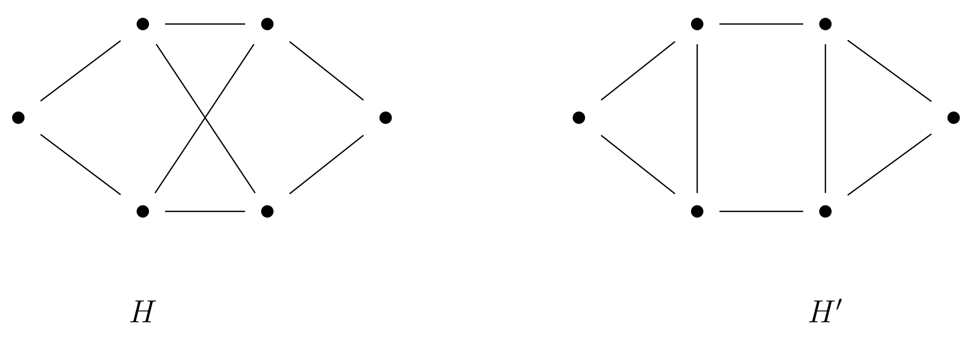
Initialize: Label the vertices using degree

Iterate: Produce characteristic vectors for all vertices in G and $G',$ order all vectors, relabel

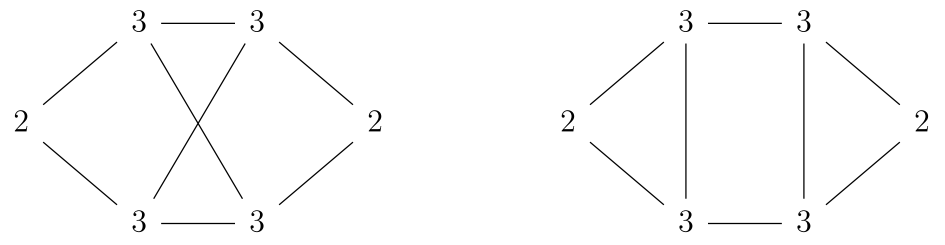
Output: A graph isomorphism from G to G' exists

or the existence of such a function has not been ruled out.

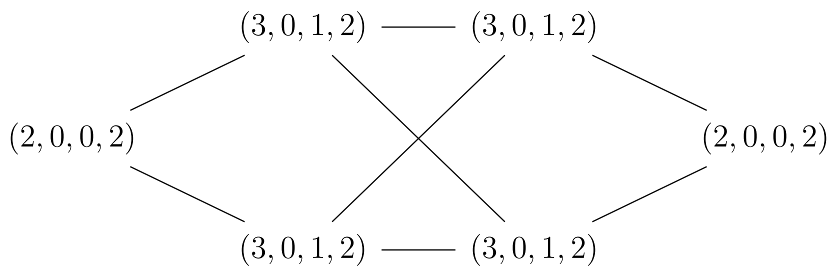
Example 2.6: This example applies the Naive WL Isomorphism Test to H and H'.

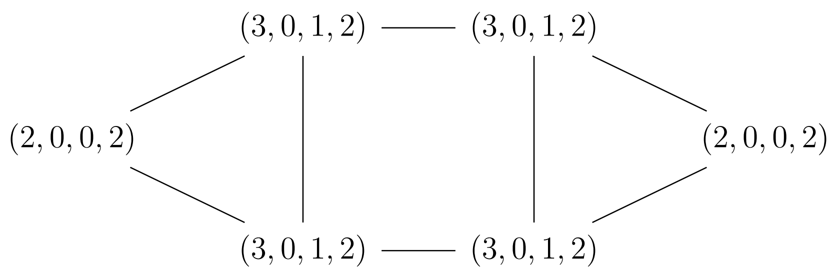


First, label the vertices by their degree.

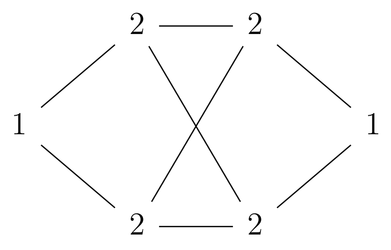


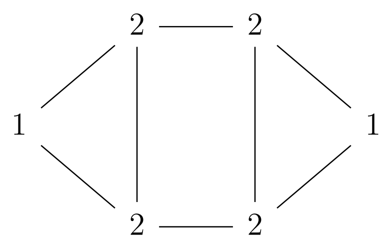
Next, produce characteristic vectors.



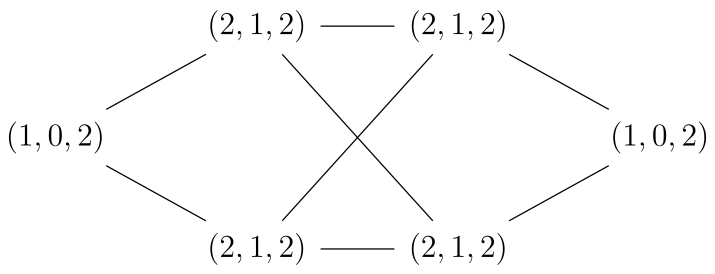


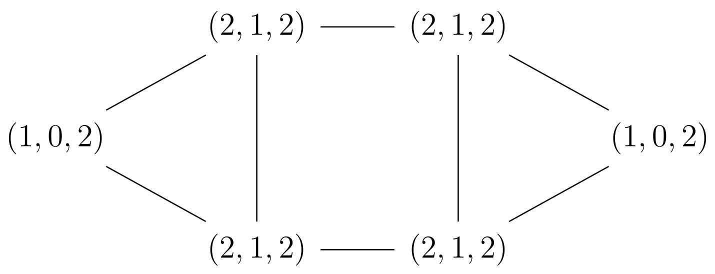
Order the vectors and relabel.

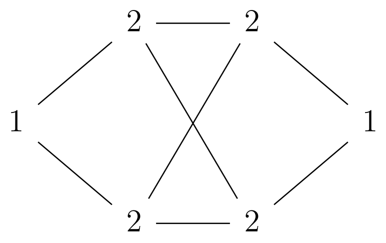


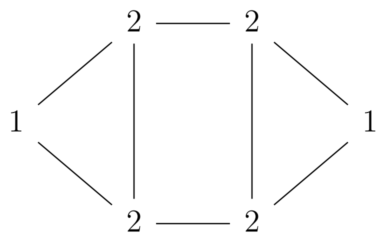


Iterate.









Since the classes did not divide, the algorithm terminates. The naive WL isomoprhism test was unable to determine if H and H' are not isomorphic.

Recall, however, that H' has a 3 cycle while H does not. Thus, H and H' are not isomorphic but the Naive WL Isomorphism test was unable to determine this. In such a case, we say that the Naive WL isomorphism Test does not *distinguish* H and H'.

References