1 Isomorphism

Two graphs: $G = (V, E)$ and $G' = (V', E')$ are called isomorphic if there is a one-to-one mapping $f$ from $V$ onto $V'$ such that any two vertices $v_i, v_j \in V$ are adjacent iff $f(v_i)$ and $f(v_j)$ are adjacent. We would say that $G$ is isomorphic to $G'$, or $G \cong G'$. By permutating the rows of the adjacency matrix of $G$ ($A$), we should be able to create the adjacency matrix of $G'$ ($A'$); e.g., there exists a permutation matrix $P$ such that $PAP^T = A'$.

If $G(V, E)$ and $G'(V', E')$ are isomorphic, then

1. $|V| = |V'|$ and $|E| = |E'|$;
2. the degree sequences of $G$ and $G'$ sorted in the non-increasing order are identical;
3. $G$ has a cut vertex iff $G'$ does.
4. the lengths of the shortest cycles in $G$ and in $G'$ are equal.

Note that properties (1) - (4) are necessary but not sufficient.

The isomorphism relation on the set of ordered pairs from $G$ to $G'$ is:
reflexive: $G \cong G$
symmetric: if $G \cong H$, then $H \cong G$
transitive: if $G \cong H$ and $H \cong J$, then $G \cong J$

An isomorphism class is an equivalence class of graphs that are under the isomorphic relation.

An automorphism is an isomorphism from $G$ to itself. E.g. think of a clique.

Sometimes we may talk about the subgraph isomorphism problem, which is: Given a graph $G$ and a graph $H$ of equal or smaller size of $G$, does there exist a subgraph of $G$ that is isomorphic to $H$? Subgraph isomorphism and related problems (subgraph counting: how many different subgraphs of $G$ are isomorphic to $H$? subgraph enumeration: what are those subgraphs of $G$ that are isomorphic to $H$?) are common techniques of graph mining.

2 Decomposition and Special Graphs

The complement $\overline{G}$ of a graph $G$ has $V(G)$ as its vertex set. Two vertices are adjacent in $\overline{G}$ iff they are not adjacent in $G$. A graph $G$ is called self-complementary if $G$ is isomorphic to $\overline{G}$.
There are a couple specific names for certain graphs that we might use repeatedly:

**Triangle:** A three vertex cycle $C_3$ or clique $K_3$

**Claw:** The complete bipartite graph $K_{1,3}$

Note: the claw is also a **star graph**, which are the class of complete bipartite graphs $K_{1,n}$

Also note: the book gives several other examples in 1.1.35; we likely won’t be talking much specifically about the other ones.

The **Peterson Graph** is the simple graph with vertices from all 2-element subsets of a 5-element set and with edges formed by connecting pairs of disjoint 2-element subsets.

Prove: If two vertices are nonadjacent in the Peterson graph, then they have exactly one common neighbor.

Prove: The shortest cycle in the Peterson graph (its **girth**) is a 5-cycle.

### 3 Walks and Connectivity

A **walk** is a list of of vertices and edges (e.g., $v_0, e_5, v_6, e_1, v_2$) such that each listed edge connects the preceding and proceeding listed vertices. The list begins and ends with vertices. A **trail** is a walk with no repeated edges. A **path** has no repeated edges or vertices. A $u, v$-walk and $u, v$-trail begin with vertex $u$ and end with vertex $v$. A $u, v$-path is a path with endpoint vertices $u$ and $v$ having degree 1 and all other vertices being internal. The **length** of a walk/trail/path is the number of contained edges. A walk is **closed** if the start and end vertices are the same.

A graph $G$ is **connected** if for every $u, v \in V(G)$ there is a path connecting $u$ and $v$. Otherwise $G$ is disconnected. A **connected component** of $G$ is a maximal connected subgraph. A **cut-edge** or **cut-vertex** are the edges or vertices that, when removed from $G$, increase the number of connected components.

An edge is a cut-edge iff it belongs to no cycle.

We’ll talk about connectivity a bit more in-depth later in the course.

### 4 Graph Traversal

Computationally, graph traversal refers to the visitation of each vertex in a graph. The most common means of traversing a graph are through **breadth-first search** (BFS) or **depth-first search** (DFS) starting from some **root**. Graph traversal forms the basis of numerous connectivity decomposition algorithms. Refer to the 1ec02.cpp example code.
where we implement BFS and DFS and use them to determine connectivity on an input graph. We’ll use the following basic algorithm:

\[
\begin{align*}
&c \leftarrow 0 & \triangleright \text{number of connected components} \\
&\text{for all } v \in V(G) \text{ do} \\
&\quad visited(v) \leftarrow \text{false} \\
&\text{for all } v \in V(G) \text{ do} \\
&\quad \text{if } visited(v) = \text{false} \text{ then} \\
&\quad \quad X \leftarrow \text{traverse}(G, v) & \triangleright \text{find all vertices reachable from } v \\
&\quad \quad \text{for all } u \in X \text{ do} \\
&\quad \quad \quad visited(u) \leftarrow \text{true} \\
&\quad c \leftarrow c + 1
\end{align*}
\]