2.1 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 $G \cong G'$. Isomorphism can also be extended to non-simple graphs, through adding the language that there also must exist a one-to-one mapping from E to E', where every edge $v_i, v_j \in E$ must map to an edge $(f(v_i), f(v_j) \in E')$.

By permuting the rows of the adjacency matrix of G (A), we should be able to create the adjacency matrix of G' (A'); i.e., there exists a permutation matrix P such that $PAP^T = A'$.

If $G(V, E)$ and $G'(V', E')$ are isomorphic, then we can make general statements such as:

1. $|V| = |V'|$ and $|E| = |E'|$
2. the **degree sequences** of G and G' sorted in non-increasing order are identical
3. the lengths of the longest shortest paths (a graph's **diameter**) in G and G' are equal
4. the lengths of the shortest cycles (a graph's **girth**) in G and in G' are equal

Note that properties (1) - (4) are necessary but not sufficient conditions for isomorphism. We'll discuss more about what this means soon.

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 all 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. We also might want to differentiate between **vertex-induced** subgraphs and **non-induced** subgraphs. For both, we consider a subgraph S of G that contains a set of vertices $V(S) \in V(G)$. We would say that the subgraph is induced if $\forall e(u, v) \in E(G)$

s.t. $u, v \in V(S) \implies e(u, v) \in E(S)$. The subgraph is non-induced if it only contains a subset of the edges $e(u, v) \in E(G)$ s.t. $u, v \in V(S)$.

In terms of computational complexity, graph isomorphism is thought to be solvable in quasi-polynomial time [$exp((\log n)^{O(1)})$, Babai 2015], though it remains an open problem. Subgraph isomorphism is NP-complete. A naive algorithm for subgraph isomorphism would involve exhaustively checking the local neighborhood of all $v \in V(G)$ for an isomorphic relation to H, and requires $O(n^k)$ time, where $k = |V(H)|$.

2.2 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 $\overline{G}$. A **decomposition** of a graph is a list of subgraphs such that each edge appears in only a single subgraph in the list.

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 probably won't be talking much specifically about the other ones.

2.3 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. *Random walks* performed by starting at a given vertex, moving to an adjacent vertex selected at random, then iteratively continuing this procedure from the newly selected vertex have a number of interesting uses and properties; we'll talk a little more about these later.

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. We say a component is **trivial** when it consists of a single vertex and no edges. Otherwise, the component is **nontrivial**. A **cut-edge** or **cut-vertex** are the edges or

vertices that, when removed from G, increase the number of connected components.

We'll talk more about connectivity later in the course.

2.4 Induction

In this class, we'll often be proving properties about graphs using **induction** and **necessity and sufficiency**.

Review: **Weak Induction** as a proof method. Consider a natural number n, let $P(n)$ be a mathematical statement. If properties 1 and 2 below hold, then $P(n)$ is true for all $n \in \mathbb{N}$.

1. $P(1)$ is true
2. for $k \in \mathbb{N}$, if $P(k)$ is true, then $P(k+1)$ is true

1 is the **basis step** and 2 is the **induction step**. The inductive step includes our **induction hypothesis**, which is the assumption that our current step of $P(k)$ is true. The basis step might utilize $P(0)$ or multiple k for $P(k)$.

Weak inductive proofs then work to show that if e.g. $P(1)$ is true, and $P(k) \implies P(k+1)$ is true, then $P(1) \implies P(1+1)$, $P(1+1) \implies P(1+1+1)$, etc. So $P(k)$ is true for all natural numbers.

Prove that: $2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 2$

Basis Step: $P(n = 1) = 2^1 = 2^2 - 2 = 2$ ✓
Induction Step: $P(n = k + 1) = 2^1 + 2^2 + \ldots + 2^k + 2^{k+1}$
$= [2^1 + 2^2 + \ldots + 2^k] + 2^{k+1}$
$= 2^{k+1} - 2 + 2^{k+1}$
$= [2^{k+1} + 2^{k+1}] - 2$
$= 2 \times 2^{k+1} - 2$
$= 2^{k+2} - 2$
$= 2^{(k+1)+1} - 2$
$= 2^{n+1} - 2$ ✓

However, a number of graph proofs require **Strong Induction**:

1. $P(1)$ is true
2. for $k > 1$, if $P(k)$ is true, then $P(n)$ is true for $1 \leq k < n$

Instead of limiting ourselves to $P(n = k + 1)$ in our inductive step, we assume that all $P(k)$ less than our n are true. When we're working with graphs, we often do induction on the number of vertices/edges in some graph G. With strong induction, we can establish a relation between $G(k)$ and $G(n)$ as the difference of a larger subgraph beyond just a single vertex or a single edge. This opens doors for greater proof possibilities; however, a lot of the proofs we do can still be accomplished with weak induction.

Necessity and sufficiency is often used to prove *equivalence relationships*, such as we'll soon show that a graph is bipartite iff it has no odd cycle. To more generally prove an equivalence relationship, we can show that the given conditions are both necessary and sufficient; i.e., by proving that if condition A implies condition B and if conditions B implies condition A, we prove their equivalence. For any equivalence relationship, knowledge about one condition gives us knowledge about the other condition – so if we know a graph has no odd cycles, we also know that it therefore must be bipartite.

2.5 More on Walks and Cycles

An **even** walk/path/trail/cycle has an even length, or number of edges. Likewise, an **odd** walk/path/trail/cycle has an odd length, or number of edges. An **even graph** has all vertex degrees even. A vertex is even if it has an even degree or odd when it has an odd degree.

Prove with induction: Every closed odd walk contains an odd cycle.

Basis Step: $P(l = 1)$: a length 1 walk is a single loop, hence is an odd cycle of length 1
Induction Step: $P(l = n > 1)$: We use the induction hypothesis to assume walks of length $k < n$ have an odd cycle. Consider walk W. If W is odd and has no repeated vertices, then W is an odd cycle by itself. Otherwise, some vertex v is repeated. Consider breaking W into walks W_1 and W_2 originating from v. As the length of W is odd, then W_1 must be odd and W_2 even. We say that the length of W_1 is $k < n$, which by our inductive hypothesis must have some odd cycle. As W_1 is a subpath of W, then W must also contain an odd cycle. This is the power of strong induction in action!!!

Prove with necessity and sufficiency: A graph is bipartite iff it has no odd cycle.

2.6 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) algorithms starting from some **root**. Graph traversal forms the basis of numerous connectivity decomposition algorithms, and is an important subroutine

for a number of other graph analytical methods. Refer to the `lec02.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:

```
c ← 0                                          ▷ number of connected components
for all v ∈ V(G) do
    visited(v) ← false
for all v ∈ V(G) do
    if visited(v) = false then
        X ← traverse(G, v)                     ▷ find all vertices reachable from v
        for all u ∈ X do
            visited(u) ← true
        c ← c + 1
```

It is straightforward to see that the complexity of traversal is $O(n + m)$. Using traversal, **we can solve basic graph problems**, such as determining the number and sizes of connected components, evaluating if a graph is bipartite, or determining if a graph is acyclic or a tree.