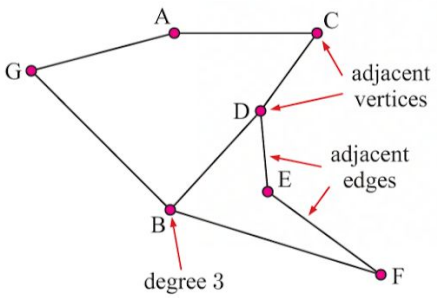
Graph Theory Summary

A **graph** or **network** is a structure consisting of **vertices** and **edges**, which shows the physical connections or relationships between things of interest.

If we are allowed to move in either direction along the edges, the graph is **undirected**.



In an undirected graph:

- **Adjacent vertices** are vertices which are connected by an edge.
- **Adjacent edges** are edges which share a common vertex.
- The **degree** of a vertex is the number of edges connected to it.

A **directed graph** contains arrows indicating the direction we can move along the edges.

In a directed graph:

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

Properties of graphs

- 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.
- An undirected graph is **connected** if it is possible to travel from every vertex to every other vertex by following edges.
- A directed graph is **strongly connected** if it is possible to travel from every vertex to every other vertex by following edges in the correct direction.
- A **complete graph** is a simple undirected graph in which every vertex is connected by an edge to every other vertex.

Routes on graphs

- A **walk** is a sequence of vertices such that each successive pair of vertices is adjacent.
- A **trail** is a walk in which no *edge* is repeated.
- A **path** is a walk in which no *edge* or *vertex* is repeated.
- A **circuit** is a trail which starts and finishes at the same vertex.
- A **cycle** is a circuit in which no vertex is repeated (except the finish coinciding with the start).

The **length** of a route is the number of edges traversed.

Adjacency matrices

For a given graph:

- The **adjacency matrix A** shows the number of direct routes between each pair of vertices.
- $\mathbf{A}^n$ shows the number of routes of length n between each pair of vertices.

Minimum spanning trees

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

A **spanning tree** of a graph is a tree which contains all the vertices of the graph.

On a weighted graph, the spanning tree with minimum weight is called the **minimum spanning tree**.

We can use **Kruskal's algorithm** or **Prim's algorithm** to find the minimum spanning tree.

Kruskal's algorithm

For a graph with n vertices:

Step 1: Start with the shortest (or **least weight**) edge. If there are several, choose one at random.

Step 2: Choose the shortest edge remaining that does not complete a cycle. If there is more than one possible choice, pick one at random.

Step 3: Repeat *Step 2* until $n - 1$ edges have been chosen.

Prim's algorithm

Step 1: Start with any vertex.

Step 2: Join this vertex to the nearest vertex. If two or more vertices are an equal distance away, choose one at random.

Step 3: Join the vertex which is nearest to *either* of those already connected.

Step 4: Continue joining connected vertices to the nearest unconnected vertex until all vertices are connected.

You should also be able to apply Prim's algorithm to a weighted adjacency table.

Eulerian graphs

An **Eulerian circuit** is a circuit which traverses every edge exactly once.

An **Eulerian trail** is a trail which traverses every edge exactly once, but does not start and end at the same vertex.

A graph is:

- **Eulerian** if it contains an Eulerian circuit
- **semi-Eulerian** if it contains an Eulerian trail but not an Eulerian circuit.

We can use the degrees of a graph's vertices to identify Eulerian and semi-Eulerian graphs.

- A connected graph is **Eulerian** if there are *no* vertices of odd degree.
- A connected graph is **semi-Eulerian** if there are *exactly two* vertices of odd degree. The Eulerian trail starts at one of these vertices and ends at the other.

Chinese Postman Problem

The **Chinese Postman Problem** is to find the route of minimum weight which starts and finishes at the same vertex, and traverses every edge of the graph.

If the graph is Eulerian, any Eulerian circuit is a solution to the Chinese Postman Problem.

If a graph is not Eulerian, some of the edges must be traversed twice to solve the Chinese Postman Problem. We need to traverse twice the edges forming the shortest paths between pairs of odd vertices.

Hamiltonian graphs

A **Hamiltonian cycle** is a cycle which visits each vertex (except the starting and ending vertex) exactly once.

A **Hamiltonian path** is a path which visits each vertex exactly once, but does not start and end at the same vertex.

A graph is:

- **Hamiltonian** if it contains a Hamiltonian cycle
- **semi-Hamiltonian** if it contains a Hamiltonian path, but not a Hamiltonian cycle.

Travelling Salesman Problem

The **Travelling Salesman Problem** (TSP) is to find the route of minimum weight which starts and finishes at the same vertex, and visits every vertex of the graph.

The **nearest neighbour algorithm** can be used to find an *upper bound* for the TSP.

Step 1: Choose a starting vertex.

Step 2: Follow the edge of least weight to an unvisited vertex. If there is more than one such edge, choose one at random.

Step 3: Repeat *Step 2* until all vertices have been visited.

Step 4: Return to the starting vertex by adding the corresponding edge.

The **deleted vertex algorithm** can be used to find a *lower bound* for the TSP.

Step 1: Delete a vertex, together with all edges connected to it, from the original graph.

Step 2: Find the minimum spanning tree for the remaining graph.

Step 3: Add to the length of the minimum spanning tree, the lengths of the two shortest deleted edges.

You should understand the difference between the **classical TSP** and the **practical TSP**. A practical TSP can be converted into a classical TSP by adding or replacing edges to show the least weight between vertices.