GRAPH THEORY

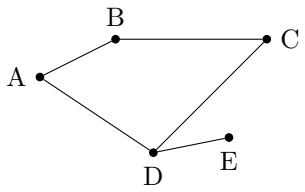
Graphs are used to show how things are connected. They can be used to help solve problems in train scheduling, traffic flow, bed usage in hospitals, and project management.
The theory of graphs was developed centuries ago, but the application of the theoretical ideas is relatively recent. Real progress was made during and after the Second World War as mathematicians, industrial technicians, and members of the armed services worked together to improve military operations.

A DEFINITIONS

Definition Graph

A **graph** (or network) is a structure which shows the physical connections or relationships between things of interest.
The things of interest are represented by **vertices** (singular: vertex), also called nodes or points.
The connections or relationships are represented by **edges**, also called lines or arcs.

Ex:

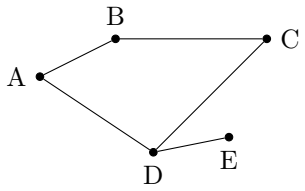


This graph shows road connections between several towns. We can see that towns A and B are directly connected, but towns A and C are not.

Definition Degree of a Vertex

The **degree** of a vertex, denoted $\deg(v)$, is the number of edges connected to it.
If a loop exists (an edge connecting a vertex to itself), it contributes 2 to the degree of that vertex.

Ex:



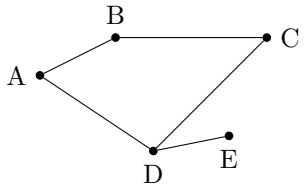
For vertex A: It has one connection to B and one connection to D. $\deg(A) = 2$.

Definition Undirected Graph

In an **undirected graph**, movement is allowed in either direction along the edges. The relationship is symmetric.

- **Adjacent vertices** are vertices connected by an edge.
- **Adjacent edges** are edges which share a common vertex.

Ex:



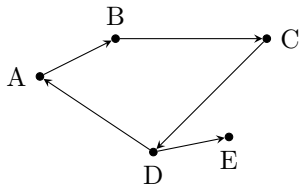
In this graph, A is connected to B and B is connected to A.

Definition Directed Graph

A **directed graph** (or digraph) contains arrows indicating the direction we can move along the edges.

- The **in-degree** is the number of edges coming into the vertex.
- The **out-degree** is the number of edges going out from the vertex.

Ex:

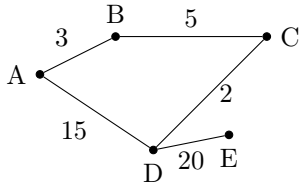


In this graph, A is connected to B and B is **not** connected to A.

Definition Weighted Graph

A **weighted graph** has numbers assigned to its edges. These numbers may represent the cost, time, or distance.

Ex:

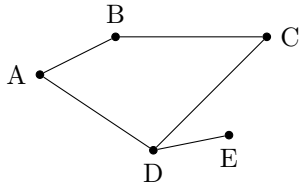


The weight of edge AD is 15.

Definition Walk, Trail, Path, circuit and Cycle

- A **walk** is a sequence of vertices such that each successive pair is adjacent.
- A **trail** is a walk in which no edge is repeated.
- A **path** is a walk in which no vertex is repeated.
- A **circuit** is a walk in which no edge is repeated that starts and ends at the same vertex.
- A **cycle** is a walk in which no vertex is repeated except the first/last vertex that starts and ends at the same vertex.

Ex:



In this graph:

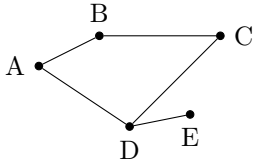
- $A \to B \to C$ is a **walk** because there is an edge between A and B and an edge between B and C.
- $A \to B \to A$ is not a **trail** because the edge between A and B is repeated.
- $A \to B \to C \to D \to A$ is not a **path** because the vertex A is repeated.
- $A \to B \to C \to D \to A$ is a **circuit** (and a cycle).

Definition Subgraph

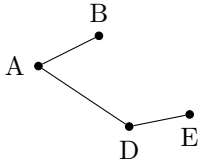
A **subgraph** is a graph that is made from a subset of the vertices and edges of another graph.

Ex: Consider the original graph G. We can create a **subgraph** H by selecting only a subset of the vertices (for example, removing vertex C) and only a subset of the edges.

Graph G



Subgraph H



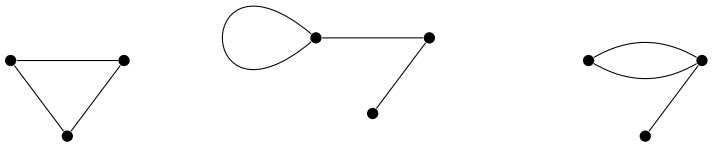
B PROPERTIES OF GRAPHS

Definition Simple Graph

A graph is called **simple** if it contains no loops and if there is a maximum of one edge joining any pair of distinct vertices.

Ex:

Simple Graph **Not Simple (loop)** **Not Simple (double edges)**



Definition Connected Graph

A graph is called **connected** if it is possible to travel from any vertex to any other vertex via a sequence of edges.

Ex:

Connected **Disconnected**

In the graph on the right, you cannot get from the left pair of vertices to the right pair. It has two disconnected components.

Definition Complete Graph

A **complete graph** is a simple undirected graph in which every vertex is connected by an edge to every other vertex. The notation K_n is used to describe the complete graph with n vertices.

Ex:

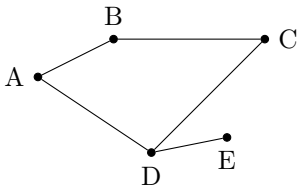
This is K_4. It has 4 vertices and $\frac{4(3)}{2} = 6$ edges.

C ADJACENCY MATRICES

Definition Adjacency Matrix

For a graph with n vertices, the **adjacency matrix** $\mathbf{M}$ is an $n \times n$ matrix where the element m_{ij} represents the number of direct edges from vertex i to vertex j.

Ex:



The table below shows the number of 1-step walks between the vertices:

Starting vertex \ *Finishing vertex*	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	0	0
C	0	1	0	1	0
D	1	0	1	0	1
E	0	0	0	1	0

The adjacency matrix for the graph is therefore:

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Since we do not usually write the labels for the vertices on the adjacency matrix, it is important to remember what the rows and columns refer to. To make this easier, we put them in alphabetical order.

Proposition Multi-step Walks

If $\mathbf{M}$ is the adjacency matrix, then the element (i, j) of $\mathbf{M}^k$ gives the number of walks of length k from vertex i to vertex j.

Ex: To understand *why* the powers of $\mathbf{M}$ give the multi-step adjacency matrices, consider the 2-step matrix:

$$\mathbf{M}^2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 2 & 0 & 1 \\ 0 & 2 & 0 & 2 & 0 \\ 2 & 0 & 2 & 0 & 1 \\ 0 & 2 & 0 & 3 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The element in row 2 column 4 of $\mathbf{M}^2$ shows there are **two** 2-step walks from B to D.
This value comes from the multiplication **row 2** $\times$ **column 4** as follows:

- $1 \times 1 = 1$ walk from B to A $\times$ 1 walk from A to D
- $0 \times 0 = 0$ walks from B to B $\times$ 0 walks from B to D
- $1 \times 1 = 1$ walk from B to C $\times$ 1 walk from C to D
- $0 \times 0 = 0$ walks from B to D $\times$ 0 walks from D to D
- $0 \times 1 = 0$ walks from B to E $\times$ 1 walk from E to D

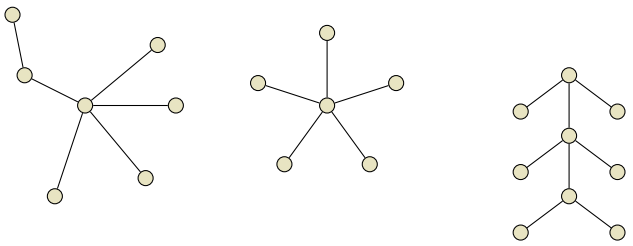
Adding these terms gives the two 2-step walks from B to D. The walks are B $\to$ A $\to$ D and B $\to$ C $\to$ D.

D TREES AND MINIMUM SPANNING TREES

Definition Tree

A **tree** is a connected graph with no cycles.

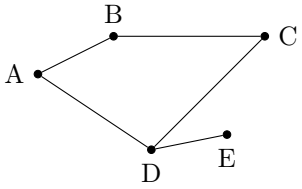
Ex: Some examples of trees are shown below:



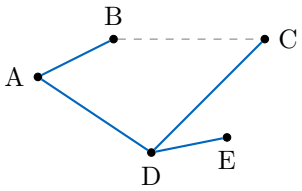
Definition Spanning Tree

A **spanning tree** of a connected graph is a subgraph that contains **all** the vertices of the original graph and is a tree (connected and with no cycles).

Ex: Consider the following graph G:



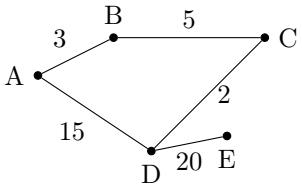
This graph has a cycle ($A - B - C - D - A$). To form a spanning tree, we must remove an edge to break the cycle while keeping all vertices connected.
For instance, if we remove the edge connecting B and C, we get a spanning tree (highlighted in blue):



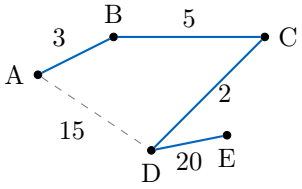
Definition Minimum Spanning Tree

For a weighted connected graph, a **minimum spanning tree** (MST) is a spanning tree where the sum of the weights of the edges is minimized. It connects all vertices with the lowest possible total cost.

Ex:



Consider the weighted graph below. The Minimum Spanning Tree is:

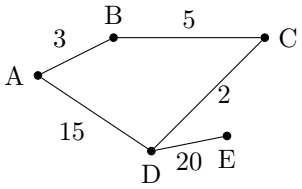


The total weight of the Minimum Spanning Tree is $20 + 2 + 3 + 5 = 30$.

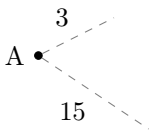
Method Prim's Algorithm for MST

1. Start with any vertex.
2. Choose the edge with the lowest weight connecting a vertex in the tree to a vertex outside.
3. Add the chosen edge and the new vertex. Repeat until all vertices connected.

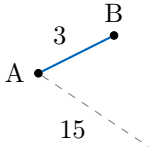
Ex: Let's apply Prim's algorithm to this graph.



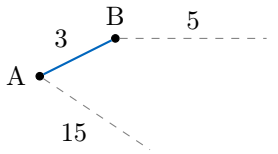
1. **Start at A**. The edges connected to A are AB (3) and AD (15).



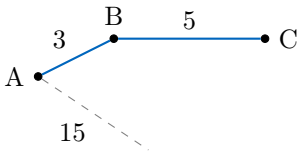
2. **Choose AB** (weight 3) as it is the lowest. Our tree is $\{A, B\}$.



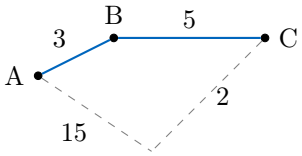
3. Look at the edges connecting $\{A, B\}$ to the outside: AD (15) and BC (5).



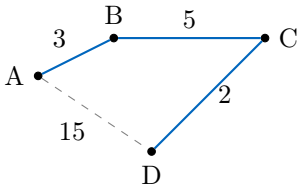
4. **Choose BC** (weight 5). Our tree is $\{A, B, C\}$.



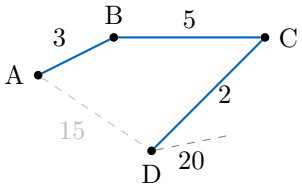
5. Look at the edges connecting $\{A, B, C\}$ to the outside: AD (15) and CD (2).



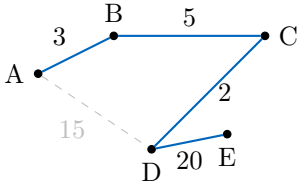
6. **Choose CD** (weight 2). Our tree is $\{A, B, C, D\}$.



7. Look at the edges connecting $\{A, B, C, D\}$ to the outside: DE (20). Note that AD is now an internal edge (both endpoints are in the tree), so we ignore it.



8. **Choose DE** (weight 20).

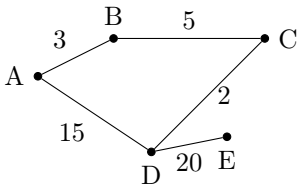


All vertices are connected. Total weight: $3 + 5 + 2 + 20 = 30$.

Method **Kruskal's Algorithm for MST**

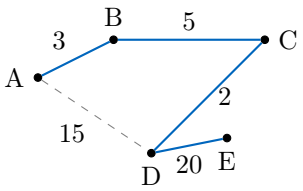
1. List (or sort) all edges in **non-decreasing** order of weight.
2. Start with an empty set of selected edges (a forest of isolated vertices).
3. Consider edges in that order. For each edge, **add it** if it connects two **different** components (i.e. it does **not** create a cycle); otherwise **skip** it.
4. Stop when you have selected $n - 1$ edges (equivalently, when all n vertices are connected).

Ex: Let's apply Kruskal's algorithm to the same graph.



1. **List all edges by weight (ascending):**
   - CD (2)
   - AB (3)
   - BC (5)
   - AD (15)
   - DE (20)
2. **Select edges in order:**
   - **Select CD** (weight 2). No cycle.
   - **Select AB** (weight 3). No cycle.
   - **Select BC** (weight 5). No cycle.
   - **Consider AD** (weight 15). Adding this edge would form a cycle ($A - B - C - D - A$). **Reject**.
   - **Select DE** (weight 20). No cycle.

The algorithm stops when $n - 1$ edges are selected (here, 4 edges).



Total weight: $2 + 3 + 5 + 20 = 30$.

E EULERIAN GRAPHS

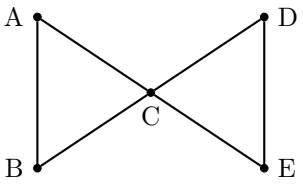
Definition Eulerian Graphs

- An **Eulerian circuit** is a trail which traverses every edge of the graph exactly once and starts and ends at the same vertex.
- An **Eulerian trail** traverses every edge exactly once but starts and ends at different vertices.

Proposition Conditions for Eulerian Graphs

- A connected graph has an **Eulerian circuit** if and only if every vertex has an **even degree**.
- A connected graph has an **Eulerian trail** if and only if it has **exactly two vertices of odd degree**.

Ex: Consider the following undirected graph (a "bowtie" shape).



Let's calculate the degree of each vertex (the number of edges connected to it):

- $\deg(A) = 2$
- $\deg(B) = 2$
- $\deg(C) = 4$ (connected to A, B, D, E)
- $\deg(D) = 2$

- $\deg(E) = 2$

Since **all vertices have an even degree**, the graph contains an **Eulerian circuit**.
One possible circuit is: $C \to A \to B \to C \to D \to E \to C$.

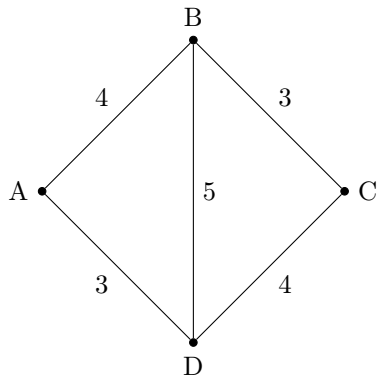
The **Chinese Postman Problem** was posed by the mathematician Kwan Mei-Ko. The problem asks: how can a postman deliver mail along every road in a network and return to the depot traveling the minimum possible distance? Mathematically, this means traversing every edge of a weighted graph at least once and returning to the start.

Method The Chinese Postman Problem

The goal is to find the shortest closed walk that traverses every edge at least once.
If the graph is not Eulerian, we must repeat edges to make all vertices even-degree.

1. Identify all vertices with odd degrees.
2. Find all possible pairings of these odd vertices.
3. For each pair, find the shortest path between them.
4. Select the pairing that adds the minimum total weight.
5. Add this weight to the total weight of the graph.

Ex: Solve the Chinese Postman Problem for the following weighted graph.



1. **Total weight of the graph:** $4 + 3 + 4 + 3 + 5 = 19$.
2. **Identify odd degrees:**
   - $\deg(A) = 2$, $\deg(C) = 2$ (Even).
   - $\deg(B) = 3$, $\deg(D) = 3$ (Odd).

   The odd vertices are B and D.
3. **Find shortest path between odd vertices:** The odd vertices must be paired. Since there are only two, the pair is (B, D).
   Paths from B to D:
   - Direct: B $\to$ D (weight 5).
   - Via A: B $\to$ A $\to$ D (weight $4 + 3 = 7$).
   - Via C: B $\to$ C $\to$ D (weight $3 + 4 = 7$).

   The shortest path is the direct edge BD with weight 5.
4. **Calculate optimal distance:** Total = Graph Weight + Shortest Path Addition Total = $19 + 5 = 24$.

To minimize distance, the postman must traverse the edge BD twice.
One possible optimal walk is:

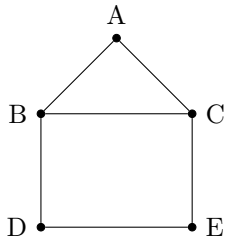
$$A \to B \to D \to B \to C \to D \to A$$

F HAMILTONIAN GRAPHS

Definition Hamiltonian Cycle and Path

- A **Hamiltonian cycle** is a cycle that visits every vertex of the graph exactly once.
- A **Hamiltonian path** visits every vertex exactly once but does not necessarily return to the start.

Ex: Consider the following graph.



- A **Hamiltonian Path**: $D \to B \to A \to C \to E$. (It visits every vertex exactly once, but does not return to D).
- A **Hamiltonian Cycle**: $A \to B \to D \to E \to C \to A$. (It visits every vertex exactly once and returns to the start).

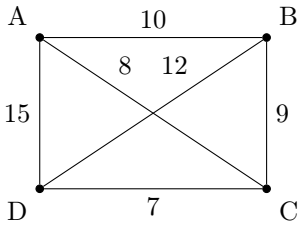
The **Travelling Salesman Problem (TSP)** describes a classic optimization scenario: *A salesman must leave his home town, visit a specific set of other towns exactly once, and then return home. The objective is to find the walk that minimizes the total distance traveled.*
In Graph Theory terms, this is walk to finding the **Hamiltonian Cycle** with the minimum total weight in a complete weighted graph.

Method The Travelling Salesman Problem (TSP)

The goal is to find the Hamiltonian cycle with the minimum total weight in a complete weighted graph.
Since finding the exact solution is hard, we find bounds:

- **Upper Bound (Nearest Neighbour Algorithm):** Start at a vertex, always move to the nearest unvisited vertex, and return to the start at the end.
- **Lower Bound (Deleted Vertex Algorithm):**
  1. Delete a vertex V and its edges.
  2. Find the MST of the remaining graph.
  3. Add the two shortest edges from V to the MST weight.

Ex: Find the bounds for the optimal tour starting at A for the following graph.



1. **Upper Bound (Nearest Neighbour from A):**
   - Start at A. Nearest unvisited neighbors: B(10), C(8), D(15). Choose **C**.
   - From C. Unvisited: B(9), D(7). Choose **D**.
   - From D. Unvisited: B(12). Choose **B**.
   - From B, return to start (A). Weight 10.
   - Walk: $A \to C \to D \to B \to A$.
   - Total Weight: $8 + 7 + 12 + 10 = 37$.
2. **Lower Bound (Delete Vertex A):**
   - Remove A. Remaining vertices: B, C, D. Edges: BC(9), CD(7), BD(12).
   - MST of remainder: Select CD(7), then BC(9). Total = 16.

- Edges from A: AB(10), AC(8), AD(15). Two shortest are **8** and **10**.
- Total Weight: $16 + 8 + 10 = 34$.

The optimal solution is between 34 and 37.