1 Monday, March 30

1.1 Rosen 10.2: Graph Terminology and Types of Graphs

Reading: Rosen 10.2, Ducks 3.5, LLM

Def: Let G and H be graphs. We say H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Ex: $\overline{K_n} \subseteq C_n \subseteq K_n$.

Ex: Using the set definitions, $V(W_n) = \{0, 1, \dots, n\}$, so $W_n \not\subseteq K_{n+1}$. However, a graph that *looks like* W_n does exist inside K_{n+1} (let $n+1$ take the place of 0). We will later talk about *isomorphism* that deals with this.

One way to think about subgraphs is that we can delete some number of vertices and edges from G in order to find the graph H.

Def: Let G be a graph and $S \subseteq V(G)$. The *subgraph of G induced by S*, denoted $G[S]$, is the graph H with $V(H) = S$ and $E(H) = \{uv : uv \in E(G), u, v \in S\}$. That is, $G[S]$ has vertices in S and all edges that are in G with both endpoints in S. A graph H is an *induced subgraph* of G (sometimes denoted $H \leq G$) if $H = G[S]$ for some $S \subseteq V(G)$.

Ex: Let $1 \leq n \leq m$. Then $K_n = K_m[\{1, \dots, n\}]$ and hence $K_n \leq K_m$.

Ex: C_n is not an induced subgraph of K_n.

Ex: $C_n = W_n[\{1, \dots, n\}]$, so C_n is an induced subgraph of W_n.

One way to think about induced subgraphs is that we can delete some number of vertices (and only the edges incident to those vertices) to find H.

Thm: If G is a bipartite graph and H is a subgraph of G, then H is bipartite.

Proof. Since G is bipartite, there is a bipartition $V(G) = X \cup Y$ such that $X \cap Y = \varnothing$ and for every edge $e \in E(G)$, exactly one endpoint of e is in each X and Y. Since H is a subgraph of G, $V(H) \subseteq V(G) = X \cup Y$. Let $A = X \cap V(H)$ and $B = Y \cap V(H)$. Then $V(H) = A \cup B$. Let e be an edge in $E(H)$. Then since $E(H) \subseteq E(G)$, exactly one endpoint of e is in $X \cap V(H) = A$; exactly one endpoint is in $Y \cap V(H) = B$. Thus, $A \cup B$ is a bipartition of H and H is bipartite. □

Def: Let G_1 and G_2 be graphs. The *union* of G_1 and G_2, denoted $G_1 \cup G_2$, is the graph G' where $V(G') = V(G_1) \cup V(G_2)$ and $E(G') = E(G_1) \cup E(G_2)$.

Def: Let G_1 and G_2 be graphs. The *disjoint union* of G_1 and G_2, denoted $G_1 + G_2$, is the graph G' where $V(G') = \{(i, v) : i \in \{1, 2\}, v \in V(G_i)\}$ and $E(G') = \{\{(i, u), (i, v)\} : i \in \{1, 2\}, uv \in E(G_i)\}$.

Ex: Consider a class of students. Every pair of students could work together on a homework assignment. In the first homework, pairs $a_1b_1, a_2b_2, \dots, a_kb_k$ work together (and $c_1, \dots, c_\ell$ work by themselves). A restriction on the later homeworks is that no pair can work together a second time. Thus, the graph of possible pairings is given by all possible edges, minus the pairs that have been completed. What possible pairings are possible?

Def: Let G be a graph. A *matching* is a set $M \subseteq E(G)$ such that for any two edges $e_1, e_2 \in M$ with $e_1 \neq e_2$, the edges e_1 and e_2 do not share a common endpoint.

Def: Let G be a graph with a matching M. A vertex v is *matched* if there is an edge e that is both in M and is incident to v (and v is *matched to* the other endpoint of e); otherwise v is *unmatched*. A *maximum matching* is a matching of maximum size (among all matchings in G).

2 Wednesday, April 1

Def: Let G be a bipartite graph with bipartition $V(G) = X \cup Y$. Note that a matching $M \subseteq E(G)$ consists of edges with one endpoint in X and another endpoint in Y. The matching M is called a *complete matching from X to Y* if every vertex in X is matched.

Ex: $K_{n,n}$ has a complete matching. $K_{4,5}$ has a complete matching from left to right, but not right to left.

Ex: C_{2n} has a complete matching.

(Note: A matching is NOT a subgraph, as it is only a set of edges. However, there is an associated subgraph where we include the matched vertices. Sometimes it is helpful to think about that.)

Def: Recall $N(v)$ is the neighborhood of a vertex. Thus, if $S \subseteq V(G)$, then $N(S) = \cup_{v\in S} N(v)$ (recall that we may use the notation $N_G(S) = \cup_{v\in S} N_G(v)$ when two graphs are in the discussion.)

Hall's Theorem: Let G be a bipartite graph with bipartition $V(G) = X \cup Y$. G has a complete matching from X to Y if and only if for all $S \subseteq X$ we have $|S| \leq |N(S)|$.

Proof. We shall prove both directions of the "if and only if".

We first prove that if G has a complete matching from X to Y, then $|S| \leq |N(S)|$ for all $S \subseteq X$. We prove by contradiction: Suppose there exists a complete matching M from X to Y and there also exists a set $S \subseteq X$ where $|S| > |N(S)|$. Let $S = \{s_1, \dots, s_k\}$. Since M is a complete matching, every element $s_i \in S$ is matched to an element $y_i \in Y$. However, since $M \subseteq E(G)$, $y_i \in N(s_i)$. Hence, $\{y_1, \dots, y_k\} \subseteq N(S)$. However, $|N(S)| < |S| = k$ so not all of the elements y_i are distinct and hence M is not a matching, a contradiction!

We now prove the other direction: If $|S| \leq |N(S)|$ for all $S \subseteq X$ then G has a complete matching from X to Y. We will prove this using induction on $|X| \geq 0$.

Base Case 1: If $|X| = 0$, then let $M = \varnothing$.

Base Case 2: If $|X| = 1$, then there exists a single vertex $x \in X$. Since $N(x) = N(\{x\})$, we have $|N(x)| \geq 1$ and thus there is an edge $xy \in E(G)$ and we let $M = \{xy\}$.

(Strong Induction Hypothesis) Let $n \geq 1$ and suppose that if G is a bipartite graph with bipartition $V(G) = X \cup Y$ and $|X| \leq n$, the statement holds.

Case $n + 1$: Suppose G is a bipartite graph with bipartition $V(G) = X \cup Y$ where $|X| = n + 1 \geq 2$. We will use two sub-cases.

Subcase 1: Suppose that for all sets $S \subseteq X$ with $1 \leq |S| \leq n$ we have $|N(S)| \geq |S| + 1$. In this case, let $x \in X$ and let $y \in N(x)$ (which exists, since $|N(x)| \geq |\{x\}| + 1$). Let $G' = G - x - y$ (the graph given by removing the vertices x and y from G). Then G' is bipartite with bipartition $V(G') = (X \setminus \{x\}) \cup (Y \setminus \{y\})$. Also, for all $S \subseteq X \setminus \{x\}$, $|N_{G'}(S)| \geq |N_G(S)| - 1 \geq |S|$. Thus, by the induction hypothesis, G' has a complete matching M' from $X \setminus \{x\}$ to $Y \setminus \{y\}$. Let $M = M' \cup \{xy\}$ and observe that M is a complete matching of G from X to Y.

Subcase 2: Now suppose that there exists a set $S \subseteq X$ with $1 \leq |S| \leq n$ and $|S| = |N(S)|$. Let H be the subgraph of G induced by $S \cup N(S)$. H has bipartition $V(H) = S \cup N(S)$ and H has the property that for all $T \subseteq S$ that $|N_H(T)| \geq |T|$ (since $T \subseteq S \subseteq V(G)$ and $N_H(T) = N_G(T)$). Thus, by the strong induction hypothesis, H has a complete matching M from S to $N(S)$.

Let $X' = X \setminus S$ and $Y' = Y \setminus N(S)$. Let G' be the subgraph of G induced by $X' \cup Y'$ and observe that G' has bipartition $X' \cup Y'$. We need to verify that for all $T \subseteq X'$ we have $|N_{G'}(T)| \geq |T|$. For the sake of contradiction, suppose that there exists a set $T \subseteq X'$ where $|N_{G'}(T)| < |T|$. Now consider $S \cup T \subseteq X$. We have $N_G(S \cup T) = N_G(S) \cup (N_G(T) \setminus N_G(S)) = N_G(S) \cup N_{G'}(T)$. Since $N_G(S) \cap N_{G'}(T) = \varnothing$, we have $|N_G(S \cup T)| = |N_G(S)| + |N_{G'}(T)| < |S| + |T| = |S \cup T|$, contradicting the condition on G!

Thus, no such set T exists, and by the strong induction hypothesis there exists a complete matching M' from X' to Y' in G'. Therefore, $M \cup M'$ is a complete matching of G from X to Y. $\square$

2.1 Rosen 10.3: Representing Graphs and Graph Isomorphism

Reading: Rosen 10.3, Ducks , LLM

Def: Let G be a graph with vertex set $V(G) = \{v_1, \dots, v_n\}$. The *adjacency matrix* of G is the $n \times n$ matrix $A = A(G)$ where the i,j-entry is 1 if v_iv_j is an edge of G, and 0 otherwise.

Ex: Create the adjacency matrices for $C_4, C_5, C_6, K_3, K_4, K_5, K_{2,2}, K_{2,3}, K_{4,4}, Q_3, Q_4$.

Note: $A(G)$ is *symmetric*, as v_iv_j is an edge of G if and only if v_jv_i is an edge of G.

Def: Given an $n\times n$ symmetric matrix A with entries in $\{0,1\}$ and all diagonal entries 0, the graph $G = G(A)$ is the graph with vertex set $\{1, \dots, n\}$ where the edge ij is in $E(G)$ if and only if the i,j-entry of A is 1.

Ex: Draw the graphs $G(A_i)$ associated with the adjacency matrices A_i below:

$$A_1 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad A_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Note: When encoding A as a doubly-indexed array in C, Java, or almost any programming language that is 0-indexed, we treat A as having i,j-entries with $0 \le i < n$ and $0 \le j < n$. Thus we order $V(G) = \{v_0, \dots, v_{n-1}\}$ and then we can use the following type of code:

```
int** A = (int**)malloc( n * sizeof(int*) );
for ( int i = 0; i < n; i++ )
{
    A[i] = (int*)malloc( n * sizeof(int) );
    for ( int j = 0; j < n; j++ )
    {
        A[i][j] = 0;
    }
}
// We now have an adjacency matrix for the empty graph on n vertices
```

We can add edges to the graph as follows:

```
void addEdge(int** A, int i, int j)
{
    A[i][j] = 1;
    A[j][i] = 1; // Stay symmetric!
}
```

Note: Since A is symmetric and has 0's on the diagonal, we only need to know the i,j-entries when $1 \le i < j \le n$ (i.e. the upper triangle, minus the diagonal). There are $\binom{n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$ such entries, so if you have $\frac{1}{2}n^2$ bits, you can "pack" the information of an adjacency matrix into that small of a container! This is hard to do, so you should only do it when your memory footprint needs to be REALLY small!

Def: Let G be a graph with vertex set $V(G) = \{v_0, \dots, v_{n-1}\}$. An *adjacency list* representation of G is a data structure that stores $N(v_i)$ for every vertex $v_i \in V(G)$.

Here is an example in C:

```
int** adjacency_list = (int**)malloc( n * sizeof(int*) );

for ( int i = 0; i < n; i++ )
{
    int degree = 0;
    for ( int j = 0; j < n; j++ )
    {
        degree += A[i][j];
    }

    adjacency_list[i] = (int*)malloc( (degree+1) * sizeof(int*) );

    int cur_pos = 1;
    for ( int j = 0; j < n; j++ )
    {
        if ( A[i][j] == 1 )
        {
            adjacency_list[i][cur_pos] = j;
            cur_pos++;
        }
    }
    adjacency_list[i][cur_pos] = -1; // "null" terminated.
}
```

There are many ways to store an adjacency list or an adjacency matrix. Bit-packing can decrease the size of an adjacency list to be essentially $Cn + Dm\log n$ for some constants $C, D > 1$ where n is the number of vertices and m is the number of edges. You could also use linked lists, (self-balancing) binary trees, or hash tables to store the lists of adjacent vertices. You may also want to store a few things in addition, such as number of vertices, number of edges, the degree sequence, labels on the vertices, etc.

	Pros	**Cons**
Adjacency Matrix	Random-Access Lookups Matrix algebra encodes more information	$\frac{1}{2}n^2 - \frac{1}{2}n$ bits required for n vertices.
Adjacency List	$\approx m\log n$ bits required for n vertices and m edges (good if $m \ll n^2$).	Linear-Time Lookups

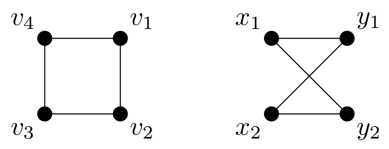
3 Friday, April 3

3.0.1 Isomorphism

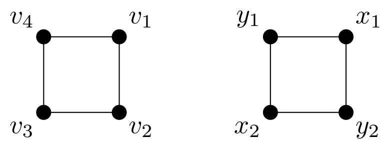
Consider the following adjacency matrices:

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

You may notice that $A_1 = A(C_4)$ and $A_2 = A(K_{2,2})$ (if you did the earlier exercises!)



However, if we "redraw" $K_{2,2}$ we will get a different picture:



By these pictures, these graphs look the same! In fact, if we erased the vertex labels, we could not tell them apart!

When I say we "redraw" the graphs, what we are really doing is pairing the vertices of C_4 with the vertices of $K_{2,2}$ with the following map:

$$\begin{array}{rcl} v_1 & \mapsto & x_1 \\ v_2 & \mapsto & y_2 \\ v_3 & \mapsto & x_2 \\ v_4 & \mapsto & y_1 \end{array} \qquad \begin{array}{lrcl} & v_1v_2 & \mapsto & x_1y_2 \\ & v_2v_3 & \mapsto & y_2x_2 \\ & v_3v_4 & \mapsto & x_2y_1 \\ \text{edges} & v_4v_1 & \mapsto & y_1x_1 \\ \hline \text{nonedges} & v_1v_3 & \mapsto & x_1x_2 \\ & v_2v_4 & \mapsto & y_2y_1 \end{array}$$

In the above two lists, we define a bijection from $V(C_4)$ to $V(K_{2,2})$ on the left. On the right, we see how that bijection sends pairs of vertices from $V(C_4)$ to pairs of vertices in $V(K_{2,2})$. Observe that the edges of C_4 are sent to edges of $K_{2,2}$ and the nonedges of C_4 are sent to nonedges of $K_{2,2}$.

There is something *fundamental* happening here, given by the following definition.

Def: Let G and H be graphs. An *isomorphism from G to H* is a function $f : V(G) \to V(H)$ such that (a) f is a bijection and (b) for every pair $u, v \in V(G)$, the edge uv is in $E(G)$ if and only if $f(u)f(v)$ is in $E(H)$. (That is, the function f sends edges in G to edges in H *and* nonedges in G to nonedges in H; informally, the bijection $f : V(G) \to V(H)$ induces a bijection from $E(G)$ to $E(H)$.) If there exists an isomorphism from G to H, then G and H are *isomorphic*, denoted $G \cong H$.

Note: The bijection $f : V(G) \to V(H)$ can be though as a way to *relabel* the vertices of G to match the vertices of H and end up with the same edge set as H!

To demonstrate $G \cong H$: Define a bijection $\pi : V(G) \to V(H)$ then demonstrate why $uv \in E(G)$ if and only if $\pi(u)\pi(v) \in E(H)$. For small graphs (such as those below), we only need to check each pair on a case-by-case basis.

Ex: $C_4 \cong K_{2,2}$.

Ex: $W_3 \cong K_4$.

Ex:

Thus, when I draw a graph on the board or on a piece of paper, I'm not just talking about one graph, but talking about all of the graphs isomorphic to that graph! This is why we do not need to list the labels of our vertices (unless it is helpful).

Def: Let G be a graph. The *class of graphs isomorphic to* G is the set of all graphs H such that $H \cong G$ (this is infinite, as we could select $V(H)$ to be ANY set of n elements, but typically we think of $V(H) = V(G) = \{1, \dots, n\}$.) In general, an *isomorphism class* is a family of graphs that are isomorphic to each other.

The most wonderful thing about isomorphism is that most graph properties that we care about are *invariant under isomorphism*. That is, if $G \cong H$, then G has a property P if and only if H has a property P. Here are some examples (proofs are omitted so you can try to prove them!):

Prop: Let G and H be graphs. If $G \cong H$, then $|V(G)| = |V(H)|$ and $|E(G)| = |E(H)|$.

Thm: Let G and H be graphs. If $G \cong H$, then the degree sequence of G equals the degree sequence of H.

Thm: Let G and H be graphs. If $G \cong H$, then G is bipartite if and only if H is bipartite.

Thm: Let G and H be graphs with $|V(G)| = |V(H)| = n$. Let m be an integer with $1 \le m \le \lfloor \frac{n}{2} \rfloor$. If $G \cong H$, then G contains a matching of size m if and only if H contains a matching of size m.

Thm: Let G, H, and C be graphs. If $G \cong H$ and $C \subseteq G$, then there is a subgraph $C' \subseteq H$ such that $C' \cong C$.

With the above theorem, we want to relax our definition of subgraph a little bit.

Def: Let G and H be graphs. We say G *contains a copy of* H if there is a subgraph $H' \subseteq G$ such that $H' \cong H$. Frequently, we will simply say that H *is a subgraph* of G, even if the set-inclusion is not true, but we mean that G contains a copy of H.

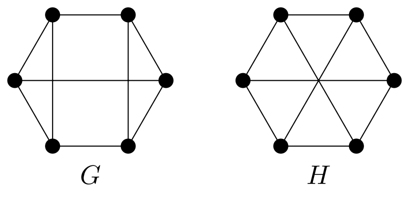
Thm: Let G, H, and C be graphs. If $G \cong H$, then G contains a copy of C if and only if H contains a copy of C.

To demonstrate $G \ncong H$: To show *nonisomorphism*, we are saying that no isomorphism exists! Non-existence is always a harder statement (in general) but can frequently be made easier by using results such as the above! If P is a property that is invariant under isomorphism and $P(G) \neq P(H)$, then $G \ncong H$!

Ex: $C_4 \ncong C_5$ since $|V(C_4)| = 4 \neq 5 = |V(C_5)|$.

Ex: $W_4 \ncong K_5$ since $|E(W_4)| = 8 \neq 10 = |E(K_5)|$.

Ex: Prove that the two graphs below are not isomorphic:



Before we prove $G \ncong H$, let's first note some things: $|V(G)| = 6 = |V(H)|$, $|E(G)| = 9 = |E(H)|$, and every vertex has degree 3 in each graph!

Proof. Note that G contains K_3 as a subgraph. Thus, G is not bipartite (as K_3 is not bipartite).

However, H is bipartite as we can take every other vertex around the outer cycle to form a bipartition.

Since G is not bipartite and H is bipartite, $G \ncong H$. □

There is no tried and true method to easily determine if two graphs are isomorphic or not. This is mostly due to the fact that there is no "easy proof" that two graphs are not isomorphic. (Contrast this with the fact that we have a short proof of a bipartite graph not having a complete matching, by Hall's Thm.) This is part of why the "GraphIsomorphism" problem (take two graphs, decide if they are isomorphic) is thought to be a so-called *intermediate problem*: No known polynomial-time algorithm exists, but also no proof of being NP-complete exists! GraphIsomorphism is one of the few intermediate problems that occur "naturally" (that is, it was not first defined for the sake of being intermediate).

HOWEVER: a good strategy for showing $G \ncong H$ is to go through your list of graph properties: do they have the same number of vertices? do they have the same number of edges? Do they have the same degree sequence? Are they both bipartite or both not bipartite? Does one contain a subgraph the other does not?

As we discover more graph properties, you will have more questions to add to your arsenal!