# GRAPH THEORY LECTURE NOTES

ARUN MAITI

Contents

## 1. Basics of Graphs

The graph are mathematical structures used to effectively model pairwise relations between objects. We will use the following standard notations throughout the text.

1.1. **Definition:** Formally, we define a *graph* to be an ordered pair $G = (V, E)$ where $E \subseteq [V]^2 := \{Y \subseteq V : |Y| = 2\}$ or
A *graph*, $G$ is an ordered pair $(V, E)$, where $V$ is a finite set and $E \subseteq \binom{V}{2}$ is a set of pairs of elements in $V$.

- $G = (V, E)$ is an arbitrary (undirected, simple) graph
- $|V|$ is its number of vertices
- $|E|$ is its number of edges

**Example 1.1.** *In this example* $V = \{1, 2, 3, 4, 5, 6, 7\}$*, with edge set* $E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 4\}, \{5, 7\}\}$.

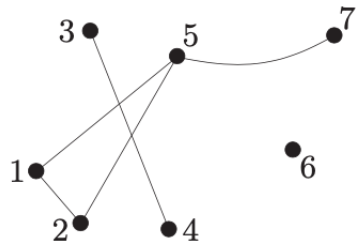
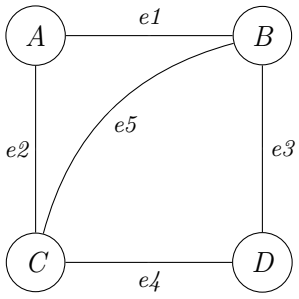



*Fig. 1.1.1.* The graph on $V = \{1, \ldots, 7\}$ with edge set $E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 4\}, \{5, 7\}\}$

**Example 1.2.** *Another example, where vertices are shown as* $A, B, C$ *and* $D$ *and edges labelled as* $e_1 := (A, B), e_2 := (A, C) e_3 := (B, D), e_4 := (C, D), e_5 := (C, D)$.




1.2. **Notations and Terminologies:** For a graph $G$, the set $V(G)$ is the *vertex set* of $G$ and $E(G)$ is the *edge set*.

- The *order* of $G$ is $|V(G)|$, often written $|G|$.
- The *size* of G is $|E(G)|$, often written $||G||$.
- *Self loop* An edge that is associated with same vertex as both its end vertices is called self loop. or, having an edge with both ends at the same vertex. Such edges are called *loop or self loop.*

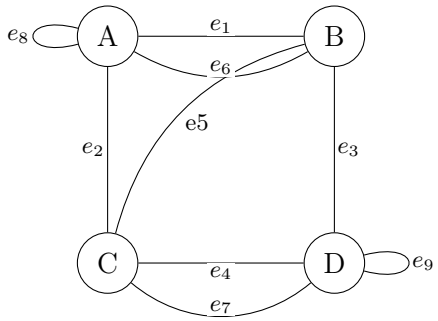
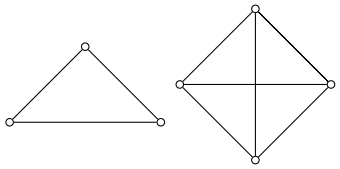



**Fig. 1.1.** graph with parallel edges and loops

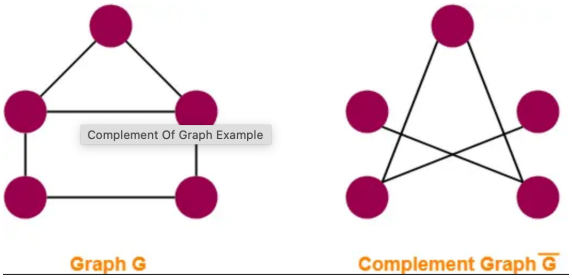
- *Parallel edges* The edges associated with same pair of start & end vertices are called parallel edges, or having more than one edge between the same two vertices. Such edges are called *parallel edges*.
- *Simple graph* A graph that has neither self loop nor parallel edges is called a simple graph. The examples of graph above in Fig 1.1 and 1.2 are clearly simple graphs.
- We shall not always distinguish strictly between a graph and its vertex or edge set. For example, we may speak of a vertex $v \in G$ (rather than $v \in V(G)$), an edge $e \in G$, and so on.
- A vertex $v$ is *incident with an edge* $e$ if $v \in e$; then e is an edge at v. The set of all incidents edges to a vertex $v$ is denoted by $|E(v)|$. The *degree of a vertex* $v$ is defined to be $|E(v)|$.
- Two vertices $x, y$ of $G$ are *adjacent*, or neighbours, if $x, y$ is an edge of G.
- A *pendant vertex* is a vertex whose degree is 1.
- An edge that has a pendant vertex as an end vertex is a pendant edge.
- An *isolated vertex* is a vertex whose degree is 0.
- A graph with no edges is called *Null Graph*.
- The degree of a vertex is sometimes also known as valency.
- Two graphs $G, H$ are said to be *isomorphic* if there exists a bijection $f : V(G) \to V(H)$ such that $(u, v) \in E(G) \iff (f(u), f(v)) \in E(H)$.
- A *subgraph* of $G$ is another graph $H$ with $V(H) \subseteq V(G), E(H) \subseteq E(G)$.
- A simple graph that contains every possible edge between all the vertices is called a *complete graph*. The complete graph with $n$ vertices is denoted by $K_n$.
- A complete graph with $n$ vertices is $([n], \binom{n}{2})$. Every graph of order $\leq n$ is a subgraph of $K_n$. The following graphs are complete graphs:



$K_3$ $K_4$

- The *minimum degree* of the vertices in a graph $G$ is denoted $\delta(G)$ (which equals 0 if there is an isolated vertex in $G$). Similarly, we write $\Delta(G)$ as the maximum degree of vertices in $G$.

- The *complement or inverse* of a graph $G$ is a graph H on the same vertices such that two distinct vertices of $H$ are adjacent if and only if they are not adjacent in $G$. That is, to generate the complement of a graph, one fills in all the missing edges required to form a complete graph, and removes all the edges that were previously there.




- A graph with infinite number of vertices is an example of *Infinite graph*. For example:

$$v_1 \to v_2 \to v_3 \to v_4 \to \cdots$$

**Remark 1.3.** *In this course, we only consider finite graphs, i.e. $V$ and $E$ are finite sets. We shall not always distinguish strictly between a graph and its vertex or edge set. For example, we may speak of a vertex $v \in G$ (rather than $v \in V(G)$), an edge $e \in G$, and so on.*

1.3. **Subgraphs.** A *subgraph* $G'$ of a graph G is a graph $G'$ whose vertex set and edge set are subsets of those of G. If $G'$ is a subgraph of G, then G is said to be a *supergraph* of $G'$. An *induced subgraph* of a graph is another graph, formed from a subset of the vertices of the graph and all of the edges, from the original graph, connecting pairs of vertices in that subset.

1.4. **Graph operations.** Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be the following graphs. Then

- $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$
- $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$
- $G_1 - G_2 = (V_1 \setminus V_2, E_1 \setminus E_2)$

Given two graph $G$ and $G'$ shown below, the union, difference and intersection $G \cup G'$, $G \cap G'$, respectively of these graphs as shown below:

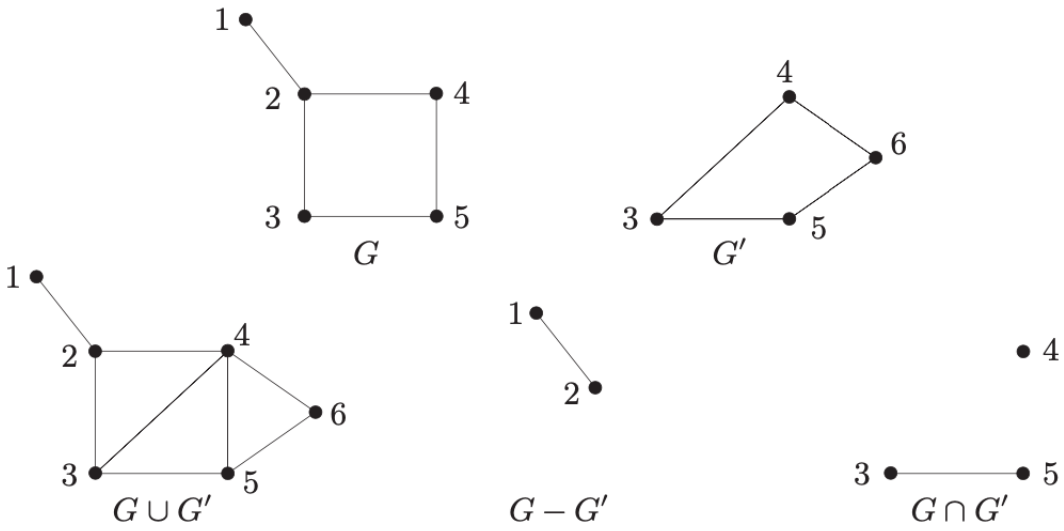



*Fig. 1.1.2.* Union, difference and intersection; the vertices 2,3,4 induce (or span) a triangle in $G \cup G'$ but not in $G$

**Definition 1.4.** *A graph $G$ is said to have been decomposed into two subgraphs $H_1$ and $H_2$ if*

$$H_1 \cup H_2 = G \quad \text{and} \quad H_1 \cap H_2 = \text{a null graph.}$$

*i.e.,*

- *Every edge of $G$ occurs in either $H_1$ or $H_2$ but not in both.*
- *Some vertices, however, may occur in both $H_1$ and $H_2$.*

**Definition 1.5. Deletion of Vertex and Edge, and Fusion in Graph**
**Vertex Deletion**

*Let $G = (V, E)$ be a graph with vertex set $V$ and edge set $E$. For a vertex $v \in V$, the deletion of $v$ from $G$ results in a subgraph $G'$ defined as follows:*

$$G' = G - v = (V', E')$$

*where $V' = V \setminus \{v\}$ and $E' = \{e \in E \mid v \notin e\}$. This means that $G'$ is obtained by removing $v$ and all edges incident to $v$.*

**Edge Deletion**
*Let $G = (V, E)$ be a graph with vertex set $V$ and edge set $E$. For an edge $e = \{u, v\} \in E$, the deletion of $e$ from $G$ results in a subgraph $G'$ defined as follows:*

$$G' = G - e = (V, E')$$

*where $E' = E \setminus \{e\}$. This means that $G'$ is obtained by removing $e$ but keeping all vertices of $G$.*

**Example 1.6.** *Illustration of the above process:*

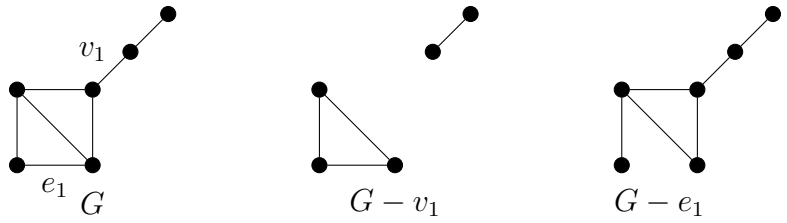



**Fig. 1.2.** Examples of vertex and edge deletion in a graph.

***Fusion in Graph***

*Let $G = (V, E)$ be a graph with vertex set $V$ and edge set $E$. Fusion of two vertices $u, v \in V$ (also known as vertex contraction) results in a graph $G'$ defined as follows:*

- *Replace vertices $u$ and $v$ with a single new vertex $w$.*
- *Replace all edges $\{u, x\}$ and $\{v, x\}$ with edges $\{w, x\}$, removing any duplicate edges (if $G$ is a simple graph).*
- *Remove any self-loops formed during this process.*
- *The number of vertices will always be reduced by one in fusion operation, one cannot say the same about edges.*

*Formally, the new graph $G' = (V', E')$ is defined as:*

$$V' = (V \setminus \{u, v\}) \cup \{w\}$$

$$E' = \{\{w, x\} \mid \{u, x\} \in E \text{ or } \{v, x\} \in E, x \neq u, x \neq v\}$$

**Example 1.7.** *Example of Fusion:*

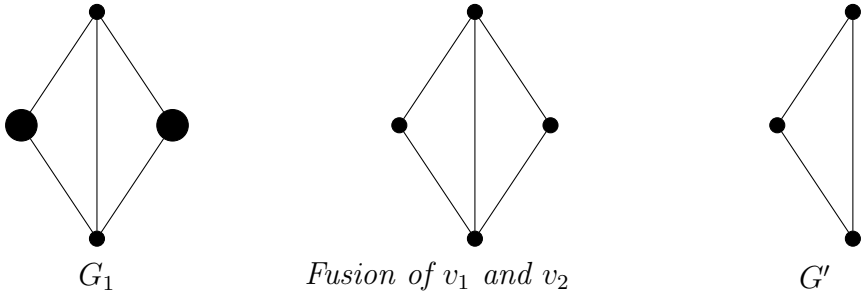



**Fig. 1.3.** Fusion of vertices $v_1$ and $v_2$ in a graph.

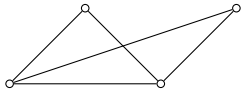
1.5. **Walks, Paths, and Cycles.** A *walk* in a graph $G = (V, E)$ is a finite sequence of vertices and edges, $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$, such that $e_i = (v_{i-1}, v_i)$ for $i = 1, 2, \ldots, k$. The number $k$ is the *length* of the walk. A walk is *closed* if $v_0 = v_k$ and *open* otherwise.

A *trail* is a walk in which all edges are distinct.

A *path* is a walk in which all vertices (and thus all edges) are distinct.

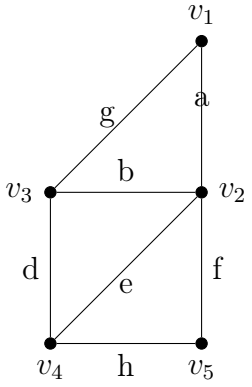
A *cycle* is a trail in which only the first and last vertices are equal.

Example. Consider the following graph $G = (V, E)$:



In this graph:

- $v_1, e_1, v_2, e_2, v_3, e_3, v_4$ is a walk of length 3.
- $v_1, e_1, v_2, e_2, v_3, e_5, v_1$ is a closed walk of length 3.
- $v_1, e_1, v_2, e_2, v_3, e_3, v_4$ is a path of length 3.
- $v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, v_1$ is a cycle of length 4.



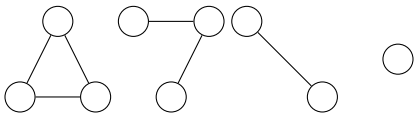

In above graph,

- Example of an open walk: $v_1 \to v_2 \to v_4 \to v_3 \to v_2$
- Example of a closed walk: $v_1 \to v_2 \to v_4 \to v_5 \to v_2 \to v_1$
- Example of a trail: $v_1 \to v_2 \to v_4 \to v_5 \to v_2$
- Example of a path: $v_1 \to v_2 \to v_4 \to v_5$
- Example of a cycle: $v_2 \to v_4 \to v_5 \to v_2$

1.6. **Connected Graphs.** A graph is *connected* if there is a $u - v$ path (a path from $u$ to $v$ in $G$ for every pair $u, v \in V(G)$.

- A null graph of more than 1-vertex is a disconnected graph.
- The *components* of $G$ are the maximal connected subgraphs of $G$ (induced by the equivalence classes of the relation $u \leftrightarrow v \iff u = v$ or there is a $u - v$ path).

**Example 1.8.** *The following graph has 4 components.*



**Theorem 1.9.** *A graph $G$ is disconnected if and only if its vertex set $V$ can be partitioned into two nonempty disjoint subsets $V_1$ and $V_2$ such that there exists no edge in $G$ whose one end vertex is in $V_1$ and the other in $V_2$.*

**Proof:**

($\Rightarrow$) Suppose $G$ is disconnected. Then there exist two nonempty disjoint subsets $V_1$ and $V_2$ of $V$ such that there is no edge between any vertex in $V_1$ and any vertex in $V_2$.

This follows directly from the definition of a disconnected graph, where there exist at least two components with no edges connecting them.

($\Leftarrow$) Conversely, suppose $V$ can be partitioned into two nonempty disjoint subsets $V_1$ and $V_2$ such that there is no edge between $V_1$ and $V_2$. Then there are no paths connecting vertices in $V_1$ to vertices in $V_2$. Therefore, $G$ is disconnected.

## 1.7. **The degree of a vertex.**

**Lemma 1.10.** *(Euler's handshaking lemma). The sum of the degrees of the vertices of a graph is equal to twice the number of edges. Or we say the graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$, satisfies*

$$\sum_{i=1}^{n} d(v_i) = 2m.$$

*Proof.* Each edge of $G$ contributes 2 to the total degree count $\sum_{i=1}^{n} d(v_i)$, one for each incident vertices. Therefore, $m$ edges of $G$ contributes $2m$ to the sum $\sum_{i=1}^{n} d(v_i)$. Since each count of $\sum_{i=1}^{n} d(v_i)$ must be coming from one of the edges of $G$, hence $\sum_{i=1}^{n} d(v_i) = 2m$. □

**Theorem 1.11.** *The number of vertices of odd degree in a graph is always even.*

Proof. Let $G$ be a graph with vertices $v_1, v_2, \cdots, v_n$ and $m$ edges. Then by the Handshake lemma,

$$\begin{aligned} &\sum_{i=1}^{n} d(v_i) = 2m \\ \implies &\sum_{d(v_i) \text{ is odd}} d(v_i) + \sum_{d(v_i) \text{ is even}} d(v_i) = 2m \\ \implies &\sum_{d(v_i) \text{ is odd}} d(v_i) = 2m - \sum_{d(v_i) \text{ is even}} d(v_i) \end{aligned}$$

RHS is an even number because it is the sum of even numbers. Therefore, the LHS is an even number. Hence proved.

**Theorem 1.12.** *If a graph has exactly two vertices of odd degree, there must be a path joining these two vertices.*

**Proof:** Consider a graph $G$ with exactly two vertices $u$ and $v$ of odd degree. If there is no path between $u$ and $v$, then $u$ and $v$ lie in two different connected components of $G$.

In any connected component, the sum of the degrees of all vertices is even, since it is twice the number of edges (each edge contributes 2 to the sum of degrees).

However, if $u$ and $v$ are in different components, then each component has an odd sum of degrees (since each has one vertex of odd degree). This contradicts the fact that the sum of the degrees of vertices in any graph is always even. Thus, there must be a path joining $u$ and $v$.

**Theorem 1.13.** *A simple graph with $n$ vertices and $k$ components can have at most $\frac{(n-k)(n-k+1)}{2}$ edges.*

*Proof.* Let $G$ be a simple graph with $n$ vertices and $k$ connected components. Let the components be $C_1, C_2, \ldots, C_k$, and let the number of vertices in component $C_i$ be $n_i$.

The total number of vertices is the sum of the vertices in each component

$$\sum_{i=1}^{k} n_i = n$$

Since each component must have at least one vertex, we have $n_i \geq 1$ for all $i$.

The total number of edges in the graph, $|E|$, is the sum of the edges in each component. To maximize $|E|$, we must maximize the number of edges within each component for a given set of $n_i$'s. The maximum number of edges in a simple graph with $n_i$ vertices is achieved when it's a complete graph, which has $\binom{n_i}{2}$ edges.

So, we want to maximize the function

$$|E(n_1, n_2, \ldots, n_k)| = \sum_{i=1}^{k} \binom{n_i}{2}$$

subject to the constraints $\sum_{i=1}^{k} n_i = n$ and $n_i \geq 1$.

Let's consider two components, $C_i$ and $C_j$, with $n_i$ and $n_j$ vertices respectively, where we assume $n_i \geq n_j > 1$. Let's see what happens if we move a vertex from the smaller component $C_j$ to the larger one $C_i$. The new vertex counts become $n_i' = n_i + 1$ and $n_j' = n_j - 1$.

The change in the total number of edges is

$$\Delta|E| = \left[\binom{n_i+1}{2} + \binom{n_j-1}{2}\right] - \left[\binom{n_i}{2} + \binom{n_j}{2}\right]$$

We know that $\binom{a+1}{2} - \binom{a}{2} = a$. So, the change is

$$\Delta|E| = \left[\binom{n_i+1}{2} - \binom{n_i}{2}\right] - \left[\binom{n_j}{2} - \binom{n_j-1}{2}\right] = n_i - (n_j - 1) = n_i - n_j + 1$$

Since we assumed $n_i \geq n_j$, we have $n_i - n_j \geq 0$, which means $\Delta E = n_i - n_j + 1 > 0$.

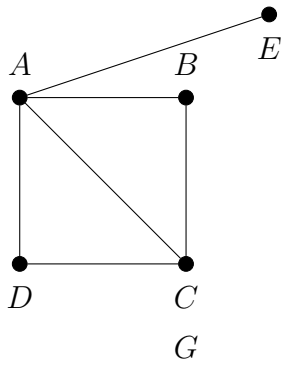
This shows that we can increase the total number of edges by moving a vertex from a smaller component to a larger one. This process of consolidating vertices into a single component maximizes the edge count.

The maximum is therefore achieved when we make one component as large as possible and the other $k-1$ components as small as possible. The minimum size for any component is 1 vertex. So, we set $n_1 = n_2 = \cdots = n_{k-1} = 1$, and $n_k = n - (k-1) = n - k + 1$ (all the remaning vertices).

A component with 1 vertex has 0 edges. The large component has $\binom{n-k+1}{2} = \frac{(n-k+1)(n-k)}{2}$ edges. Thus, a simple graph with $n$ vertices and $k$ components can have at most $\frac{(n-k)(n-k+1)}{2}$ edges.

□

1.8. **Adjacency and incident matrix.** For a simple graph with vertex set $U = \{u_1, \cdots, u_n\}$, the adjacency matrix is a square $n \times n$ matrix A such that its element $A_{ij}$ is one when there is an edge from vertex $u_i$ to vertex $u_j$, and zero when there is no edge.




$$A_G = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

In general, for a graph (not necessarily simple) with vertex set $U = \{u_1, \cdots, u_n\}$, the adjacency matrix is a square $n \times n$ matrix A such that its element $A_{ij}$ for $i \neq j$ is the number of edges from vertex $u_i$ to vertex $u_j$, and zero when there is no edge. $A_{ii}$ is two times the number of loops based at $u_i$. Clearly, adjacency matrix is symmetric.

The **incidence matrix** of a simple graph $G$ with $n$ vertices and $m$ edges is a $n \times m$ matrix $B_G$, where n and m are the numbers of vertices and edges respectively, such that

$$(B_G)_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is incident with edge } e_j, \\ 0 & \text{otherwise.} \end{cases}$$

For example, the incidence matrix of the graph shown on the right is a matrix consisting of 4 rows (corresponding to the four vertices, 1–4) and 4 columns (corresponding to the four edges, $e_1, e_2, e_3, e_4$):

The *degree matrix* $D_G$ os $n \times n$ diagonal matrix with $(i, i)$ entry given by degree of $i$-th vertex $v_i$ .
Exercise: $B_G B_G^T = A_G + D_G$.

**Definition 1.14.** *For a simple graph $G$, the Laplacian matrix of $G$ is defined to be $L_G = D_G - A_G$.*

- The sum the rows of the Laplacian matrix (or the columns, since it's symmetric), is always zero.
- Laplacian is a singular matrix
- Null space $L_G$ is one dimensional

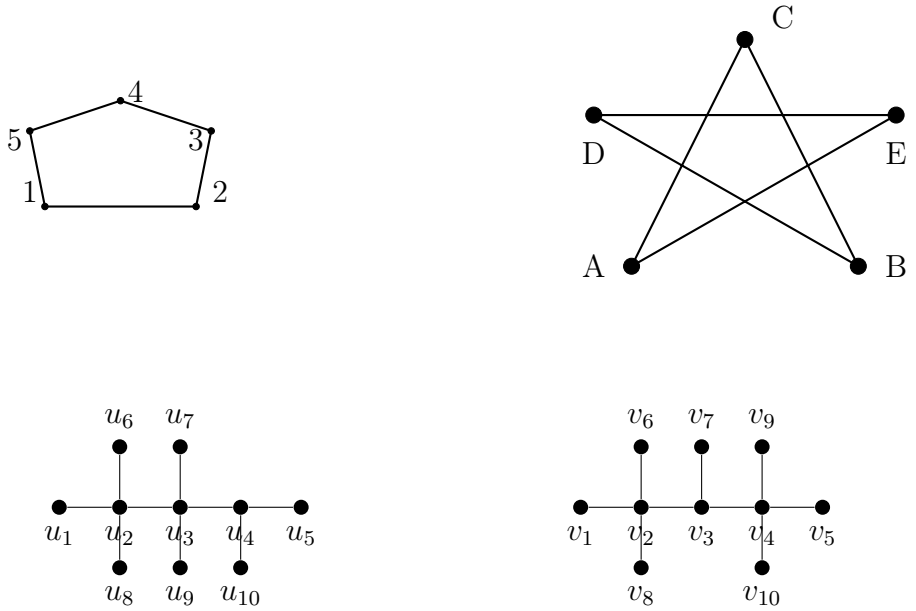
- $L_G$ is positive semi-definite matrix, consequently, eigenvalues of $L_G$ are non-negative.

**Theorem 1.15** (Matrix tree theorem/Kirchoff Theorem)**.** *For a given connected graph $G$ with $n$ labeled vertices, let $\lambda_1, \lambda_2, \cdots, \lambda_n$ be the non-zero eigenvalues of its Laplacian matrix. Then the number of spanning trees of $G$ is*

$$t(G) = \frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}\,.$$

*Proof.* Beyond the scope of this course. □

1.9. **Isomorphism of graphs.** Two graphs $G = (V, E)$ and $G' = (V', E')$ are said to be isomoprhic if there exist a bijective function (called isomorphism) $f : V \to V'$ such that $f(e) \in E'$ for every edge $e \in E$.
Example: The following two graphs are isomorphic. It is easy to verify that following bijection $f : V \to V$ given by $1 \to A, 2 \to C, 3 \to B, 4 \to D, 5 \to E$ defines an isomorphism.

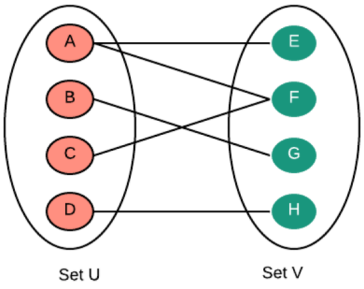



Let us consider the above two graphs. An isomorphism between them must map a vertex of degree $d$ of the first graph to a vertex of degree $d$ in the second graph. Therefore, the only vertex with degree 3, $u_4$ must be mapped to the only vertex of degree 3, $v_3$ . An isomorphism must also map adjacent vertices to adjacent vertices. Subsequently, $u_5$ (adjacent to $u_4$) must map to a degree 2 vertex adjacent to $v_3$. But both the vertices adjacent to $v_3$ ($v_2$ and $v_4$) have degree 4. Hence, an isomorphism is impossible. Hence, there doesn't exist any isomorphism between the two graph.

**Theorem 1.16.** *Two graphs $G$ and $H$ are isomorphic if and only if there is a permutation matrix $P$ such that $A_G = PA_HP^{-1}$*

*Proof.* Exercise. □

1.10. **Bipartite Graph.** A bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint and independent sets $U$ and $V$, that is, every edge connects a vertex in $U$ to one in $V$. Vertex sets $U$ and $V$ are usually called the parts of the graph.




**Bipartite Graph OR Bi-Graph**

A complete bipartite graph is a bipartite graph where every vertex of the first set is connected to every vertex of the second set. It is denoted by $K_{m,n}$ where $m$ and $n$ denote the number of vertices in the first and second set respectively.

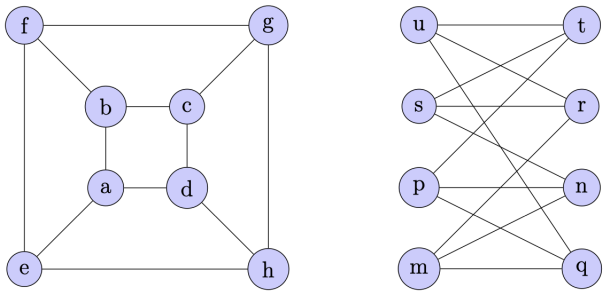
1.11. **Practice problems set 1.** 1. Can there exist a graph on 13 vertices and 31 edges, with three vertices of degree 1, and seven vertices of degree 4? Explain.

2. Given the following degree sequences either construct a graph with such a degree sequence, or explain why this would be impossible.
(a) 1,1,1,1,1,1
(b) 5,4,3,2,1
(c) 6,6,4,2,2,2,2,1

3. What is the maximum number of vertices on a graph that has 35 edges and every vertex has degree $\leq 3$?

4. Suppose all vertices in a graph, G, have odd degree, k. Prove that k divides $|E(G)|$.

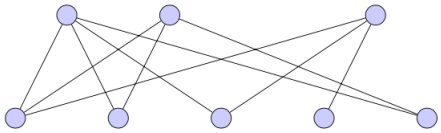
5. Given the following two graphs, write an explicit isomorphism between them.



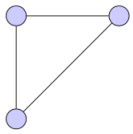

6. Draw all non-isomorphic graphs with n vertices for $n = 3$ and 4.

7. Can a graph have $K_3$ subgraph and be bipartite? Explain.

8. Let G be the following graph:



a) Is the following a subgraph of G?



(b) Draw an induced subgraph of G with exactly 3 edges.

## 2. Eulerian and Hamitonian Graphs

Graph theory is typically dated as beginning with Leonhard Euler's solution to the problem of the Seven Bridges of Königsberg. Click below link for the description of the problem.
Königsberg

2.1. **Eulerian circuit.** A closed walk in a graph an *Euler circuit* if it traverses every edge of the graph exactly once. A graph is *Eulerian* if it admits an Euler circuit.
*Concatenation* of two walks

$$v_1 \to v_2 \cdots \to v_i \to \cdots \to v_{k-1} \to v_k, \quad \text{and} \quad v_i \to v_l \to \cdots \to v_i$$

is defined to be the walk

$$v_1 \to v_2 \cdots \to v_i \to v_l \to \cdots \to v_i \to \cdots \to v_{k-1} \to v_k$$

**Theorem 2.1.** *A connected graph is Eulerian if and only if every vertex has even degree.*

*Proof.* The degree condition is clearly necessary: a vertex $v$ appearing $k$ times in an Euler circuit of the form

$$v_1 \to v_2 \cdots v_i \to v \to v_{i+1} \cdots v_k, v_{k+1} \cdots \to v \to \cdots v_1$$

must have degree $2k$ (not counting starting and ending vertex twice if v is as such).

Conversely, we show by induction on $||G||$ that every connected graph $G$ with all degrees even has an Euler circuit. The induction starts trivially with $||G|| = 0$. Now let $||G|| \geq 1$. We can form a closed walk in $G$ starting from an arbitrary vertex. This is possible because the degree of each vertex is even; for every edge that we traverse to reach/enter a vertex there must be an edge to exit. Let $W$ be such a walk of maximal length. If the set of edges traversed in $W$ is the same as $E(G)$, then W is an Euler circuit.

Suppose, therefore, that $G' := G - W$ (removing edges of W from G) has an edge. For every vertex $v \in G$, there are even number of edges incident to $v$ in $W$, so the degree of each vertex of $G'$ are again all even. Since G is connected, $G'$ has an edge $e$ incident with a vertex on W. By the induction hypothesis, the component C of $G'$ containing e has an Euler tour. Concatenating this with $W$ (suitably re-indexed), we obtain a closed walk in G that contradicts the maximal length of W. □

An *Eulerian trail* (or Eulerian path) is a walk that visits every edge exactly once (allowing for revisiting vertices).

2.2. **Hamiltonian circuit.** A *Hamiltonian cycle* in a graph is a closed cycle that visits each vertex of the graph exactly once. Hamiltonian cycles are named after Irish mathematician Sir William Rowan Hamilton, who studied such properties of graphs in the 19th century. William Hamilton, described a game on the graph of the dodecahedron in which one player specifies a 5-vertex path and the other must extend it to a spanning cycle. The game was marketed as the "Traveller's Dodecahedron", a wooden version in which the vertices were named for 20 important cities.

Until 1970s, interest in Hamiltonian cycles centered on their relationship to the four color problem, later study was stimulated by practical applications and by the issue of complexity.

- **Hamiltonian Path:** A path in a graph that visits every vertex exactly once.
- **Hamiltonian Cycle:** A cycle in a graph that visits every vertex exactly once and returns to the starting vertex.
- **Hamiltonian Graph:** A graph that contains a Hamiltonian cycle.

2.3. **Theorems and Results.** Not all graphs are Hamiltonian, and determining whether a given graph is Hamiltonian is a NP-complete problem. There are, however, a number of important results providing sufficient conditions for the existence of Hamiltonian cycle in a graph.

**Theorem 2.2. Dirac's Theorem** *Let $G$ be a simple graph with $n \geq 3$ vertices. If every vertex of $G$ has degree at least $\frac{n}{2}$, then $G$ is Hamiltonian.*

*Proof.* Let $G$ be a simple graph with $n$ vertices where each vertex has degree at least $\frac{n}{2}$. Assume for contradiction that $G$ is not Hamiltonian. Then $G$ does not contain a Hamiltonian cycle.

Consider the longest cycle $C$ in $G$. Since $G$ is not Hamiltonian, there exists a vertex $v$ in $G$ not in $C$. Since $G$ is connected and every vertex has degree at least $\frac{n}{2}$, $v$ is adjacent to at least $\frac{n}{2}$ vertices of $C$.

Consider two consecutive vertices $x$ and $y$ on $C$. If $v$ is adjacent to both $x$ and $y$, then the cycle $C' = (v, x, C_{x\to y}, y, v)$ is longer than $C$, contradicting the assumption that $C$ is the longest cycle. Thus, $v$ cannot be adjacent to two consecutive vertices on $C$, and therefore, $v$ must be adjacent to vertices separated by at least one vertex on $C$. But this implies degree of $v$ can at most be $\frac{n-1}{2}$, contradicting the hypothesis of the theorem. Thus, the the largest cycle $C$ cannot leave any vertex of $G$. Hence $G$ must contain a Hamiltonian cycle. □

**Remark 2.3.** *A pentagon does not satisfy the hypothesis of Dirac's theorem stated above, but it is clearly a Hamilton graph. Thus the hypothesis is not necessary for the existence of Hamiltonian cycle in a graph.*

**Theorem 2.4. Ore's Theorem** *Let $G$ be a simple graph with $n \geq 3$ vertices. If for every pair of non-adjacent vertices $u$ and $v$, we have* $\deg(u) + \deg(v) \geq n$, *then $G$ is Hamiltonian.*

The closure $[G]$ (also denoted by cl(G)) of a graph G with n vertices, obtained by repeatedly adding a new edge $(u, v)$ connecting a nonadjacent pair of vertices $u$ and $v$ with $\deg(v) + \deg(u) \geq n$ until no more pairs with this property can be found.

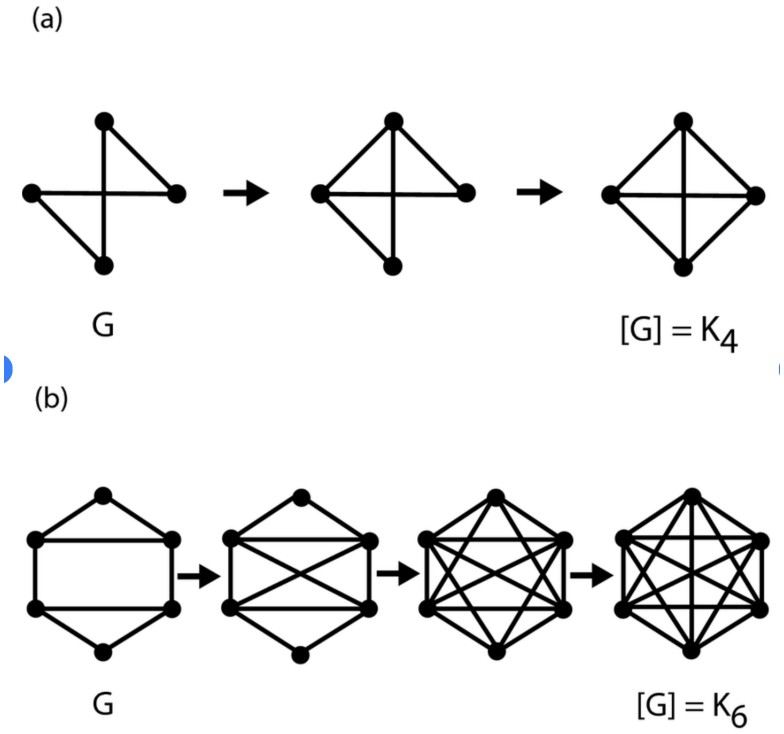



**Fig. 2.1.** Steps to form closure of a graph

**Theorem 2.5. Bondy-Chvátal's Theorem** *A graph is Hamiltonian if and only if its closure is Hamiltonian.*

*Proof.* TBD □

**Corollary 2.6.** *The Dirac theorem and Ore's theorem can be seen as straightforward consequences of the Bondy-Chvátal's Theorem.*

**Definition 2.7.** *Graph Connectivity: The connectivity of a graph, denoted $\kappa(G)$, is the minimum number of vertices whose removal disconnects the graph or reduces it to a single vertex.*

Higher Connectivity Increases Likelihood of a Hamiltonian Path: Higher connectivity (e.g., 2-connected or 3-connected) guarantees multiple vertex-disjoint paths between vertices, providing flexibility to construct a Hamiltonian cycle by choosing alternative routes to avoid revisiting vertices. For example, $C_5$ is 2-connected, and it admits a Hamiltonian cycle. In contrast, a graph with a cut vertex (1-connected), like two triangles joined at a single vertex, does not admit a Hamiltonian cycle.

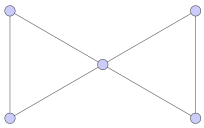
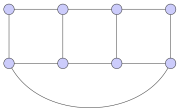
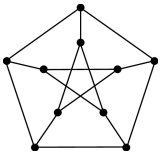
**Application** Hamiltonian graphs are a significant area of study in graph theory, with applications in various fields such as computer science (computational complexity theory), logistics (TSP), and biology. The results discussed above, though provides only sufficient conditions, but offers insight into the structure of such graphs.

2.4. **Practice Problems Set 2.** 9. For which integers $m$ and $n$ is $K_{m,n}$ Eulerian or Hamiltonian graph?

10. Is a Eulerian circuit necessarily a cycle? Prove or find a counterexample.

11. For which positive integers $n$ does $K_n$ have an Eulerian circuit?

12. Prove or disprove:
(i) Every Eulerian bipartite graph has an even number of edges.
(ii) Every Eulerian simple graph with an even number of vertices has an even number of edges.
13. Let $v$ be a cut vertex of a simple graph $G$. prove that $\bar{G} - v$ is connected.
14. Which of the following graphs are Hamiltonian? If they are Hamiltonian identify a Hamiltonian cycle. If they are not, explain briefly why.

15. A group of n people are going out to dinner,where $n \equiv 0(mod\ 2)$ and $n \geq 3$. If every person going to dinner is friends with at least half the group, prove it is possible to seat the friends around a circular table so each person is seated next to two friends.
7. Determine if each statement is true or false. If true, provide a brief proof. If false, find an explicit counterexample.
(a) A graph of order $n \geq 4$ that contains a triangle cannot be Hamiltonian.
(c) If there exists a Hamiltonian path between any two vertices in a graph, then the graph is Hamiltonian.

## 3. **Tree**

**Definition 3.1.** *A tree is a connected graph that has no circuit. A forest is a graph that has no circuit, in other words, it is a graph whose each component is a tree.*

**Example 3.2.** *As the naming suggest, any real tree*

**Theorem 3.3.** *For a connected graph $G$ with $n$ vertices the following statements are equivalent.*
*(a) $G$ is a tree*
*(b) $G$ has $n-1$ edges*
*(c) There is a unique path between any two given vertices in $G$.*

*Proof.* $(a) \implies (b)$.
We will use induction on the number of vertices.
For one edge connected graph, we have two vertices. therefore $(b)$ is true.
By induction, let us assume that (b) is true for all graphs $G$ with $|V(G) \leq n$. To prove (b) for $|V(G) = n$
First claim, removing an edge $e$ from $G$ (not any vertex) leaves G disconnected with two component. Let an edge $uv$ is removed from $G$ then $G \setminus uv$ is disconnected with two component. Suppose not, then there is a path $\gamma := u \to$ from $u$ to $v$, say of the form $\gamma := u \to v_1 \to v_2, \cdots, .. .$ Compositing/ concatenating the edge/path $u \to v$ with $\gamma$ we obtain a cycle in $G$, but G is a tree, hence our supposition cannot be true. Hence the claim.
Now suppose $G \setminus e = G_1 \cup G_2$, and the components $G_1$ and $G_2$ have $n_1$ and $n_2$ vertices respe
By induction $|E(G_1)| = V(G_1) - 1$ and $|E(G_2)| = V(G_2) - 1$. Adding these, we obtain

$$\begin{aligned} |E(G_1)| + |E(G_2)| &= V(G_1) + V(G_2) - 2 \\ \implies E(G) - 1 &= V(G) - 2 \\ \implies E(G) &= V(G) - 1 \end{aligned}$$

$(b) \implies (a)$
Suppose. $|E(G)| = n - 1$ but $G$ is not a tree. We remove (one or more ) edges from cycles to obtain a tree $T_G$ (also called spanning tree), so $|E(G)| > |E(T_G)|$ . On the other hand, $|V(T_G)| = n$, so, by the above implication $|E(T_G)| = n - 1$. It follows that $|E(G)| > n - 1$, a contradiction, hence $G$ is a tree.
$(a) \implies (c)$
Suppose we have two distinct paths $\gamma_1$ and $\gamma_2$ from $u$ to t $v$ in a tree, then the concatenation $\gamma_2^{-1} \circ \gamma_1$ (walk from $u$ to $v$ along $\gamma_1$ first return to $u$ along $\gamma_2$) produces a cycle, which we cannot have in a tree. Hence the implication.
$(c) \implies (a)$,
Suppose we have a cycle $(v_1, v_2, \cdots v_n, v_1)$ in G with property (3) then we can form two paths from $v_1$ to $v_2$ by traversing in two opposite directions, one is $v_1 \to v_2$ and the other is $v_1 \to v_n \to \cdots, v_2$

□

A **minimally connected graph** is a connected graph where the removal of any single edge results in the graph becoming disconnected. It is clear from the proof above that a tree is minimally connected, and vice-versa.

**Theorem 3.4.** *Any tree with at least two vertices has at least two pendant vertices (also called leaves).*

*Proof.* Consider a maximal path $\gamma$ in $T$. If $u$ and $v$ are the end vertices of $\gamma$ , then the only neighbour of u must be the vertex adjacent to $u$ on $\gamma$. If $u$ was adjacent to any other vertex on $\gamma$, then T would contain a cycle; if $u$ was adjacent to any vertex $w$ not on $\gamma$, extending $\gamma$ to $w$ would yield a path longer than $\gamma$, thereby contradicting the maximality of $\gamma$. Hence $d(u) = 1$, and by the same argument $d(v) = 1$.

**Alternative proof:** Let $T$ be a tree with $n$ vertices. By the previous theorem it has $n-1$ edges. By the Handshake Lemma

$$2(n-1) = \sum_{v \in V(G)} d(v)$$

Let's assume $T$ has exactly one leaf, which means every other vertex has degree at least 2. Consequently,

$$2(n-1) = \sum_{v \in V(G)} d(v) \geq 2(n-1) + 1$$

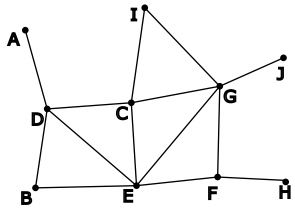
A contradiction. The same contradiction arises if there is no leaf in $T$. Hence proved.

□

**Definition 3.5.** *For a graph $G$, length of a path in $G$ is defined as the number of edges contained in the path. The distance between two vertices $u$ and $v$ in $G$ is the minimum of the length of all possible paths between $u$ and $v$. Eccentricity of any particular vertex $v$ in $G$ is the maximum distance from vertex $v$ to any other vertex $w$,*

$$\mathcal{E}(v) = \max_{u \in G} d(u, v)$$

*Center of $G$ are the vertices having minimum ecentricity. Radius of $G$ is the minimum of the ecentricities of all its vertices.*




The ecentricity of some of the vertices in the above graph is given by: $\mathcal{E}(G) = 3$ ( as $d(E, B) = 3$) $\mathcal{E}(G) = 3$ (as $d(C, H) = 3$), $\mathcal{E}(G) = 2$.
It is not difficult to verify that the vertex $E$ has the least ecentricity. Hence, the center of this graph is the vertex $E$.

**Theorem 3.6.** *Every tree has either one or two centers.*

*Proof.* Let $T$ be a tree. By the previous lemma T has at least two leaves of a tree. Let $T'$ be the tree obtained by removing the leaves of $T$. Then we note that, for any vertex $v$ in $T'$, the ecentricity of $v$ in $T'$ is one less than the ecentricity of $v$ in $T$. Hence $T'$ has the same set of vertices as $T$. The theorem now follows by induction. □

**Definition 3.7.** *A spanning tree $T$ of an undirected graph $G$ is a subgraph that is a tree which includes all of the vertices of $G$.*

**Theorem 3.8.** *Every connected graph has a spanning tree.*

*Proof.* By induction on the number of edges. If $G$ is connected and has zero edges, it is a single vertex, so $G$ is already a tree.

Now suppose $G$ has $m \geq 1$ edges. If $G$ is a tree, it is its own spanning tree. Otherwise, $G$ contains a cycle; remove one edge of this cycle. The resulting graph $G'$ is still connected and has fewer edges, so it has a spanning tree; this is also a spanning tree for $G$. □

In general, spanning trees are not unique, that is, a graph may have many spanning trees. It is possible for some edges to be in every spanning tree even if there are multiple spanning trees. For example, any pendant edge must be in every spanning tree, as must any edge whose removal disconnects the graph.

A *Prüfer sequence* is a sequence of $n-2$ numbers, each being one of the numbers 1 through n. The set of these sequences is denoted by $P_n$. The set number of trees with $n$ labelled vertices is denoted by $T_n$.

**Theorem 3.9.** *(Cayley's Formula) The number of trees with n labelled vertices is* $n^{n-2}$.

*Proof.* We are going to find a bijection between the set of Prüfer sequences and the set of spanning trees. The following is an algorithm that can be used to encode any tree into a Prüfer sequence:

1. Take any tree, $T \in T_n$, whose vertices are labeled from 1 to n in any manner.
2. Take the vertex with the smallest label whose degree is equal to 1, delete it from the tree and write down the value of its only neighbor. (Note: above we showed that any tree must have at least two vertices of degree 1.)
3. Repeat this process with the new, smaller tree. Continue until only one vertex remains.

This algorithm will give us a sequence of $n-1$ terms. Since we already know the number of vertices in our graph by the length of our sequence, we can drop the last term as it is redundant. So now we have a sequence of $n-2$ elements encoded from our tree.
(every vertex has degree equal to 1 + a, where a is the number of times that vertex appears in our sequence.)

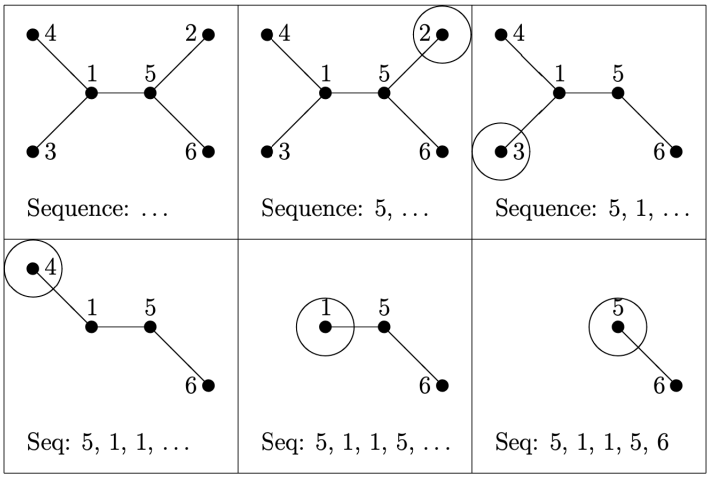


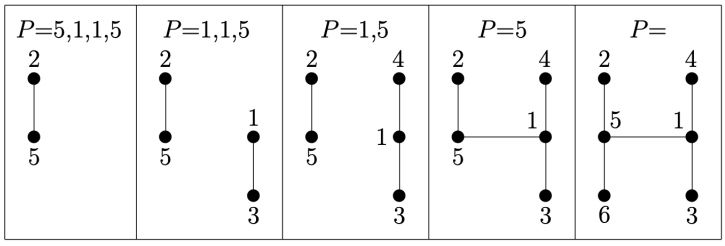

**Fig. 3.1.** Encoding of a tree by Prüfer sequence

Each tree gives a unique Prüfer sequece (Reconstruction of tree from a Prüfer sequece) : Given a Prüfer sequence, let $R$ be the list of numbers from 1 to n (in increasing order). Draw

n vertices with labels 1 to n. Then perform the following steps.
1. Draw an edge between the vertex labeled by the first number in $R$ that is NOT in $P$ and the vertex labeled by the first number in P.
2. Update $R$ by removing its first number (the minimum) and also update $P$ by removing its first number. Then repeat step 1.
3. Do this until there is no number left in $P$. Finally, remember to connect the remaining isolated vertex (only one, why?) with the vertex labelled with the last number of the orginal sequence $P$.
So, we have a bijective function between $T_n \to P_n$. Since, there are $n^{n-2}$ Prüfer sequence for any given $n$, so $|P_n| = n^{n-2}$, and thus $|T_n| = n^{n-2}$.

Tree reconstruction illustrated: click here

Let's reconstruct our original tree from our sequence, $P = 5, 1, 1, 5$:




□

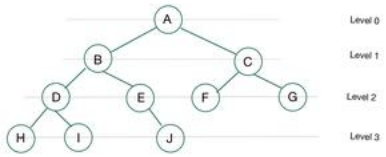
**Example 3.10.** *Supplementary materials on Prüfer sequence. click here*

**Corollary 3.11.** *The number of spanning tree in a complete graph with n labelled vertices is $n^{n-2}$.*

**Definition 3.12.** *An edge in a spanning tree $T$ is called a **branch** of the spanning tree $T$. If an edge of a connected graph $G$ is not a branch of a spanning tree $T$ of $G$ then the edge is called a **chord** of $T$. A fundamental circuit is obtained by adding a chord to a spanning tree.*

## 3.1. **Binary trees.**

**Definition 3.13.** *A rooted tree is a tree in which one vertex has been designated the root. The level of a vertex in a rooted tree refers to its distance from the root. Two ends of an edge clearly lies in two consecutive level, i.e., if lower level end of an edge has level $k$ then the other end has level $k+1$. The former vertex is called the parent vertex and the later its child. In the context of rooted trees, the term "node" is more in usage than "vertex" in the literatures. And Pendant vertices in a tree are also called leaves. An internal node/vertex is any vertex of a rooted tree that has child nodes.*
*A binary tree is a rooted tree whose vertices has degree at most 3 and root has degree exactly 2.*




**Various types of Binary trees**

- **Full Binary Tree:** A binary tree where each node has either 0 or 2 children.
- **Complete Binary Tree:**A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible
- **Perfect Binary Tree:** A binary tree in which all the internal nodes have two children, and all leaf nodes are at the same level.

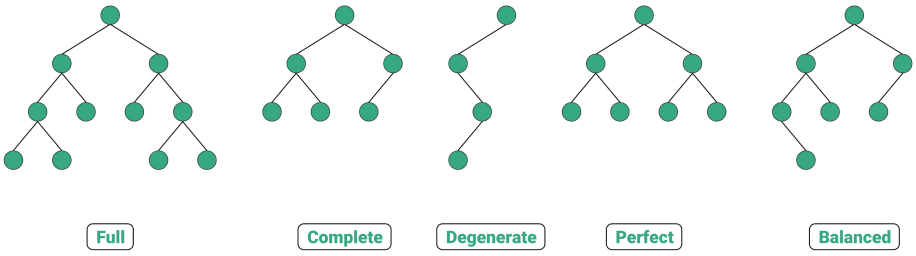



**Fig. 3.2.** Various types of binary trees

- **Height of the tree:** The height of a rooted tree is the number of edges in the longest path from the root to a leaf. A tree with only one vertex has a height of 0.
- **Number of vertex:** A binary tree of height $h$ has at most $2^{h+1} - 1$ nodes.
- **Number of leaves:** In a full binary tree, the number of leaves is $L = \frac{N+1}{2}$, where $N$ is the total number of nodes.

**Theorem 3.14.** *The number of leaves in a binary tree with $n$ vertices is at least 2 and at most $\lceil \frac{n}{2} \rceil$*

*Proof.* The minimum number is guaranteed by Theorem 3.4.

Let $T$ denote a tree with $n$ vertices. Let $I(T)$ and $L(T)$ denote the set of internal vertices and leaves of $T$, respectively.
Then

$$|I(T)| + |L(T)| = |T| = n \tag{3.1}$$

We will use a double counting argument. Let $S$ denote the set of pair of vertices $(v_p, v_c)$, where $v_p$ is parent of $v_c$. Since in a binary tree each internal vertex has at most 2 child,

$$|S| \leq 2|I(t)|$$

Further, each child has one parent except the root; consequently,

$$|S| = n - 1$$

Equating the above two counts of $|S|$, we obtain, $|I(T)| \geq \frac{n-1}{2}$. It follows that

$$|L(T)| \leq n - \frac{n-1}{2} = \frac{n+1}{2}$$

Since number of leaves can only be an integer, hence the theorem.

□

3.2. **Practice problem set 3.** TBA

## 4. Connectivity

**Definition 4.1.** *Let $G = (V, E)$ be a connected graph. A subset $E'$ of $E$ is called a cut set of $G$ if deletion of all the edges of $E'$ from $G$ makes $G$ disconnect. A minimal cut set is a cutset so that no proper subset of it is again a cutset.*

**Example 4.2.** *A minimum cut-set of the complete graph $K_n$ consistes of any of its $n-1$ vertices.*

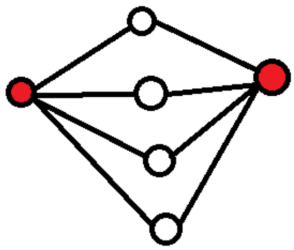
**Proposition 4.3.** *Every cut-set of a connected graph contains at least one branch from every spanning tree.*

*Proof.* If a cut-set $S$ does not contain any branch of a spanning tree $T$ in a graph $G$, i.e, $G \setminus S$ contains $T$, then $G \setminus S$ is clearly a connected graph as spanning tree connects every vertex of $G$. So, $S$ must contain at least one branch from every spanning tree.
Suppose $S$ contains one branch of every spanning tree. Then $G \setminus S$ doesn't contain any spanning tree of $G$. Therefore $G \setminus S$ disconnected. Hence $S$ is a cut set. □

**Theorem 4.4.** *(Menger's theorem, edge connectivity version) The size of a minimum cut-set separating two vertices $u$ and $v$ in a graph is equal to the maximum number of edge-disjoint paths from $u$ to $v$.*

*Proof.* Proof is not included in syllabus. Curious reader can check out Diestel's book, Theorem. 3.3.1. □

4.1. **Covering and independent sets.** A **vertex cover** of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. A vertex cover is minimal if it does not contain a proper subset which is also a vertex cover.
**An independent set** of a graph, is a set of vertices in a graph, no two of which are adjacent. An independent set that is not a proper subset of another independent set is called maximal.



The following facts are now obvious:
1. A set of vertices is a vertex cover if and only if its complement is an independent set. In the above graph two red color vertices clearly forms a minimum vertex cover, and (therefore)

four uncolor vertices forms an independent set.
2. Consequently, the number of vertices of a graph is equal to its minimum vertex cover number plus the size of a maximum independent set.
**Notes:** In computer science, several computational problems related to independent sets have been studied.
• The independent set decision problem is NP-complete, and hence it is not believed that there is an efficient algorithm for solving it.
• The maximum independent set problem is NP-hard and it is also hard to approximate.

4.2. **Matchings.** Given a graph G, a matching S in G is a set of vertices such that no two vertices are adjacent. Equivalently, a matching is a set of edges such that no two edges share common vertices and none are loops.
A matching $S$ of G is called a perfect matching if every vertex of G is covered by an edge of $S$.

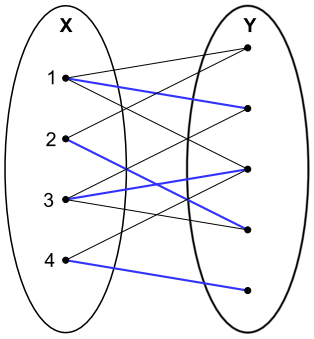



**Fig. 4.1.** blue edges represents a matching but not a perfect matching

Recall that a bipartite graph $G$ with bipartition $X, Y$ is a graph where the vertices can be divided into two subsets $X$ and $Y$ such that all the edges of $G$ connects a vertex of $X$ to a vertex of $Y$. For a matching $S$ in $G$, if every vertex in $X$ is covered by an edge of S, then we say that $S$ is a matching from $X$ into $Y$.
Firstly, given $A \subset V(G)$ define $N_G(A)$ denote the set of vertices of $G$ that are adjacent to some vertex in $A$, that is,

$$N_G(A) = \{v \in V(G) | vw \in E(G) \text{ for some } w \in T\}.$$

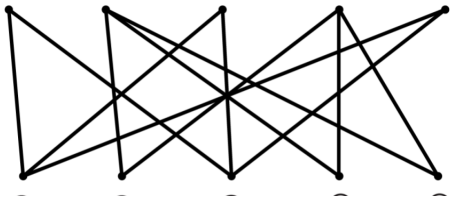
**Theorem 4.5** (Hall's Marriage Theorem)**.** *Let $G$ be a bipartite graph with bipartition $X, Y$. Then there is a matching in $G$ from $X$ to $Y$ if and only if:*

$$|N_G(A)| \geq |A|, \forall\ A \subseteq X$$

*This condition is known as Hall's condition.*

**Corollary 4.6.** *For a connected bipartite graph $G$ with bipartition $X$ and $Y$, there is a perfect matching in $G$ if and only if $|X| = |Y|$, ($|X|$ = vertices in $X$), and the Hall's condition is satisfied for either $X$ or $Y$.*

Ex. Using Hall's marriage theorem or the corollary above, show that the following graph does not a perfect matching.

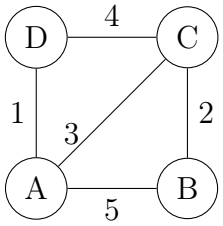


Solution. Count the number of neighbours the subset of the vertices in the top row consisting of 2nd and 4th vertex, and count $N_G(A)$.

4.3. **Practice problem set 4.** 1. Find a vertex cover of the complete graph $k_5$ and the Petersen graph.
TBA

## 5. Directed and weighted graphs

A weighted graph is a graph in which each edge is assigned a numerical value, called a weight. These weights can represent various real-world quantities such as distances, costs, or capacities. Weighted graphs are essential in modeling many practical problems, including network flow, shortest path problems, and minimum spanning tree problems.
For a weighted graph with vertex set $U = \{u_1, \cdots, u_n\}$, the adjacency matrix is a square $n \times n$ matrix $A$ such that its element $A_{ij}$ is the weight of the edge from vertex $u_i$ to vertex $u_j$, and zero when there is no edge.
**Example:** Consider the following weighted graph with 4 vertices. Its adjacency matrix is shown on the right.



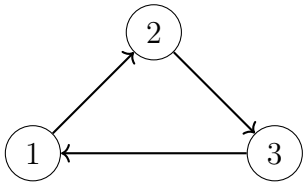

$$A_G = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 5 & 3 & 1 \\ 5 & 0 & 2 & \infty \\ 3 & 2 & 0 & 4 \\ 1 & \infty & 4 & 0 \end{pmatrix} \end{array}$$

A **directed graph** (or digraph) is a pair $G = (V, E)$, where:

- $V$ is a set of vertices.
- $E$ is a set of directed edges, where each edge is an ordered pair of vertices. Specifically, an edge from vertex $u$ to vertex $v$ is denoted by $(u, v)$.

Unlike undirected graphs, in a directed graph, the edge $(u, v)$ indicates a direction from $u$ to $v$, which implies that $(u, v) \neq (v, u)$ unless there are two distinct edges with opposite directions.
**Example :** Consider a directed graph $G = (V, E)$, where:

- $V = \{1, 2, 3\}$.
- $E = \{(1, 2), (2, 3), (3, 1)\}$.




$$A_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \end{array}$$

**Fig. 5.1.** G

This graph contains three vertices and three directed edges, forming a directed cycle. The edges specify directions between the vertices.

One can define the adjacency matrix $A(G)$ of a directed graph $G$ such that a non-zero element $A(G)_{ij}$ indicates a directed edge from i to j. The adjacency matrix of a directed graph can be asymmetric.

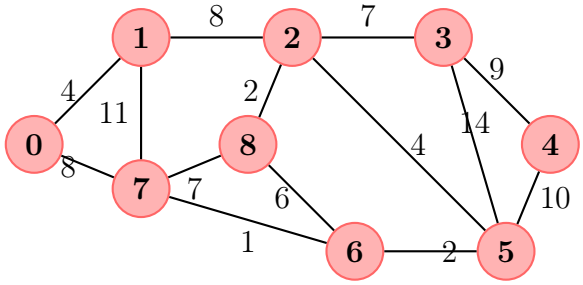
## 5.1. **Graph variants.**

- A *multigraph* is a pair $G = (V, E)$ where $V$ is a finite set and $E$ is a multiset of elements from $\binom{V}{1} \cup \binom{V}{2}$, i.e., we also allow loops and multiedges.
- A *hypergraph* is a pair $H = (X, E)$ where $X$ is a finite set and $E \subseteq 2^X \setminus \{\emptyset\}$.

## 6. Graph Algorithms (Minimum path, Minimum spanning tree (MST), Network, Flow Problems)

6.1. **Dijkstra Algorithm.** Dijkstra's algorithm for finding the shortest paths between vertices in a weighted graph. It was conceived by W. Dijkstra in 1956.
It can also be used to find the shortest path to a specific destination vertices, by terminating the algorithm once the shortest path to the destination node is known. Given a weighted graph and a source vertex in the graph, find the shortest paths from the source to all the other vertices in the given graph.
**Description of the Algorithm :**
Create a set spSet (shortest path tree set) that keeps track of vertices included in the shortest path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty. Assign a distance value to all vertices in the input graph. Initialize all distance values as infinite . Assign the distance value as 0 for the source vertex so that it is picked first. While sptSet doesn't include all vertices Pick a vertex u that is not there in sptSet and has a minimum distance value. Include u to sptSet . Then update the distance value of all adjacent vertices of u . To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v , is less than the distance value of v , then update the distance value of v.
Note: The algorithm is applicable to only the graphs that does not contain any edge with negative weight.
**Visualisation**: click here

6.2. **Example.** The following graph contains 9 vertices (0 to 8) with weighted edges as shown in the figure below. We will be using Dijkstra's algorithm to find the shortest path from vertex "0" to vertex "4".




**Step-by-Step Execution:**
We initialize the distances to all vertices as infinity, except for the source vertex (0), which is set to 0. The table below shows the progress of the algorithm at each step.
**Initial Setup**

- **Source**: Vertex 0
- Distances: $\text{dist}(0) = 0$, all other vertices set to $\infty$.
- Unvisited Set: {0, 1, 2, 3, 4, 5, 6, 7, 8}

**Step 1: Visit Vertex 0**

- Neighbors: $\{1, 7\}$
- Update distances:

$$\text{dist}(1) = \min(\infty, 0 + 4) = 4$$

$$\text{dist}(7) = \min(\infty, 0 + 8) = 8$$

Unvisited Set: {1, 2, 3, 4, 5, 6, 7, 8}

**Step 2: Visit Vertex 1**

- Current vertex: 1 (minimum distance = 4)
- Neighbors: $\{0, 2, 7\}$
- Update distances:

$$\text{dist}(2) = \min(\infty, 4 + 8) = 12$$
$$\text{dist}(7) = \min(8, 4 + 11) = 8$$

Unvisited Set: {2, 3, 4, 5, 6, 7, 8}

**Step 3: Visit Vertex 7**

- Current vertex: 7 (minimum distance = 8)
- Neighbors: $\{0, 1, 6, 8\}$
- Update distances:

$$\text{dist}(6) = \min(\infty, 8 + 1) = 9$$
$$\text{dist}(8) = \min(\infty, 8 + 7) = 15$$

Unvisited Set: {2, 3, 4, 5, 6, 8}

**Step 4: Visit Vertex 6**

- Current vertex: 6 (minimum distance = 9)
- Neighbors: $\{5, 7, 8\}$
- Update distances:

$$\text{dist}(5) = \min(\infty, 9 + 2) = 11$$
$$\text{dist}(8) = \min(15, 8 + 6) = 15$$

Unvisited Set: {2, 3, 4, 5, 8}

**Step 5: Visit Vertex 5**

- Current vertex: 5 (minimum distance = 11)
- Neighbors: $\{2, 3, 4, 6\}$
- Update distances:

$$\text{dist}(2) = \min(\infty, 11 + 1) = 12$$
$$\text{dist}(3) = \min(\infty, 11 + 14) = 25$$
$$\text{dist}(4) = \min(\infty, 11 + 10) = 21$$
$$\text{dist}(4) = \min(15, 11 + 6) = 15$$

Unvisited Set: {2, 3, 4, 8}

**Step 6: Visit Vertex 2**

- Current vertex: 2 (minimum distance = 12)
- Neighbors: $\{1, 3, 5, 8\}$
- Update distances:

$$\text{dist}(3) = \min(\infty, 12 + 7) = 19$$

Unvisited Set: {3, 4, 8}

**Step 7: Visit Vertex 3**

- Current vertex: 3 (minimum distance = 19)
- Neighbors: $\{2, 4, 5\}$
- Update distances:

$$\text{dist}(4) = \min(21, 19 + 9) = 21$$

Unvisited Set: {4, 8}

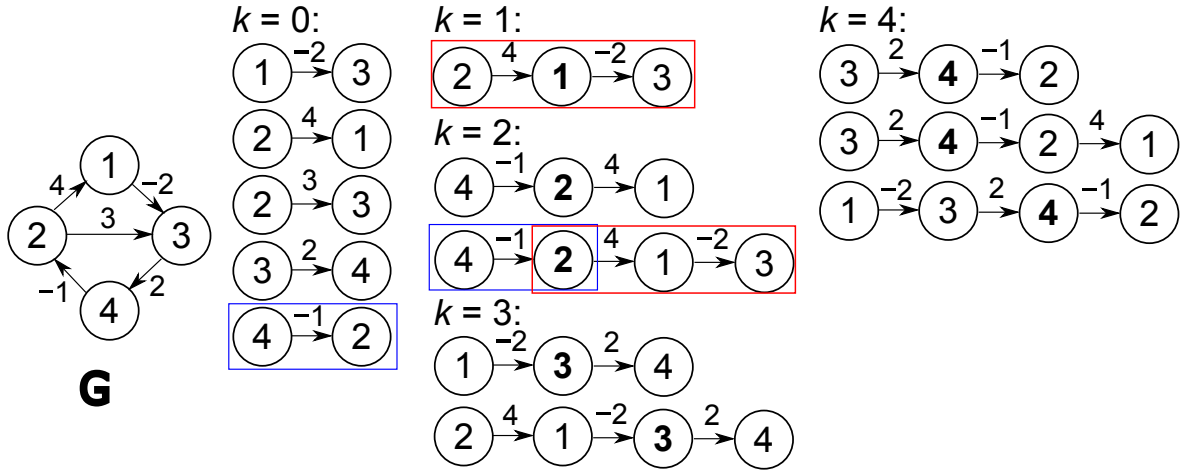
**Step 8: Visit Vertex 4**

- Current vertex: 4 (minimum distance = 21)
- Neighbors: $\{3, 5\}$

The destination vertex "4" has been reached. Hence the distance from vertex 0 to vertex 4 is 21. that the shortest path from vertex 0 to vertex 4 is $0 \to 7 \to 6 \to 5 \to 2 \to 3 \to 4$ and

6.3. **Floyd-Warshall Algorithm.** It is an algorithm for finding shortest paths in a directed weighted graph. There can be edges with positive or negative weights but no negative cycles. The algorithm finds the lengths (summed weights) of shortest paths between all pairs of vertices. Although it does not directly gives the details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm.
Description: Given a a directed weighted graph (equivalently its adjacency matrix), initialize the solution matrix same as the given adjacency matrix as a first step. Then update the solution matrix by considering all vertices as an intermediate vertex. The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices $\{0, 1, 2, \cdots, k-1\}$ as intermediate vertices. For every pair $(i, j)$ of the source and destination vertices respectively, there are two possible cases. k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is. k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j], if dist[i][j]> dist[i][k] + dist[k][j]

**Example:** Let $G$ be the weighted graph given on the left of the following figure.




The distance matrix at each iteration of $k$, with the updated distances in bold is shown below.

Matrix for $k = 0$

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 0 | $\infty$ | $-2$ | $\infty$ |
| $B$ | 4 | 0 | 3 | $\infty$ |
| $C$ | $\infty$ | $\infty$ | 0 | 2 |
| $D$ | $\infty$ | $-1$ | $\infty$ | 0 |

Matrix for $k = 1$

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 0 | $\infty$ | $-2$ | $\infty$ |
| $B$ | 4 | 0 | **2** | $\infty$ |
| $C$ | $\infty$ | $\infty$ | 0 | 2 |
| $D$ | $\infty$ | $-1$ | $\infty$ | 0 |

Matrix for $k = 2$

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 0 | $\infty$ | $-2$ | $\infty$ |
| $B$ | 4 | 0 | 2 | $\infty$ |
| $C$ | $\infty$ | $\infty$ | 0 | 2 |
| $D$ | **3** | $-1$ | **1** | 0 |

Matrix for $k = 3$

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 0 | $\infty$ | $-2$ | **0** |
| $B$ | 4 | 0 | 2 | **4** |
| $C$ | $\infty$ | $\infty$ | 0 | 2 |
| $D$ | 3 | $-1$ | 1 | 0 |

Matrix for $k = 4$

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 0 | $-1$ | $-2$ | 0 |
| $B$ | 4 | 0 | 2 | 4 |
| $C$ | **5** | **1** | 0 | 2 |
| $D$ | 3 | $-1$ | 1 | 0 |

**Comparison:** For graphs with non-negative edge weights, Dijkstra's algorithm can be used to find all shortest paths from a single vertex with running time $\Theta(|E| + |V| \log |V|)$. Thus, running Dijkstra starting at each vertex takes time $\Theta(|E||V| + |V|^2 \log |V|)$. Since $|E| = O(|V|^2)$, this yields a worst-case running time of repeated Dijkstra of $O(|V|^3)$. While this matches the asymptotic worst-case running time of the Floyd-Warshall algorithm, the constants involved matter quite a lot. When a graph is dense (i.e., $|E| \approx |V|^2$), the Floyd-Warshall algorithm tends to perform better in practice. When the graph is sparse (i.e., $|E|$ is significantly smaller than $|V|^2$), Dijkstra tends to dominate.

6.4. **DFS and BFS.** Breadth-First Search (BFS) and Depth-First Search (DFS) are two fundamental algorithms used for traversing or searching graphs.

**DFS:** The algorithm starts at the root node and explores as far as possible along each branch before backtracking. This is similar to a tree, where we first completely traverse the left subtree and then move to the right subtree. The key difference is that, unlike trees, graphs may contain cycles (a node may be visited more than once). To avoid processing a node multiple times, we use a boolean visited array.

- DFS algorithm can be used to implement the topological sorting.
- It can be used to find the paths between two vertices.
- It can also be used to detect cycles in the graph.
- DFS algorithm is also used for one solution puzzles.

- DFS is used to determine if a graph is bipartite or not.

**Breath-first search (BFS):** It starts at the root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

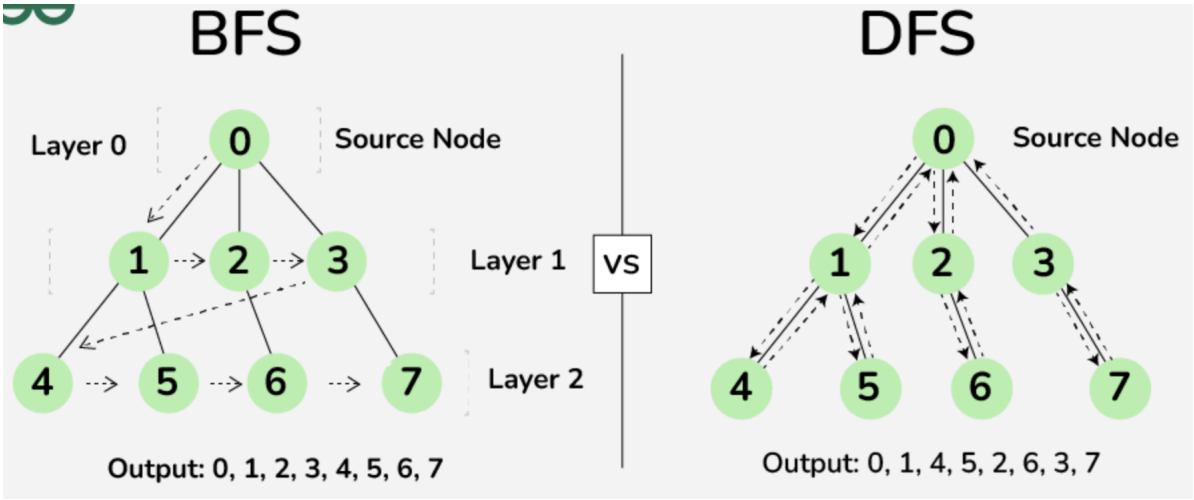



**Fig. 6.1.** Comparison between BFS and DFS

BFS itself can be used to detect cycle in a directed and undirected graph, find shortest path in an unweighted graph and many more problems.

6.5. **Algorithms to find minimum spanning tree.** Minimum spanning tree is a spanning tree whose sum of edge weights is the smallest possible. **Prim's algorithm** is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. The algorithm for MST is as follows :
Step 1. Intialize with an arbitrary vertex.
Step 2. Perform the following steps till there are vertices that are not included in the MST . This vertices are known as fringe vertex
Step 3. Find edges connecting any tree vertex with the fringe vertices and find the minimum among them.
Step 4. Add the chosen edge to the MST if it does not form any cycle.

**Visualisation**: click here
**Kruskal's Algorithm**:
Step 1.Sort all the edges in non-decreasing order of their weight.
Step 2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
Step 3. Repeat step 2. until there are edges in the spanning tree.
**Visualisation:** click here

6.6. **Practice problem set 6.** Q 1. The given statements are false. For each, draw a small counterexample with $\leq 5$ vertices.
(i) A minimum spanning tree cannot include the maximum-weight edge in the graph.

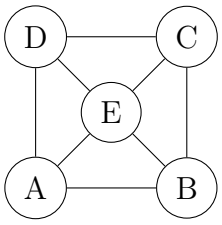
(ii) A weighted undirected graph can only have one (unique) minimum spanning tree.

(iii) For each cut in a connected and weighted undirected graph, a minimum spanning tree can only contain the minimum-weight crossing edge.

## 7. **Planar graphs**

A graph that can be drawn in the plane without edges crossing is **planar**. A graph drawn in the plane is a *plane graph*. A *face or region* of a plane graph is a connected region of $\mathbb{R}^2 \setminus G$. We define the *size* of a region to be the number of edges it has.
The set of regions of a planer graph $G$ is denoted by $R(G)$.

Consider the following graph:




It is clearly planar. It has five vertices, eight edges: $AB, BC, CD, DA, AE, BE, CE$, and $DE$. There are 5 regions in total: Face 1: The outer unbounded region around the entire graph, Face 2: The triangle $ABE$, Face 3: The triangle $BCE$, Face 4: The triangle $CDE$, Face 5: The triangle $DAE$.

**Theorem 7.1** (Euler's Formula)**.** *For a connected planar graph $G$,*

$$|V(G)| - |E(G)| + |R(G)| = 2$$

*Proof.* We prove the formula by induction on $m := |E(G)|$.

**Base case.** If $m = |V(G)| - 1$ then $G$ is a tree. A (plane) tree has exactly one region (the unbounded outer face), so $|R(G)| = 1$. Therefore

$$|V(G)| - |E(G)| + |R(G)| = |V(G)| - (|V(G)| - 1) + 1 = 2,$$

and the formula holds for trees.

**Inductive step.** Fix $n := |V(G)|$. Assume the formula holds for every connected planar graph with fewer than $m$ edges, and let $G$ be a connected planar graph with $m$ edges. If $m = n - 1$ we are in the base case, so assume $m \geq n$. In a connected graph having at least $n$ edges there must be a cycle; hence there exists an edge $e \in E(G)$ that is not a bridge (i.e. $G - e$ remains connected).

Remove such an edge $e$. The graph $G - e$ is connected (i.e., $e$ is not a *bridge*) and planar and has $m - 1$ edges, so by the induction hypothesis

$$|V(G - e)| - |E(G - e)| + |R(G - e)| = 2.$$

Because $e$ is not a bridge we have $|V(G - e)| = |V(G)|$ and $|E(G - e)| = |E(G)| - 1$. In the chosen planar embedding the edge $e$ lies on a cycle and therefore is incident with two distinct faces; removing $e$ merges those two faces into one, so $|R(G - e)| = |R(G)| - 1$. Substituting these equalities into the induction hypothesis gives

$$|V(G)| - (|E(G)| - 1) + (|R(G)| - 1) = 2,$$

which simplifies to

$$|V(G)| - |E(G)| + |R(G)| = 2.$$

This completes the inductive step and hence the proof. □

The following result gives criteria to verify planarity of graphs.

**Theorem 7.2.** *For a planar graph $G$, $|E(G)| \leq 3|G| - 6$. Furthermore, if there is no triangle in $G$, then $|E(G)| \leq 2|G| - 4$.*

*Proof.* Let us denote $|E(G)| = e$ and $|G| = n$. Let $R(G) = \{R_1, R_2, \cdots, R_m\}$ denote all the regions of $G$. We know that for a planar graph

$$deg(R_i) \geq 3$$

This implies

$$\sum deg(R_i) \geq 3m \qquad (1)$$

By handshake lemma for planar graph

$$\sum deg(R_i) = 2e \qquad (2)$$

From (1) and (2), we get

$$3m \leq 2e. \qquad (3)$$

From Euler's formula, we know that

$$m = e - n + 2$$

Using (3) we get

$$\implies (e - n + 2) \leq \frac{2}{3}e$$

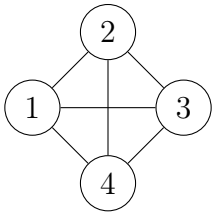
$$\implies e \leq 3n - 6 \qquad (4)$$

this completes the proof of the first assertion.

If there is no triangle in $G$, then $|R_i(G)| \geq 4$. The second assertion then follows very similarly to the above assertion.

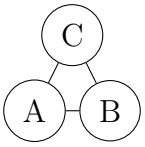
□

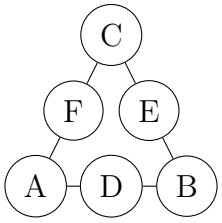
The complete graph $K_4$ on 4 vertices is planar and can be drawn without edge crossings, as shown below.




A **subdivision** of a graph is formed by inserting vertices along its edges. For example, subdividing an edge $(u, v)$ by adding a vertex $w$ gives two edges $(u, w)$ and $(w, v)$.

Consider the complete graph $K_3$ and it's subdivision by adding one vertex in the middle of each edge of $K_3$. This transforms each edge into two edges.






Subdivision does not reduce the inherent "complexity" of a graph. Any attempted planar embedding of a subdivision of a graph would still require the same non-crossing requirements as the original. Adding vertices along the edges does not create additional faces or change the essential connectivity of the graph.

**Theorem 7.3.** *A graph $G$ is planar if and only if $G$ has a subgraph that is a subdivision (or homemorphic) of $K_{3,3}$ or $K_5$.*

*"Only if " part only.* . To show t $K_5$, $K_{3,3}$ are not planar, we can use Theorem 7.3. □

7.1. **Dual graph.** The dual graph $G^*$ of a planar graph $G$ is a graph that has:

- A vertex for each face in $G$ (including the outer face)
- An edge between two vertices in $G^*$ if the corresponding faces in $G$ share an edge

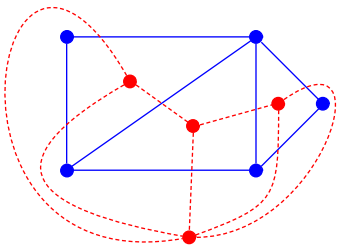


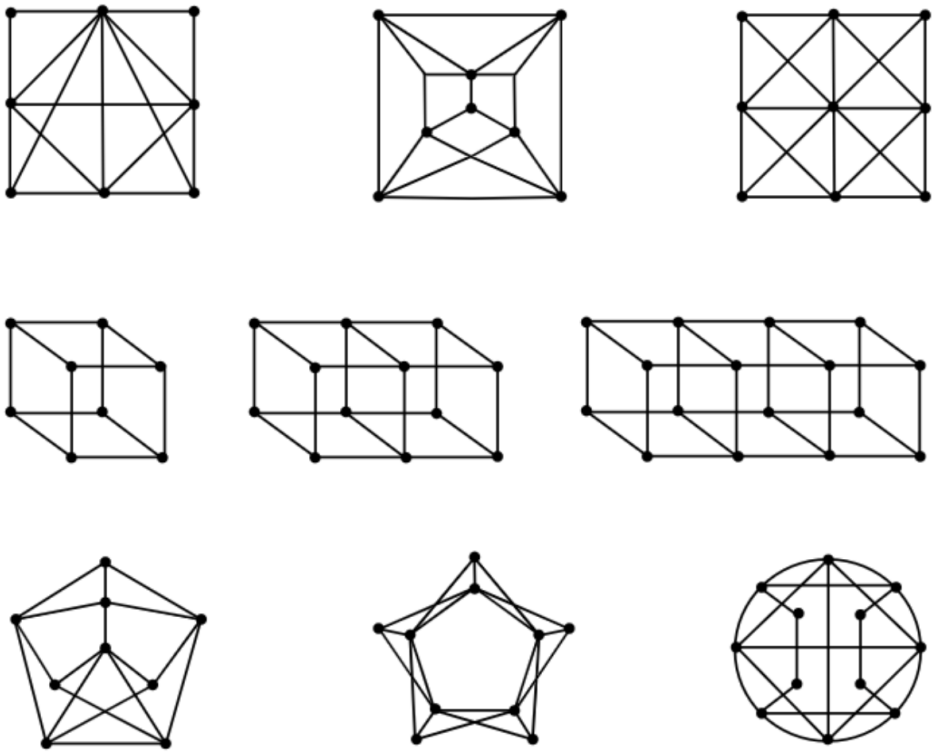
**Fig. 7.1.** Dual graph

**Important Properties:**

- The dual of a planar graph is always planar
- $(G^*)^*$ is isomorphic to $G$ (for a 3-connected planar graph)
- The number of vertices in $G^*$ equals the number of faces in $G$
- The number of faces in $G^*$ equals the number of vertices in $G$
- The number of edges remains the same in both graphs

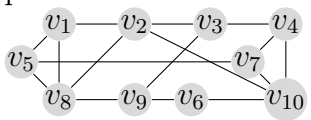
7.2. **Practice problems.** 1. Determine whether or not each of the following graphs is planar. If a graph is planar, exhibit a planar drawing of the graph and verify that Euler's formula holds for this representation of the graph. If a graph is not planar, provide an

argument that proves that the graph cannot be planar.

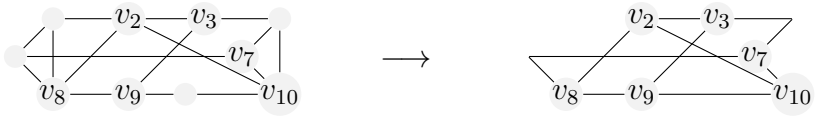


2. For each description given, either draw a planar graph that meets the description or prove that no planar graph can meet the description given:

(a) A simple graph with 5 vertices and 8 edges.
(b) A simple graph with 6 vertices and 13 edges.
(c) A simple bipartite graph with 7 vertices and 10 edges.
(d) A simple bipartite graph with 7 vertices and 11 edges.
(e) Using the Kuratowski theorem or otherwise verify whether the following graph is planar.




**Ans.** The graph is not planar as it contains a subgraph homeomorphic to $K_{3,3}$, as shown below with base sets $A = \{v_2, v_7, v_9\}$, $B = \{v_3, v_8, v_1 0\}$ .



## 8. **Graph Coloring**

In this notes coloring of a graph will always refer to a proper vertex coloring, namely a labeling of the graph's vertices with colors such that no two adjacent vertices have the same color. Since a vertex with a loop could never be properly colored, it is understood that graphs in this context are loopless.
The smallest number of colors needed to color a graph G is called its **chromatic number**, and is denoted by $\gamma(G)$ (often denoted by $\chi(G)$ as well).

**Example 8.1.** *We will color the states of India (or any other country) with minimum number of colors so that no two neghbouring states have the same color. For that we can first construct the dual graph, and then use graph coloring algorithms described further below. The following images show states of India can be colored using only 4 colors. Can you do it with lesser?*

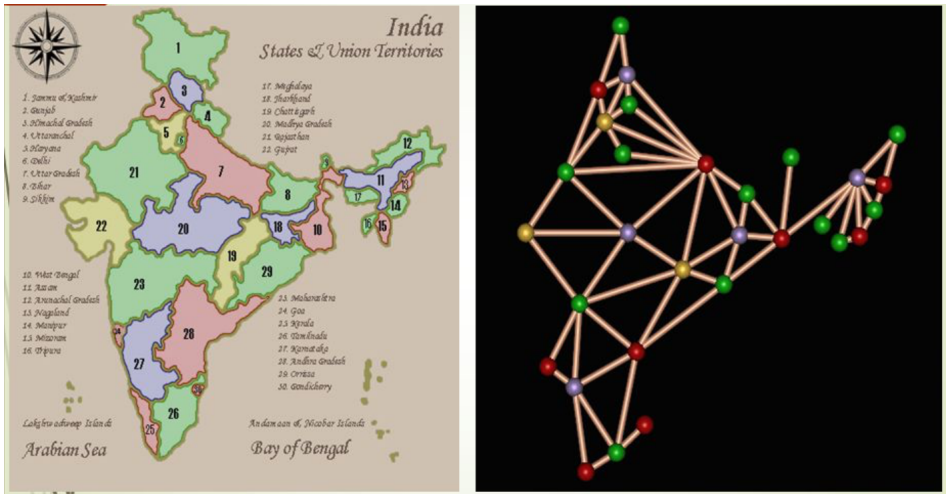



**Fig. 8.1.** States of India and the dual graph

The following observations are the immediate consequences of the above definitions.
1. A graph is 1-chromatic ($\gamma(G) = 1$) if and only if it is a null graph.
2. A graph having at least one edge is at least 2-chromatic.
3. A graph G having n vertices has $\gamma(G) \leq n$.
4. If H is subgraph of a graph G, then $\gamma(H) \leq \gamma(G)$.
5. A complete graph with n vertices is n-chromatic
6. A cycle of length $n \geq 3$ is 2-chromatic if n is even and 3-chromatic if n is odd.

**Theorem 8.2.** *Every tree with $n \geq 2$ vertices is bicolorable (2-chromatic or $\gamma(G) = 2$).*

*Proof.* Let $T$ be a tree with $n \geq 2$ vertices. Consider any vertex $v$ of T and assume T to be rooted at vertex $v$ . Assign color 1 to $v$. Then assign color 2 to all vertices which are adjacent to $v$. Let $v_1, v_2, \cdots, v_r$ be the vertices which have been assigned color 2. Now assign color 1 to all the vertices which are adjacent to $v_1, v_2, \cdots, v_r$. Continue this process till every vertex in T has been assigned the color. We observe that in T all vertices at odd distances from $v$ have color 2, and $v$ and vertices at even distances from $v$ have color 1. Therefore along any path in T , the vertices are of alternating colors. Since there is one and only one path between any two vertices in a tree, no two adjacent vertices have the same color. Thus T is colored with two colors. Hence T is 2-chromatic. □

The converse of the above theorem is not true, i. e., every 2-chromatic graph need not be a tree. The next result charaterises 2-chromatic graphs.

**Theorem 8.3.** *A graph is 2-chromatic ($\gamma(G) = 2$) if and only if it has no odd cycles.*

*Proof.* Let G be a connected graph with cycles of only even length and let T be a spanning tree in G. Then, by Theorem 8.2, T can be coloured with two colours. Now add the chords to T one by one. As G contains cycles of even length only, the end vertices of every chord get different colours of T . Thus G is coloured with two colours and hence is 2-chromatic. If $G$ is disconnected we can use the same argument for each of its components.
Conversely, let G be 2-chromatic then $G$ cannot contain an odd-cycle as to color an odd cycle one requires at least 3 colors. □

**A greedy algorithm for graph coloring.** Let $\Delta(G)$ denote the maximum degree of vertices of a given graph $G$. For a given vertex ordering $\{v_1, v_2, \cdots, v_n\}$ of $G$, and ordering of $\Delta(G) + 1$ many colors $C_1, C_2, \cdots, C_{\Delta(G)+1}$, we color first vertex $(v_1)$ with the first color $(C_1)$, and then each vertex is given the color with the smallest number that is not already used by one of its neighbors.

**Remark 8.4.** *This algorithm never uses more than $d(v_i) + 1$ colors to color $i$-th vertex $v_i$. Therefore, we have*

$$1 \leq \gamma(G) \leq \Delta(G) + 1.$$

The following result due to Brooks improves the above bound in most cases.

**Theorem 8.5** (Only the statement)**.** *For a connected graph $G$,*

$$\gamma(G) \leq \Delta(G)$$

*unless $G$ is a complete graph or an odd cycle.*

**Notes:**Finding the chromatic number of a graph is NP-Complete. It is NP-Complete even to determine if a given graph is 3-colorable (and also to find a coloring).

For planar graphs, e.g., dual graphs of maps of the states of a country, we have a much better result to determine the chromatic numbers. Here we state and prove a weaker version of such a result.
The following lemma play crucial role proving theorems about chromatic number of planar graphs.

**Lemma 8.6.** *A planar graph $G$ contains a vertex of degree at most* 5*.*

*Proof.* We may assume that $G$ is connected. Suppose all the vertex of $G$ has degree 6 or more, then by hand-shake lemma

$$2|E(G)| \geq 6|V(G)|.$$

On the other hand, for a planar graph $G$ we have seen earlier that

$$|E(G)| \leq 3|V(G)| - 3 \implies 2|E(G)| \leq 6|V(G) - 6|$$

So there is a contradiction. Hence the proof of the lemma.

□

**Theorem 8.7** (Five color theorem)**.** *For any planar graph $G$, $\gamma(G) \leq 5$*

*Proof.* (**Not included in Syllabus**) We proceed by induction on $n = |V(G)|$. The claim is trivial for $n \leq 5$. Assume the statement holds for all planar graphs with fewer than $n$ vertices, and let $G$ be a planar graph with $n \geq 6$ vertices.
By the above Lemma 8.6, $G$ has a vertex $v$ with $\deg(v) \leq 5$. Let $G' = G - v$ be the graph obtained by deleting $v$. By induction, $G'$ admits a proper 5-coloring

$$c : V(G') \to \{1, 2, 3, 4, 5\}.$$

If among the (at most five) neighbors of $v$ fewer than five colors appear, assign to $v$ any missing color and we are done. Thus we may assume $\deg(v) = 5$, and that the neighbors of $v$ in their cyclic (planar) order around $v$ are

$$v_1, v_2, v_3, v_4, v_5,$$

colored respectively

$$c(v_1) = 1,\ c(v_2) = 2,\ c(v_3) = 3,\ c(v_4) = 4,\ c(v_5) = 5.$$

Consider the subgraph $G'_{1,3}$ induced by all vertices colored 1 or 3 in $G'$ (edges between them retained). If $v_1$ and $v_3$ lie in *different* connected components of $G'_{1,3}$, then interchange colors 1 and 3 within the component containing $v_1$. This preserves proper coloring of $G'$ and frees color 1 among the neighbors of $v$, so we can set $c(v) = 1$ and obtain a valid 5-coloring of $G$.

Hence we may assume $v_1$ and $v_3$ lie in the *same* component of $G'_{1,3}$, so there is a path $P_{1,3}$ in $G'$ between $v_1$ and $v_3$ whose vertices alternate colors 1 and 3 (a *Kempe chain*).

Now consider the subgraph $G'_{2,4}$ induced by vertices colored 2 or 4. If $v_2$ and $v_4$ lie in different components of $G'_{2,4}$, swap colors 2 and 4 in the component containing $v_2$. This frees color 2 among the neighbors of $v$, allowing $c(v) = 2$ and finishing the coloring.

It remains to handle the case where both pairs are connected: there is a 2–4 Kempe chain $P_{2,4}$ between $v_2$ and $v_4$ and a 1–3 Kempe chain $P_{1,3}$ between $v_1$ and $v_3$. Because the neighbors $v_1, v_2, v_3, v_4, v_5$ occur around $v$ in cyclic order, any plane embedding forces the two Kempe chains to cross if both exist. But the chains are vertex-disjoint (they use only colors $\{1, 3\}$ and $\{2, 4\}$ respectively) and lie in the planar graph $G'$, so they cannot cross. This contradiction shows that at least one of the two recolorings above is always possible. In either case we extend the 5-coloring of $G'$ to $G$.

By induction, every planar graph is 5-colorable.

□

The following stronger statement of the above theorem is known to be true.

**Theorem 8.8** (four color theorem)**.** *For any planar graph $G$, $\gamma(G) \leq 4$*

**Exercise** Find the chromatic number of the Petersen graph?

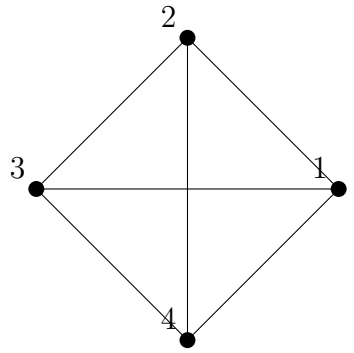
8.1. **Edge Coloring of Graphs.** An **edge coloring** of a graph $G$ is an assignment of colors to the edges of $G$ such that:

- No two adjacent edges share the same color
- The goal is to minimize the number of colors used

Important Results:

- **Edge Chromatic Number** $\gamma'(G)$: Minimum number of colors needed to color the edges

- Vizing's Theorem: For any graph $G$, $\gamma'(G) \in \{\Delta(G), \Delta(G)+1\}$, where $\Delta(G)$ represents the maximum degree of the graph




Possible edge coloring:

- Red edges: $(1,2),(3,4)$
- Blue edges: $(1,3),(2,4)$
- Green edges: $(1,4),(2,3)$
- So, the minimum colors required:3

8.2. **Chromatic partitioning.** Chromatic partitioning refers to the process of partitioning the vertex set of a graph into subsets such that each subset (or part) forms an independent set (no two vertices in the same part are adjacent) and each subset is assigned a color. Clearly, proper coloring of graph yields a chromatic partitioning of a graph. Hence, the chromatic number provides the minimum number of chromatic partitioning requires.

8.3. **Chromatic polynomial.** Chromatic polynomial is a special function that describes the number of ways we can achieve a proper coloring on a graph G given $k$ colors. For a graph G, $P_G(k)$ counts the number of its vertex k-colorings (coloring of its vertices by $k$ colors), if $k < \gamma(G)$, then by convention $P_G(k) = 0$.
**Homework:** What is value of $P_G(\gamma(G))$?

**Proposition 8.9.** *The chromatic polynomial of the complete graph with n-vertices, $K_n$, is given by*

$$P_{K_n}(k) = k(k-1)(k-2)\cdots(k-n+1)$$

*for $k \geq n$.*

*Proof.* We know that for $\gamma(K_n) = n$. Therefore, for $k < n$, $P_G = 0$ by definition. For $k \geq n$, for an arbitrary ordering of vertices $v_1, v_2, v_3, \cdots v_n$ of $K_n$, we have $k$ choices of colors for the first vertex $v_1$, and next $(k-1)$ choices for $v_2$, and so on. Continuing this way, we have $(k-n+1)$ choices of colors for the vertex $v_n$. Thus, the total number of choices to color for all the $n$ vertices is $k(k-1)(k-2)\cdots(k-n+1)$. □

The fact that the number of k-colorings is a polynomial in k follows from a recurrence relation called the deletion–contraction recurrence or the fundamental reduction theorem.

**Theorem 8.10.** *Let $G$ be a simple graph, for a pair of vertices $u$ and $v$ in the graph, $G/uv$ denote the graph obtained by fusing $u$ and $v$ (see §1.6). If $u$ and $v$ are not adjacent in $G$ and $G+uv$ is the graph with the edge $uv$ added. Then the numbers of k-colorings of these graphs satisfy:*

$$P_G(k) = P_{G+uv}(k) + P_{G/uv}(k)$$

By systematically applying Theorem 8.10 to pairs of nonadjacent vertices in a graph $G$, we eventually arrive at a collection of complete graphs. Thus, by Proposition 8.3, the chromatic polynomial is sum of polynomials, therefore itself is a polynomial.
We now illustrate this. Suppose that we wish to compute the chromatic polynomial of the graph $G$ of Figure 8.2. For the nonadjacent vertices $u$ and $v$ of G and the graph $H = G/uv$ obtained by identifying $u$ and $v$, it follows that by Theorem 8.10 that the chromatic polynomial of G is the sum of the chromatic polynomials of $G + uv$ and $H$.

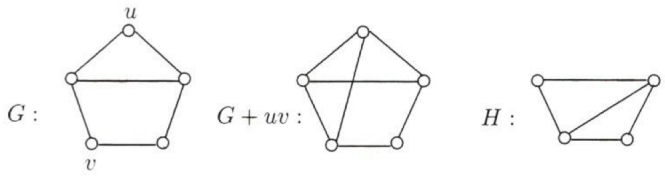
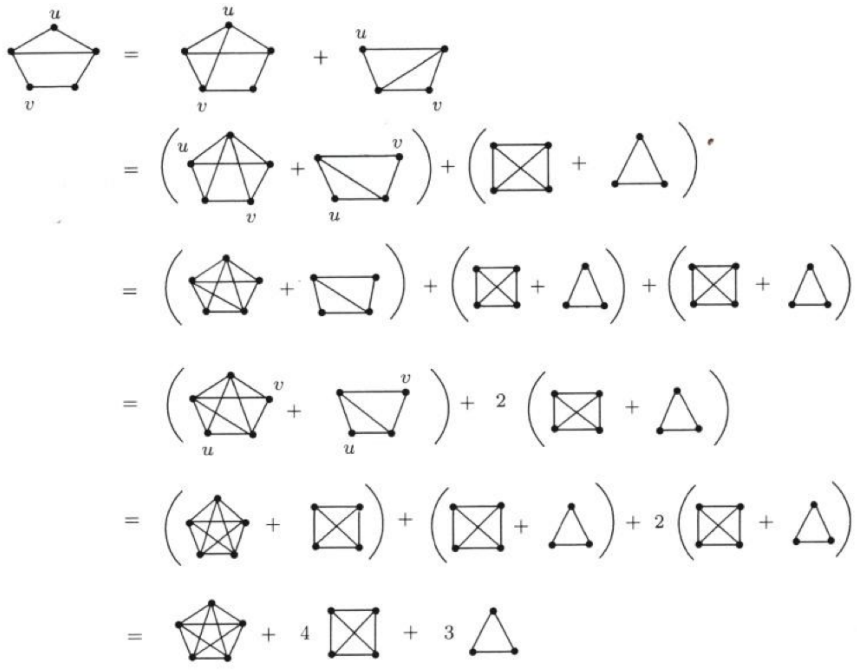



**Fig. 8.2**

To compute the chromatic polynomial of $G$, we apply this procedure recursively below.



Thus, we obtain

$$P_G(k) = k(k-1)(k-2)\cdots(k-5+1) \quad + \quad 4k(k-1)(k-2)(k-4+1) \quad + \quad 3k(k-1)(k-3+1)$$

$$= k^5 - 6k^4 + 14k^3 - 15k^2 + 6k$$

8.4. **Problem set.** 1. Find the chromatic number (both vertex and edge) for each of the graphs given above Q1. in problem set 7.2. Exhibit a coloring of the vertices of the graph that verifies your answer.
2. Find the chromatic polynomial of a tree with $n$ vertices.
3. Find the chromatic polynomial of $C_n$ and $K_{m,n}$.

## 9. Applications of graph theory

9.1. **Traveling Salesman Problem (TSP).** The Traveling Salesman Problem (TSP) is a famous problem in graph theory. It involves finding the shortest route for a "salesman" to visit a set of cities and return to the starting point. Despite its simple definition, the TSP is challenging and has many practical applications in logistics, transportation, and planning. The problem can formally be stated as follows:
Given a list of cities and the distances between each pair of cities, find the shortest possible route that: 1. Visits each city exactly once. 2. Returns to the starting city.

**Graphical Representation:** Cities are represented as vertices, the distances between cities are represented as edge weights. For Example, Consider four cities: $A, B, C$, and $D$. The distances between the cities are as follows:

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $A$ | 0 | 10 | 15 | 20 |
| $B$ | 10 | 0 | 35 | 25 |
| $C$ | 15 | 35 | 0 | 30 |
| $D$ | 20 | 25 | 30 | 0 |

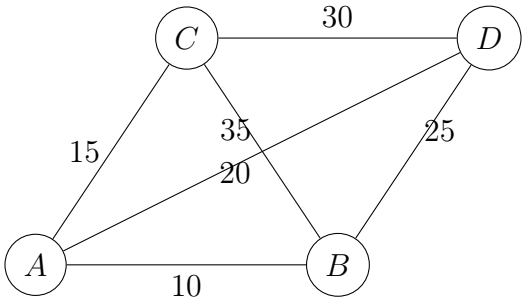
**Approaches to Solve TSP**

**1. Brute Force Method** - List all possible routes. - Calculate the total distance for each route. - Select the route with the minimum distance. But, this approach becomes infeasible for a large number of cities ($n!$ permutations).

**2. Hamiltonian Graph:** Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle

**2. Approximation Algorithms** When exact solutions are computationally expensive, approximation algorithms provide near-optimal solutions.
1) Nearest Neighbor Algorithm: Start at a city, repeatedly visit the nearest unvisited city until all cities are visited.
2) Minimum Spanning Tree (MST) Heuristic: Use MSTs to approximate a TSP solution.

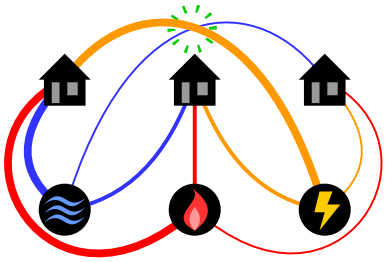
**Illustration with a Diagram**




The above diagram shows the cities as nodes and distances as edge weights.

9.2. **Utility Problems.** The utility problem, also known as the *Three Utilities Problem* or the *Three Houses Problem*, is a classic problem in graph theory. It involves connecting three houses to three utilities (gas, water, and electricity) in such a way that no two utilities cross. The problem can be phased as follows: given a configuration of three houses and three utilities, the goal is to connect each house to each utility without any of the connections

crossing. This can be represented as a bipartite graph, where the houses and utilities are the two sets of vertices, and the edges represent the connections between them.



**Impossibility of a solution:** This puzzle can be formalized as a problem in graph theory by asking whether the complete bipartite graph $K_{3,3}$, with vertices representing the houses and utilities and edges representing their connections, has a graph embedding in the plane. The impossibility of the puzzle corresponds to the fact that $K_{3,3}$ is not a planar graph, see §7.

The utility problem has been generalized to larger numbers of houses and utilities, as well as to different spatial configurations. However, the fundamental conclusion remains the same: it is impossible to solve the utility problem without at least two connections crossing, except in certain special cases.
The utility problem is a classic example in graph theory that illustrates the limitations of certain types of planar graph configurations. It has been widely studied and has applications in various fields, including urban planning and network design.

9.3. **Königsberg Bridge problem.** The Königsberg bridge problem is a classic problem in graph theory that was first posed by the Prussian mathematician Leonhard Euler in the 18th century. It is considered one of the earliest examples of the application of graph theory to a real-world problem. The problem was originally conceived as follows. The city of Königsberg (now Kaliningrad, Russia) was built on both sides of the Pregel River, with two large islands connected to each other and the two mainland portions by a series of seven bridges. The problem was to determine whether it was possible to take a walk through the city and cross each of the seven bridges exactly once.

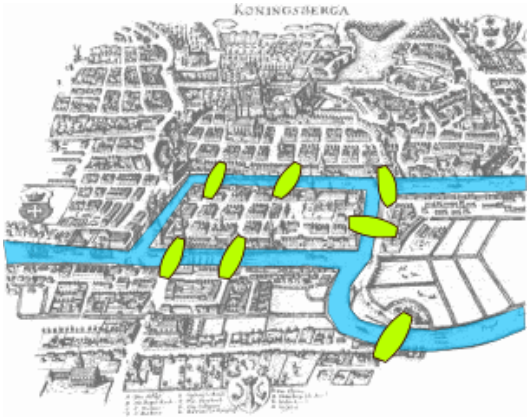


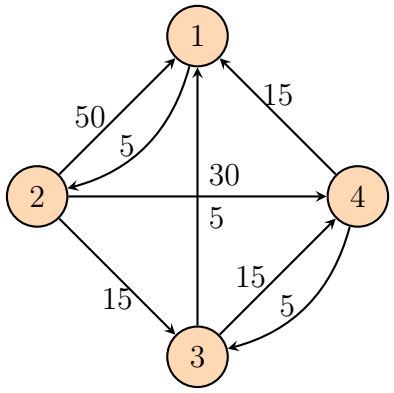

**Fig. 9.1.** Map of Königsberg in Euler's time(from wiki)

**Euler's Solution.** Euler approached the problem by representing the city as a graph, with the four land masses as vertices and the bridges as edges. He then proved that it is

impossible to find a path that crosses each bridge exactly once, unless the graph satisfies certain conditions, that is, the graph must be an Euler graph.. In the case of Königsberg, all four vertices had an odd number of edges, making it impossible to find the desired walk.

The Königsberg bridge problem is significant because it laid the foundations for the field of graph theory and the study of Eulerian paths and circuits. It also has important applications in various fields, such as network design, transportation, and computer science. The problem has been generalized to different types of graphs and has led to the development of various theorems and algorithms in graph theory, such as Euler's theorem and Fleury's algorithm.

## 10. Miscelenious




The Floyd-Warshall algorithm computes the shortest paths between all pairs of vertices in a graph. The algorithm updates the distance matrix iteratively by considering each vertex as an intermediate vertex.

**Method** 1. **Initialization:** Start with the distance matrix $D_0$, where:

$$D_0[i][j] = \begin{cases} \text{edge weight between } i \text{ and } j, & \text{if there is an edge from i to j,} \\ \infty, & \text{if there is no edge,} \\ 0, & \text{if } i = j. \end{cases}$$

2. **Update Rule:** For each intermediate vertex $k$, update the distance matrix $D_k$ as follows:

$$D_k[i][j] = \min(D_{k-1}[i][j], D_{k-1}[i][k] + D_{k-1}[k][j]).$$

3. Repeat this process for $k = 1, 2, \ldots, n$, where $n$ is the number of vertices.

**Initial Matrix ($D_0$):**

$$D_0 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

**Iteration 1 ($k = 1$):** Using vertex 1 as the intermediate vertex:

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

**Iteration 2 ($k = 2$):** Using vertex 2 as the intermediate vertex:

$$D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

**Iteration 3 ($k = 3$):** Using vertex 3 as the intermediate vertex:

$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

**Iteration 4 ($k = 4$):** Using vertex 4 as the intermediate vertex:

$$D_4 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 20 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

**Final Result:** The shortest distance matrix is:

$$D_4 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

## References

[ND] Narshingh Deo. Graph Theory with Applications to Engineering and Computer Science, 1974.
[RD] Reinhard Diestel, Graph Theory, Springer, Graduate text book.