DISCRETE MATHEMATICS

Chapter 7: Graph Theory

Engineering Study Notes for Diploma / Polytechnic Students

Topics Covered

7.1 Introduction to Graphs

A graph is one of the most fundamental structures studied in discrete mathematics and forms the theoretical backbone of computer networking, data structures, circuit design, and operations research. Formally, a graph G is defined as an ordered pair G = (V, E), where V is a finite non-empty set of elements called vertices (or nodes), and E is a set of unordered or ordered pairs of vertices called edges. An edge connects two vertices and represents a relationship between them.

Example: Let V = {A, B, C, D} and E = {(A,B), (B,C), (C,D), (D,A)}. This describes a graph with 4 vertices and 4 edges forming a cycle A–B–C–D–A.

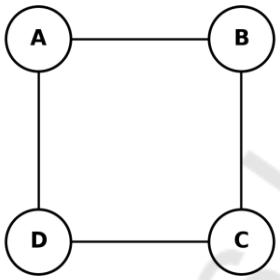

Fig 7.1(a): Undirected graph on 4 vertices

Directed and Undirected Graphs

Undirected Graph: A graph in which edges have no direction — an edge (u, v) is identical to (v, u), simply indicating that u and v are connected. Undirected graphs are used to model symmetric relationships, such as friendship networks or bidirectional roads.

Example: A road map where every road can be travelled in both directions is an undirected graph.

Directed Graph (Digraph): A graph in which every edge has a direction, represented as an ordered pair (u, v), meaning the edge goes from u to v but not necessarily from v to u. Directed graphs model asymmetric relationships such as one-way streets, task dependencies, or the flow of information.

Example: A Twitter 'follows' network is directed: if A follows B, it does not imply B follows A.

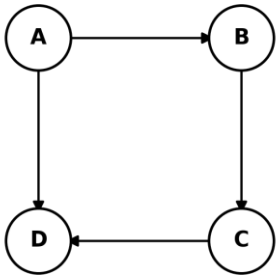

Fig 7.1(b): Directed graph (digraph)

Basic Terminology

- **Loop (Self-loop):** An edge that connects a vertex to itself, i.e., an edge of the form (v, v). A vertex with a loop is said to have a self-loop.
- **Multigraph:** A graph in which two or more edges connect the same pair of vertices (parallel edges) are allowed, but self-loops are not permitted.
- **Pseudograph:** A graph that permits both parallel (multiple) edges between the same pair of vertices as well as self-loops. A pseudograph is the most general form of graph.
- **Simple Graph:** A graph that has neither self-loops nor parallel/multiple edges. Between any pair of vertices there is at most one edge. Most introductory graph theory problems deal with simple graphs.
- **Finite Graph:** A graph in which both the vertex set V and the edge set E contain a finite number of elements.
- **Infinite Graph:** A graph in which the vertex set or the edge set (or both) is infinite. Infinite graphs are mainly of theoretical interest and are rarely used in applied engineering problems.

Example: A network diagram with vertices {P, Q, R} and edges {(P,Q), (P,Q), (Q,R), (R,R)} is a pseudograph — it has a parallel edge between P and Q and a self-loop at R.

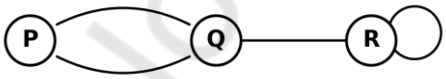

Fig 7.1(c): Pseudograph — parallel edge P–Q and self-loop at R

Other frequently used terms include the degree of a vertex (the number of edges incident on it; for a directed graph, in-degree and out-degree are counted separately), adjacent vertices (two vertices joined directly by an edge), and adjacent edges (two edges that share a common vertex). The order of a graph refers to the number of vertices $|V|$, and the size of a graph refers to the number of edges $|E|$.

7.2 Subgraphs

Given a graph G = (V, E), a subgraph is a graph G′ = (V′, E′) such that V′ ⊆V and E′ ⊆E, where every edge in E′ has both its end vertices in V′. In simple terms, a subgraph is formed by taking a subset of the vertices and a subset of the edges of the original graph, provided the chosen edges only connect the chosen vertices.

Spanning Subgraph

A spanning subgraph of G is a subgraph G′ = (V′, E′) that includes every vertex of the original graph, i.e., V′ = V, but may contain only a subset of the edges, E′ ⊆E. A spanning subgraph therefore 'spans' all the vertices

while possibly omitting some connections. A spanning tree is a well-known special case: a spanning subgraph that is also a tree (connected and acyclic).

Removal of a Vertex and an Edge

Vertex Removal (G − v): Removing a vertex v from graph G deletes v itself along with every edge incident on v. The resulting graph has one fewer vertex, and the edge set shrinks by the degree of v.

Edge Removal (G − e): Removing an edge e from graph G deletes only that edge; both end vertices of e remain in the graph. The vertex set is unchanged, and the edge set is reduced by exactly one edge.

Induced Subgraph

Given a subset of vertices $S \subseteq V$, the induced subgraph G[S] is the subgraph formed by taking all vertices in S and including every edge of the original graph G whose both endpoints lie in S. Unlike an arbitrary subgraph, an induced subgraph is completely determined by the choice of vertex subset — no edges may be selectively omitted.

> *Example: Let G have V = {1,2,3,4} and E = {(1,2),(2,3),(3,4),(1,4),(1,3)}. For S = {1,2,3}, the induced subgraph G[S] contains vertices {1,2,3} and all edges of G joining these vertices: {(1,2),(2,3),(1,3)}.*

7.3 Graph Isomorphism

Two graphs G1 = (V1, E1) and G2 = (V2, E2) are said to be isomorphic if there exists a bijective (one-to-one and onto) function f: V1 → V2 such that any two vertices u and v are adjacent in G1 if and only if f(u) and f(v) are adjacent in G2. In other words, isomorphic graphs have exactly the same structure — the same number of vertices, the same number of edges, and the same connectivity pattern — even though the vertices may be drawn or labelled differently.

Necessary Conditions for Isomorphism

- Both graphs must have the same number of vertices.
- Both graphs must have the same number of edges.
- Both graphs must have the same degree sequence (the list of vertex degrees, sorted in the same order).
- The number of vertices with a particular degree must match between the two graphs.
- Corresponding subgraphs (such as cycles of a given length) must be present in equal numbers in both graphs.

These conditions are necessary but not sufficient — two graphs may satisfy all of them and still not be isomorphic, so a candidate vertex correspondence must ultimately be verified directly against the edge sets.

> *Example: G1 has V1 = {a,b,c,d} and E1 = {(a,b),(b,c),(c,d),(d,a)} (a 4-cycle). G2 has V2 = {1,2,3,4} and E2 = {(1,2),(2,3),(3,4),(4,1)} (also a 4-cycle). The mapping f(a)=1, f(b)=2, f(c)=3, f(d)=4 preserves adjacency in both directions, so G1 and G2 are isomorphic.*

7.4 Walks, Paths, Length and Circuits

Walk: A walk in a graph is a finite alternating sequence of vertices and edges, v0, e1, v1, e2, v2, …, en, vn, beginning and ending with a vertex, such that each edge ei connects vi−1 and vi. Vertices and edges may repeat freely in a walk.

Length of a Walk: The length of a walk is the number of edges traversed in it (equivalently, the number of edges in the sequence, counted with repetition).

Open Walk: A walk in which the starting vertex and the ending vertex are different (v0 ≠ vn).

Closed Walk: A walk in which the starting vertex and the ending vertex are the same (v0 = vn).

Path: A path is a walk in which no vertex is repeated (equivalently, all vertices — and consequently all edges — are distinct). A path is always an open walk unless it consists of a single vertex.

Circuit (Cycle): A circuit is a closed walk in which no vertex, except the starting and ending vertex (which coincide), is repeated. A circuit therefore returns to its starting point without revisiting any intermediate vertex. Circuits are also referred to as cycles when the length is at least 3 in a simple graph.

> *Example: In a graph with vertices {A,B,C,D} and edges {(A,B),(B,C),(C,D),(D,A),(A,C)}: the sequence A–B–C–D–A is a circuit (closed, no repeated intermediate vertex); the sequence A–B–C–A–D is a walk but not a path (vertex A repeats); the sequence A–B–C–D is a path of length 3.*

Term	Vertices Repeated?	Open / Closed
Walk	May repeat	Either
Path	Never repeat	Open (start ≠ end)
Circuit / Cycle	Only start = end vertex coincide	Closed (start = end)

7.5 Euler Graphs

The study of Euler graphs originates from the famous Seven Bridges of Königsberg problem solved by Leonhard Euler in 1736, widely regarded as the founding problem of graph theory. Euler graphs deal with the question of whether a graph can be traced by travelling along every edge exactly once.

Euler Path (Euler Trail): A walk in a connected graph that uses every edge of the graph exactly once, but does not necessarily return to the starting vertex. A connected graph has an Euler path if and only if it has exactly zero or exactly two vertices of odd degree; if it has two odd-degree vertices, every Euler path must start at one of them and end at the other.

Euler Circuit (Euler Cycle): A closed walk in a connected graph that uses every edge exactly once and returns to the starting vertex. A connected graph possesses an Euler circuit if and only if every vertex of the graph has even degree. A graph that contains an Euler circuit is called an Euler graph.

Conditions Summary

- Euler Circuit exists ⇔ graph is connected and every vertex has even degree.

- Euler Path (but not circuit) exists ⇔ graph is connected and exactly two vertices have odd degree.
- If more than two vertices have odd degree, neither an Euler path nor an Euler circuit exists.

Example: The Königsberg bridge graph has four vertices, each of odd degree (3, 3, 3, 5 in different bridge arrangements) — since more than two vertices are odd, Euler proved no walk could cross every bridge exactly once and return to the start, or even complete the tour at all.

Example: A cycle graph C5 (5 vertices in a ring) has every vertex of degree 2 (even), so it is an Euler graph and possesses an Euler circuit — simply traverse the cycle once around.

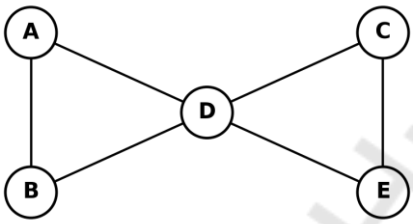

Fig 7.5: A bowtie graph — every vertex has even degree (A, B, C, E: degree 2; D: degree 4), so an Euler circuit exists, e.g. A–B–D–C–E–D–A

7.6 Hamiltonian Graphs

Hamiltonian Path: A path in a graph that visits every vertex of the graph exactly once (but need not return to the starting vertex).

Hamiltonian Circuit (Hamiltonian Cycle): A circuit that visits every vertex of the graph exactly once and returns to the starting vertex. A graph that contains a Hamiltonian circuit is called a Hamiltonian graph.

Unlike Euler graphs, there is no simple necessary-and-sufficient condition for the existence of a Hamiltonian circuit in a general graph; determining Hamiltonicity is, in fact, an NP-complete problem in computer science. However, several useful sufficient conditions exist, the most well-known being Dinitz–Ore and Dirac's theorem.

Dirac's Theorem: If G is a simple graph with $n \geq 3$ vertices and every vertex has degree $\geq n/2$, then G is guaranteed to contain a Hamiltonian circuit.

Solved Problem

Problem: Consider a complete graph K4 with vertices {A, B, C, D}, where every pair of vertices is joined by an edge. Determine a Hamiltonian circuit.

Solution: Since K4 is complete, every vertex has degree 3, which satisfies Dirac's condition ($n/2 = 2$). A Hamiltonian circuit can be traced as A–B–C–D–A, which visits all four vertices exactly once and returns to A. Hence K4 is a Hamiltonian graph.

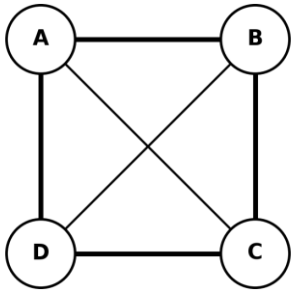

Fig 7.6: K4 — the bold edges A–B–C–D–A form a Hamiltonian circuit; the thin diagonals are the remaining edges of K4

Euler vs Hamiltonian — Key Distinction

Aspect	Euler Graph	Hamiltonian Graph
Focus	Every edge exactly once	Every vertex exactly once
Existence test	Simple degree condition	No simple general test (NP-complete)
Example	Königsberg bridges	Travelling Salesman-type routes

7.7 Sequential Representation of Graphs

Sequential representation stores a graph using arrays or matrices, giving fast, direct (constant-time) access to check whether an edge exists between any two given vertices. The two principal sequential representations are the adjacency matrix and the incidence matrix.

Adjacency Matrix

For a graph with n vertices, the adjacency matrix A is an n × n matrix where A[i][j] = 1 if there is an edge between vertex i and vertex j, and A[i][j] = 0 otherwise. For a weighted graph, A[i][j] stores the weight of the edge instead of 1. For an undirected graph the adjacency matrix is always symmetric (A[i][j] = A[j][i]); for a directed graph it need not be symmetric. The diagonal entries A[i][i] indicate self-loops.

> *Example: For vertices {1,2,3} and edges {(1,2),(2,3)}, the adjacency matrix is: Row1: [0 1 0], Row2: [1 0 1], Row3: [0 1 0]*

Incidence Matrix

For a graph with n vertices and m edges, the incidence matrix I is an n × m matrix where I[i][j] = 1 if vertex i is incident on edge j, and 0 otherwise (for directed graphs, +1 and −1 are used to indicate the tail and head of each edge respectively). The incidence matrix is useful for representing multigraphs, since parallel edges simply occupy separate columns.

Sequential representations require $O(n^2)$ space for an adjacency matrix, making them memory-efficient for dense graphs but wasteful for sparse graphs, where far fewer edges exist than the n^2 possible pairs.

7.8 Linked Representation of Graphs

Linked representation stores a graph using an adjacency list: an array (or hash map) of n lists, one per vertex, where the list for vertex v contains all vertices adjacent to v (its neighbours). Each entry may also store the weight of the corresponding edge for weighted graphs.

> *Example: For vertices {1,2,3,4} and edges {(1,2),(1,3),(2,4),(3,4)}, the adjacency list is: 1 → [2, 3] 2 → [1, 4] 3 → [1, 4] 4 → [2, 3]*

Adjacency Matrix vs Adjacency List

Criterion	Adjacency Matrix	Adjacency List
Space Complexity	$O(n^2)$	O(n + m)
Best suited for	Dense graphs	Sparse graphs
Edge existence check	O(1)	O(degree of vertex)
Iterating all neighbours of v	O(n)	O(degree of v)

In practice, most real-world networks (social networks, road maps, the internet graph) are sparse, so the adjacency list is the preferred representation in software implementations, while the adjacency matrix is favoured when frequent edge-existence queries or algebraic graph operations are required.

7.9 Traversal of Graphs

Graph traversal refers to the process of visiting every vertex of a graph in a systematic manner, typically to search for a target, check connectivity, or build auxiliary structures such as spanning trees. The two fundamental traversal strategies are Breadth-First Search (BFS) and Depth-First Search (DFS), both of which are covered with their algorithms in Section 7.8 (Shortest Path & Traversal Algorithms) below.

- Traversal ensures every reachable vertex from a given source is visited exactly once.
- A visited/unvisited marker (Boolean array) is maintained for each vertex to prevent revisiting and infinite loops (important in graphs with cycles).
- Traversal underlies many higher-level algorithms: connectivity checking, cycle detection, topological sorting, and shortest-path computation.

7.8 Shortest Path, Shortest Path Algorithms & Traversal Algorithms

The shortest path problem asks for a path between two vertices in a weighted graph such that the sum of the weights of its constituent edges is minimised. This has extensive applications in routing, navigation systems, and network design.

Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest path from a single source vertex to all other vertices in a weighted graph with non-negative edge weights. It is a greedy algorithm.

1. Assign a tentative distance value to every vertex: 0 for the source vertex and infinity for all others; mark all vertices as unvisited.
2. Select the unvisited vertex with the smallest tentative distance; call it the current vertex.
3. For the current vertex, examine all unvisited neighbours and calculate their tentative distance through the current vertex; update the neighbour's distance if this new value is smaller than its previously recorded value.
4. Mark the current vertex as visited; a visited vertex will never be checked again.
5. Repeat steps 2–4 until all vertices have been visited or the smallest tentative distance among unvisited vertices is infinity.

Time Complexity: $O(V^2)$ with a simple array implementation, or $O((V + E) \log V)$ using a binary heap / priority queue.

Floyd–Warshall Algorithm

The Floyd–Warshall algorithm computes the shortest paths between every pair of vertices in a weighted graph (all-pairs shortest path), and works correctly even when some edge weights are negative, provided there is no negative-weight cycle. It uses dynamic programming.

1. Initialise a distance matrix D where D[i][j] equals the weight of the edge (i, j) if it exists, 0 if i = j, and infinity otherwise.
2. For each vertex k from 1 to n, treat k as an intermediate vertex and update every pair (i, j): D[i][j] = min(D[i][j], D[i][k] + D[k][j]).
3. After all n iterations over k, D[i][j] holds the length of the shortest path from i to j using any of the vertices as intermediates.

Time Complexity: $O(V^3)$, which is efficient for dense graphs and small-to-medium vertex counts despite the cubic growth.

Breadth-First Search (BFS) Algorithm

BFS explores a graph level by level, visiting all neighbours of the current vertex before moving to the next level of vertices. It uses a queue (First-In-First-Out) data structure.

1. Choose a starting vertex, mark it as visited, and insert it into a queue.
2. While the queue is not empty, remove (dequeue) the front vertex and process it.
3. For every unvisited neighbour of the dequeued vertex, mark it as visited and insert (enqueue) it into the queue.
4. Repeat until the queue becomes empty.

BFS is particularly used to find the shortest path (in terms of number of edges) in an unweighted graph, and to test bipartiteness.

Depth-First Search (DFS) Algorithm

DFS explores a graph by going as deep as possible along each branch before backtracking. It uses a stack (Last-In-First-Out) structure, either explicitly or through recursion.

1. Choose a starting vertex and mark it as visited.
2. Select an unvisited neighbour of the current vertex, mark it as visited, and recursively (or via a stack) move to that vertex.
3. If the current vertex has no unvisited neighbours, backtrack to the previous vertex on the path.
4. Repeat until every vertex reachable from the source has been visited.

DFS is widely used for cycle detection, topological sorting, and finding connected components.

BFS vs DFS

Criterion	BFS	DFS
Data Structure	Queue	Stack / Recursion
Approach	Level-wise (breadth)	Branch-wise (depth)
Shortest path (unweighted)	Yes	No (not guaranteed)
Memory usage	Can be higher (stores a full level)	Generally lower
Time Complexity	O(V + E)	O(V + E)

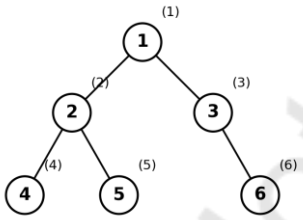

Fig 7.8(a): BFS order

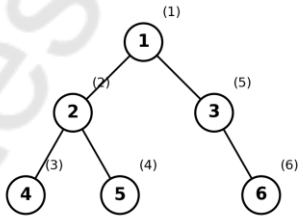

Fig 7.8(b): DFS order

On the same sample graph, BFS visits vertices level by level — (1,2,3,4,5,6) — while DFS plunges down one branch first before backtracking — (1,2,4,5,3,6). The numbers in brackets show the order in which each vertex is visited starting from vertex 1.

7.9 Applications of Graph Theory

Graph theory is not merely an abstract mathematical topic; it directly underpins the design and analysis of countless real-world systems that an engineering student will encounter across computer science, electrical, and civil engineering disciplines.

- **Computer Networks:** Routers and hosts are modelled as vertices and communication links as edges; shortest-path algorithms determine efficient data routing (e.g., OSPF routing protocol uses Dijkstra's algorithm).
- **Social Networks:** Users are vertices and friendships/follows are edges; graph analysis identifies communities, influencers, and recommends new connections.
- **Transportation and Map Navigation:** Cities/junctions are vertices and roads are weighted edges; GPS navigation systems use shortest-path algorithms to compute the fastest route.

- **Circuit Design (VLSI/Electrical Engineering):** Electronic components and their connections are modelled as graphs to detect circuit faults, plan PCB layouts, and minimise wiring.
- **Scheduling and Task Dependency:** Directed acyclic graphs (DAGs) represent tasks with dependencies; topological sorting determines a valid order of execution, used in project management (PERT/CPM) and compiler instruction scheduling.
- **Web Search and Ranking:** Web pages are vertices and hyperlinks are directed edges; Google's PageRank algorithm relies on the structure of this directed graph to rank pages.
- **Chemistry and Molecular Structures:** Atoms are represented as vertices and chemical bonds as edges to study molecular structure and isomerism.
- **Operations Research:** Graph-based models such as the Travelling Salesman Problem (related to Hamiltonian circuits) and network flow problems optimise logistics, supply chains, and resource allocation.
- **Puzzle and Game Theory:** Many classical puzzles (Königsberg bridges, the Knight's Tour, maze-solving) are naturally modelled and solved using graph concepts such as Euler and Hamiltonian paths.

Multiple Choice Questions

Answer key is provided at the end of this section.

1. A graph in which every edge has a direction associated with it is called a:

a) Directed graph
b) Undirected graph
c) Multigraph
d) Simple graph

2. An edge that connects a vertex to itself is called a:

a) Pendant edge
b) Bridge
c) Loop
d) Multi-edge

3. A graph that allows both parallel edges and self-loops is called a:

a) Complete graph
b) Simple graph
c) Multigraph
d) Pseudograph

4. A subgraph that includes every vertex of the original graph is called a:

a) Complement graph
b) Spanning subgraph
c) Null subgraph
d) Induced subgraph

5. The induced subgraph G[S] for a vertex subset S contains:

a) No edges at all
b) Only edges outside S
c) All edges of G with both endpoints in S
d) Any edges the user selects

6. Two graphs are said to be isomorphic if there exists a mapping between their vertices that preserves:

a) Vertex coordinates
b) Adjacency
c) Edge colours
d) Vertex labels

7. Which of the following is a NECESSARY condition for two graphs to be isomorphic?

a) Same vertex labels
b) Same drawing layout
c) Same number of vertices and edges
d) Same edge colours

8. A walk in which no vertex is repeated is called a:

a) Closed walk

b) Circuit

c) Path

d) Multigraph

9. A closed walk in which no vertex is repeated except the starting/ending vertex is called a:

a) Tree

b) Open walk

c) Path

d) Circuit

10. A connected graph has an Euler circuit if and only if:

a) It is a tree

b) Every vertex has even degree

c) It has a Hamiltonian path

d) It has exactly two odd-degree vertices

11. A connected graph has an Euler path (but not a circuit) if and only if it has exactly:

a) All odd-degree vertices

b) One vertex

c) Two odd-degree vertices

d) Zero odd-degree vertices

12. A circuit that visits every vertex of a graph exactly once and returns to the start is called a:

a) Spanning tree

b) Euler circuit

c) Adjacency cycle

d) Hamiltonian circuit

13. Which data structure is used in Breadth-First Search (BFS) traversal?

a) Linked list only

b) Priority heap only

c) Queue

d) Stack

14. Dijkstra's algorithm computes the shortest path in a graph provided that:

a) The graph is undirected only

b) All edge weights are negative

c) The graph is a tree

d) All edge weights are non-negative

15. The Floyd–Warshall algorithm is used to find:

a) The minimum spanning tree

b) Single-source shortest path only

c) A Hamiltonian circuit

d) All-pairs shortest paths

Answer Key

Q. No.	Answer	Q. No.	Answer
1	A	9	D
2	C	10	C
3	D	11	C
4	B	12	D
5	C	13	C
6	B	14	D
7	C	15	D
8	C		

Broad / Descriptive Questions

1. Define a graph. Distinguish between directed and undirected graphs with suitable examples.
2. Explain the following terms with examples: loop, multigraph, pseudograph, simple graph, finite graph, infinite graph.
3. What is a subgraph? Explain spanning subgraph and induced subgraph with examples. How does removal of a vertex differ from removal of an edge?
4. Define graph isomorphism. State the necessary conditions for two graphs to be isomorphic and illustrate with a worked example.
5. Define walk, path, and circuit. Explain how they differ from one another, giving the length of each with an example graph.
6. Define Euler path and Euler circuit. State and prove (or justify) the condition for the existence of an Euler circuit in a connected graph, referring to the Königsberg bridge problem.
7. Define Hamiltonian path and Hamiltonian circuit. Discuss why no simple general condition exists for Hamiltonicity, and state Dirac's theorem with an example.
8. Compare the sequential (adjacency matrix / incidence matrix) and linked (adjacency list) representations of a graph, highlighting their space complexity and suitability for dense versus sparse graphs.
9. Explain Dijkstra's algorithm for finding the shortest path with its step-by-step procedure, and solve it for a graph of your choice with at least five vertices.
10. Explain the Floyd–Warshall algorithm for all-pairs shortest paths and compare it with Dijkstra's algorithm. Also describe the BFS and DFS traversal algorithms and list at least four real-world applications of graph theory.