Data Structures

Graph terms, matrices and lists

The notes for the lesson “Graph terms, matrices and lists”: chapters 5.1 to 5.2, with their figures, worked examples, exam traps, quizzes and exam question bank, in one file.

2 chapters | Chapters 5.1 to 5.2

About these notes

These are the notes for the University free n easy video lesson **Graph terms, matrices and lists** on Data Structures: chapters 5.1 to 5.2, with every figure, worked example, exam trap and quiz, and the long-answer exam questions on these chapters.

5.1 Graph terminology: directed and undirected graphs

A graph is a set of vertices and a set of edges, where each edge joins two vertices. That is the whole definition, and it is enough to model a road map (cities and roads), a social network (people and friendships), a syllabus (courses and prerequisites) or the web (pages and links). Trees from Unit 4 are a special case: a connected graph with no cycles. This chapter gives the vocabulary every later algorithm is stated in; some syllabi list it as "Basic definition and terminology" and others as the "Graph ADT".

You need: tree terminology from Unit 4 (root, parent, child, leaf) and Big-O from Unit 1.

Vertices, edges and the two kinds of graph

Write a graph as G = (V, E). V is the vertex set (also called nodes), E the edge set. We use |V| or simply V for the number of vertices and |E| or E for the number of edges; the context makes clear whether a set or a count is meant.

In an **undirected graph** an edge is an unordered pair {u, v}: the road runs both ways, friendship is mutual. In a **directed graph** (digraph) an edge is an ordered pair (u, v), drawn as an arrow from u to v: a one-way street, a Twitter follow, a prerequisite "take C before DS". The edge (u, v) does not give you (v, u).

A **weighted graph** attaches a number w(u, v) to each edge: distance in km, cost in rupees, time in minutes, bandwidth. Chapters 5.6 to 5.8 are about weighted graphs; 5.3 to 5.5 ignore weights.

Two vertices joined by an edge are **adjacent** (neighbours), and the edge is **incident** on both. A **self-loop** joins a vertex to itself; **parallel edges** (a multigraph) join the same pair twice. A **simple graph** has neither, and unless a question says otherwise, "graph" means simple graph.

Degree, paths and cycles

The **degree** deg(v) of a vertex in an undirected graph is the number of edges incident on it (a self-loop counts twice). Because every edge contributes exactly 2 to the total, $\Sigma \deg(v) = 2|E|$: the **handshaking lemma**. A direct consequence: the number of odd-degree vertices is even.

In a directed graph each vertex has an **in-degree** (arrows coming in) and an **out-degree** (arrows going out), and Σ in-deg = Σ out-deg = $|E|$, since each arrow is counted once as an exit and once as an entry.

A **walk** is any sequence of vertices $v_0, v_1, \ldots, v_k$ where consecutive vertices are adjacent; its **length** is the number of edges, k. A **path** is a walk with no repeated vertex. A **cycle** is a walk that starts and ends at the same vertex with no other repetition and length at least 3 (in a simple undirected graph). A graph with no cycle is **acyclic**; a directed acyclic graph is a **DAG**, the subject of 5.4.

An undirected graph is **connected** if a path exists between every pair of vertices; otherwise it splits into **connected components** (5.5). For directed graphs the words are **strongly connected** (a directed path each way between every pair) and **weakly connected** (connected if you ignore arrow directions).

Special graphs and the Graph ADT

A **complete graph** K_n has every pair of its n vertices joined, so it has $n(n-1)/2$ edges; a complete digraph has $n(n-1)$. A graph is **sparse** when $|E|$ is close to $|V|$ and **dense** when $|E|$ is close to $|V|^2$. A **subgraph** keeps some vertices and some of the edges among them; a **spanning subgraph** keeps all vertices. A **spanning tree** of a connected graph is a spanning subgraph that is a tree: it has exactly $|V|-1$ edges (5.6). A **bipartite graph** can have its vertices coloured with two colours so that no edge joins two of the same colour.

As an ADT, a graph offers: `addVertex(v)`, `addEdge(u, v[, w])`, `removeEdge(u, v)`, `adjacent(u, v)` (is there an edge?), `neighbours(u)` (list the vertices adjacent to u), and the traversals of 5.3. How fast each runs depends on the representation chosen in 5.2.

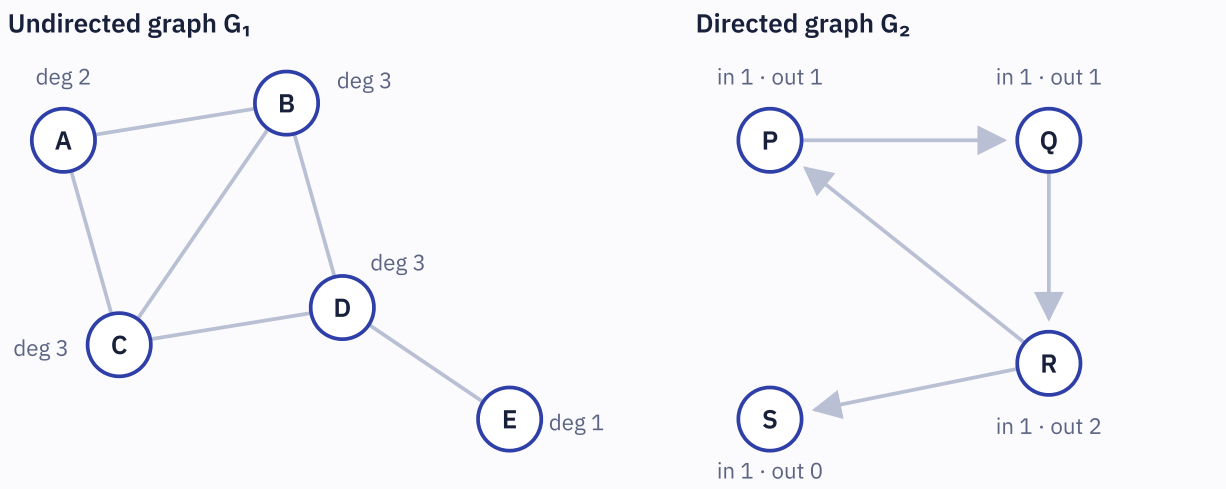

Figure 5.1.1 Left, the undirected graph G_1 on five vertices and six edges with the degree of each vertex; the degrees add to 12, twice the number of edges. Right, the directed graph G_2 with in- and out-degrees; both sums equal the four edges. Notice that in G_2 you can reach S from P (P→Q→R→S) but not P from S.

WORKED EXAMPLE

Question. For the undirected graph G_1 of Figure 5.1.1 (edges AB, AC, BC, BD, CD, DE): (a) list the degree of every vertex and verify the handshaking lemma; (b) is A, B, D, E a path, and is A, B, D, C, A a cycle? (c) how many edges would K_5 on the same vertices have, and how many edges are missing from G_1? (d) For G_2 (edges P→Q, Q→R, R→P, R→S) give in- and out-degrees and say whether it is a DAG.

Solution.

1. Count edges at each vertex. A touches AB, AC: deg(A) = 2. B touches AB, BC, BD: 3. C touches AC, BC, CD: 3. D touches BD, CD, DE: 3. E touches DE: 1.
2. Sum: 2 + 3 + 3 + 3 + 1 = 12 = 2 × 6 edges. The lemma holds. Odd-degree vertices B, C, D, E: four of them, an even count, as it must be.
3. A, B, D, E uses edges AB, BD, DE, all present, and repeats no vertex: a path of length 3. A, B, D, C, A uses AB, BD, DC, CA, all present, returns to A, repeats nothing else, length 4 ≥ 3: a cycle. (A, B, C, A is a shorter cycle, length 3.)
4. K_5 has 5 × 4 / 2 = 10 edges. G_1 has 6, so 4 are missing: AD, AE, BE, CE.
5. G_2: P in 1 (from R) out 1 (to Q); Q in 1 out 1; R in 1 (from Q) out 2 (to P, S); S in 1 out 0. Sum in = 4 = sum out = |E|. P→Q→R→P is a directed cycle, so G_2 is not a DAG.

EXAM TRAPS

- Number of edges of a complete graph: n(n−1)/2 for undirected, n(n−1) for directed. Students halve the directed count or forget to halve the undirected one.
- A self-loop adds 2 to the degree of its vertex, not 1; otherwise the handshaking lemma breaks.
- "Path" means no repeated vertex. If a question says "walk" or "trail", repetition is allowed (a trail repeats no edge). Writing a cycle of length 2 (u, v, u) in a simple undirected graph is wrong; the minimum cycle length is 3.
- Connected vs strongly connected: a directed graph whose arrows form a line P→Q→R is weakly connected but not strongly connected. Say which you mean.
- Favourite 2-mark question: "Define graph, digraph, degree, path, cycle, connected graph, complete graph." Keep each definition to one sentence with the symbol.

CHECK YOURSELF

– REMEMBER **State the handshaking lemma for an undirected graph.** 2 marks

The sum of the degrees of all vertices equals twice the number of edges: Σ deg(v) = 2|E|, because each edge contributes one to the degree of each of its two endpoints.

– REMEMBER **What is a DAG?** 2 marks

A directed acyclic graph: a directed graph with no directed cycle. It models dependencies such as course prerequisites and allows a topological ordering.

– UNDERSTAND **Why can a graph not have exactly three vertices of odd degree?** 3 marks

The degree sum is 2|E|, an even number. The even-degree vertices contribute an even amount, so the odd-degree vertices must also contribute an even amount, which needs an even number of them. Three odd degrees would give an odd sum.

– UNDERSTAND **Give an example where modelling with a directed graph is necessary and an undirected one would be wrong.** 3 marks

Course prerequisites: "C before DS" is one-way. An undirected edge C–DS would also allow DS before C. Other examples: one-way streets, web links, money transfers in UPI.

– APPLY **A simple undirected graph has 8 vertices and every vertex has degree 3. How many edges does it have? Is such a graph possible with 7 vertices?** 4 marks

Σ deg = 8 × 3 = 24 = 2|E|, so |E| = 12. With 7 vertices the degree sum would be 21, odd, which is impossible; so no 3-regular graph on 7 vertices exists.

– APPLY **A digraph has 6 vertices and 9 edges. What is the sum of all in-degrees? If five vertices have in-degree 2, what is the in-degree of the sixth?** 3 marks

Σ in-degree = |E| = 9. Five vertices give 10 > 9, which is impossible; so the data is inconsistent. (If four vertices had in-degree 2, the other two would share 1.)

– ANALYZE **A connected undirected simple graph on n vertices has exactly n – 1 edges. Argue that it must be a tree.** 4 marks

A connected graph contains a spanning tree, which already uses n – 1 edges. If the graph has only n – 1 edges, every edge is in that spanning tree, so there is no extra edge to close a cycle. Connected and acyclic is the definition of a tree.

BEYOND THE SYLLABUS

Not for the exam. Graph theory is its own mathematics (Euler's 1736 bridges of Königsberg problem is the usual starting story; look up Euler paths and Hamiltonian paths). In industry the vocabulary extends to property graphs, where vertices and edges carry key-value attributes, stored in graph databases such as Neo4j and queried with Cypher or the newer ISO GQL standard. Knowledge graphs, fraud detection on UPI transaction networks and the "people you may know" feature are all degree, path and component questions at scale. Search: "Euler path", "property graph model", "GQL standard".

5.2 Graph representations: adjacency matrix and adjacency list

A graph on paper is dots and lines; in memory it must be an array or a set of linked lists. The two standard choices are the adjacency matrix, a V × V table of 0s and 1s (or weights), and the adjacency list, one linked list of neighbours per vertex. The matrix answers "is there an edge u–v?" in constant time but costs V^2 space whatever the edge count; the list costs only V + E space and makes "visit every neighbour of u" fast, which is what every traversal does. Choosing between them is the first design decision in any graph program, and a common tutorial exercise asks you to build both for a sample graph and compare them.

You need: 2-D arrays and linked lists from Unit 2; the terms of 5.1.

Adjacency matrix

Number the vertices 0 to V−1. The adjacency matrix M is a V × V array with M[i][j] = 1 if there is an edge from i to j and 0 otherwise. For an undirected graph the edge {i, j} sets both M[i][j] and M[j][i], so M is symmetric about its main diagonal and you may store only one triangle. For a weighted graph store w(i, j) instead of 1, and a sentinel for "no edge": 0 if weights are never zero, otherwise a large INF constant. The diagonal M[i][i] is 0 unless there are self-loops.

Costs: space $\Theta(V^2)$ always, even for a graph with three edges. `adjacent(u, v)` is one array read, O(1). `neighbours(u)` scans the whole row, O(V). Adding or removing an edge is O(1). The degree of a vertex is its row sum; in a digraph the row sum is the out-degree and the column sum the in-degree. A neat property: entry (i, j) of M^2 counts the walks of length 2 from i to j.

Adjacency list

Keep an array `adj[0..V−1]` of head pointers. `adj[u]` points to a linked list of the vertices adjacent to u (for a weighted graph, each node also stores the weight). An undirected edge {u, v} appears twice: v in u's list and u in v's list, so the lists hold 2E nodes in total; a directed edge (u, v) appears once, in u's list only.

Costs: space Θ(V + E). `neighbours(u)` is a walk down one list, O(deg(u)), which is why BFS and DFS run in O(V + E) with lists but $O(V^2)$ with a matrix. `adjacent(u, v)` must search u's list, O(deg(u)), worst case O(V). Inserting an edge at the head of a list is O(1); deleting one is O(deg(u)). Finding in-degrees in a digraph is awkward: you must scan every list, or keep a second "reverse" list.

In practice the list is often an array of dynamic arrays (a `vector<int> adj[V]` in C++ or a list of lists in Python) rather than linked nodes; the costs are the same. Two other layouts you may meet: the **incidence matrix** (V × E, entry 1 when vertex is an endpoint of the edge) and the **edge list** (just an array of (u, v, w) triples), which is exactly what Kruskal's algorithm in 5.6 wants.

Which to choose

Use the matrix when the graph is dense (E near V^2), when V is small (a few hundred), or when the algorithm keeps asking "is u adjacent to v?" (Floyd-Warshall in 5.8 is naturally a matrix algorithm). Use the list when the graph is sparse, which is nearly every real graph: a road network has about 2 to 4 roads per junction, a social network a few hundred friends per user out of hundreds of millions. A list for 1 million vertices and 3 million edges takes a few tens of megabytes; the matrix would need 10^{12} cells.

```
/* C: adjacency list for an undirected graph with V vertices */
typedef struct Node { int v; int w; struct Node *next; } Node;
Node *adj[100];                       /* heads, all NULL at start */
```

```
void addEdge(int u, int v, int w) {   /* insert at head: O(1) */
    Node *a = malloc(sizeof(Node)); a->v = v; a->w = w; a->next = adj[u]; adj[u] = a;
    Node *b = malloc(sizeof(Node)); b->v = u; b->w = w; b->next = adj[v]; adj[v] = b;
}
```

Because each new node goes at the head, the list of 0 after calls addEdge(0,1), addEdge(0,4) reads 4 → 1, the reverse of insertion order. Traversal order in 5.3 depends on this, so state your list order in an exam answer.

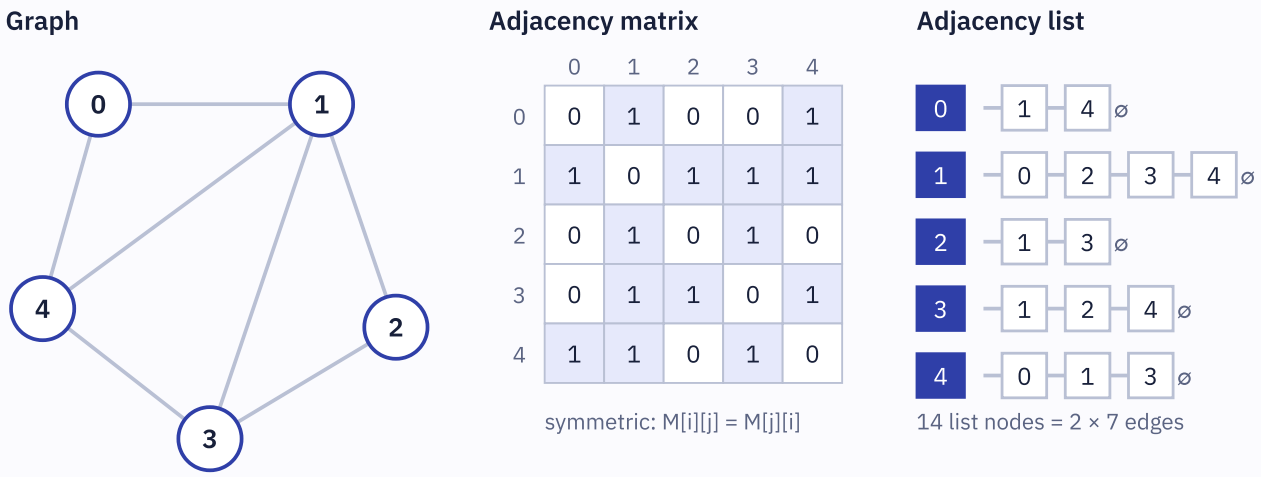

Figure 5.2.1 One undirected graph with five vertices and seven edges in both layouts. The matrix is symmetric with 14 ones in 25 cells; the lists hold 14 nodes, two per edge. Row 1 of the matrix and the list of vertex 1 both say the same thing: 1 is adjacent to 0, 2, 3 and 4, so deg(1) = 4.

WORKED EXAMPLE

Question. A directed weighted graph has vertices A, B, C, D and edges A→B (4), A→C (1), C→B (2), B→D (5), C→D (8). (a) Write its adjacency matrix and adjacency list. (b) An undirected graph has V = 1000 vertices and E = 5000 edges. Estimate the memory of each representation with 4-byte integers and 8-byte pointers, and compare the cost of listing the neighbours of a vertex.

Solution.

1. Order the vertices A, B, C, D as rows and columns. A row holds the edges leaving that vertex; a dash (–) or INF marks no edge.

```
    A   B   C   D
A   0   4   1   –
B   –   0   –   5
C   –   2   0   8
D   –   –   –   0
```

   The matrix is not symmetric: M[A][B] = 4 but M[B][A] = –. Row sums of non-blank entries give out-degrees (A 2, B 1, C 2, D 0); column counts give in-degrees (A 0, B 2, C 1, D 2); both sum to 5 = |E|.

2. Adjacency list, one node per directed edge, storing (vertex, weight):

```
A → (B,4) → (C,1)
B → (D,5)
C → (B,2) → (D,8)
D → ∅
```

   Five nodes in total, one per edge, because the graph is directed.

3. Matrix for V = 1000: 1000 × 1000 = 10^6 cells × 4 bytes = 4,000,000 bytes ≈ 4 MB (1 MB if a 1-byte flag is enough). It does not change if E grows or shrinks.

4. List for E = 5000 undirected edges: 2E = 10,000 nodes. Each node holds an int (4 bytes) and a pointer (8 bytes), about 16 bytes with alignment: 160,000 bytes. Plus 1000 head pointers × 8 bytes = 8,000 bytes. Total ≈ 168 KB, roughly 24 times smaller than the 4 MB matrix.

5. Listing neighbours: average degree = 2E/V = 10. The list visits 10 nodes; the matrix scans 1000 cells. Over a full traversal that is 10,000 steps against 1,000,000: O(V + E) versus O(V^2).

EXAM TRAPS

- Undirected edge, two entries: M[i][j] and M[j][i], and a node in both lists. Forgetting the mirror is the most common lost mark.
- Space of the list is O(V + E), not O(E): the V head pointers count, and a graph with no edges still needs them.
- A weighted matrix needs a "no edge" sentinel distinct from a real weight. If 0 can be a weight, use INF.
- Reading degrees off a digraph matrix: rows give out-degree, columns give in-degree. Not the other way round.
- Favourite question: "Compare adjacency matrix and adjacency list" as a table with rows space, adjacent(u, v), neighbours(u), add edge, best for. Write the five rows with Big-O entries, not prose.

CHECK YOURSELF

– REMEMBER **Define the adjacency matrix of a graph with V vertices.** 2 marks

A V × V array M with M[i][j] = 1 (or the weight w(i, j)) when there is an edge from vertex i to vertex j, and 0 (or INF) otherwise. For an undirected graph M is symmetric.

– REMEMBER **How many list nodes does the adjacency list of an undirected graph with E edges contain?** 1 mark

2E, since each edge {u, v} is stored once in u's list and once in v's.

– UNDERSTAND **Why does BFS take O(V²) time with an adjacency matrix but O(V + E) with lists?** 3 marks

BFS asks for the neighbours of every vertex once. With a matrix each request scans a full row of V cells, V rows in all: V². With lists each request walks deg(u) nodes; the degrees sum to 2E, plus V for the vertices themselves: V + E.

– UNDERSTAND **What does the (i, j) entry of M² tell you?** 2 marks

The number of walks of length 2 from i to j, because $(M^2)[i][j] = \Sigma_k M[i][k] \cdot M[k][j]$ counts the intermediate vertices k adjacent to both.

– APPLY **Draw the adjacency matrix of the undirected graph with vertices 0–3 and edges 0–1, 0–2, 1–2, 2–3. State the degree of vertex 2 from the matrix.** 4 marks

Rows 0: 0 1 1 0; 1: 1 0 1 0; 2: 1 1 0 1; 3: 0 0 1 0. Row 2 sums to 3, so deg(2) = 3. Total ones = 8 = 2 × 4 edges.

– APPLY **A graph has $V = 10^5$ vertices and $E = 3 \times 10^5$ edges. Which representation fits in 100 MB?** 3 marks

Matrix: 10^{10} cells, at least 10 GB even with 1-byte cells: no. List: $2E = 6 \times 10^5$ nodes × 16 bytes ≈ 9.6 MB plus 0.8 MB of heads: yes, comfortably.

– ANALYZE **An algorithm needs the in-degree of every vertex of a digraph stored as adjacency lists. Give the cost and one way to make repeated in-degree queries O(1).** 4 marks

One pass over all lists increments indeg[v] for each node v seen: O(V + E). Store that array (or maintain a reverse adjacency list) so later queries are O(1); update it on each addEdge. Kahn's algorithm in 5.4 does exactly this.

BEYOND THE SYLLABUS

Not for the exam. Large systems use the compressed sparse row (CSR) layout: all neighbour lists packed into one array with an offset array marking where each vertex's slice starts, which is cache-friendly and what GPU graph libraries and sparse-matrix packages (SciPy, cuGraph) use. Graph-processing frameworks such as Pregel and Apache Giraph partition the adjacency lists across machines. The adjacency-matrix view also turns graph algorithms into linear algebra (the GraphBLAS standard), where BFS becomes a matrix-vector product. Search: "CSR format", "GraphBLAS", "edge list vs adjacency list".

Key terms of this lesson

TERM	MEANING OR FORMULA	CHAPTER
Handshaking lemma	Σ deg(v) = 2|E|; in a digraph Σ in = Σ out = |E|	5.1
Complete graph K_n	n(n−1)/2 edges undirected, n(n−1) directed	5.1
Adjacency matrix	V × V array; space Θ(V^2); adjacent(u, v) O(1); neighbours O(V)	5.2
Adjacency list	V head pointers + 2E (or E) nodes; space Θ(V + E); neighbours O(deg u)	5.2

Exam question bank

Long-answer exam questions on these chapters, each with an outline of the points an examiner looks for. Answer one in full on paper before you read its outline.

Remember and understand

- **1. Define graph, directed graph, weighted graph, degree, path, cycle, connected graph, complete graph, spanning tree and DAG, with one example each. (8 marks)**

  One sentence per term with the symbol: G = (V, E); ordered pairs for digraphs; w(u, v); deg(v) with Σ deg = 2|E|; path with no repeated vertex; cycle returns to start; connected: a path between every pair; K_n with n(n−1)/2 edges; spanning tree with |V| − 1 edges; DAG: directed, no cycle. Examples: roads, prerequisites, distances, etc. (Chapter 5.1)

- **2. Compare adjacency matrix and adjacency list representations under space, edge query, neighbour listing, edge insertion and deletion, and suitability for sparse and dense graphs. Illustrate with a 5-vertex graph. (12 marks)**

  Five-row table with Big-O entries: Θ(V²) vs Θ(V + E); O(1) vs O(deg); O(V) vs O(deg); O(1) both; O(1) vs O(deg). Draw both for the sample graph with the mirror entries shown; state that traversals cost O(V²) vs O(V + E); conclude lists for sparse, matrices for dense or small V. (5.2)

Analyze and evaluate

- **12. A city has 2×10^5 junctions and 5×10^5 road segments. Choose a representation and algorithms for (a) checking whether the map is connected, (b) finding the fewest-turn route, (c) finding the fastest route with travel times, (d) planning the cheapest set of roads to keep open after a flood. Justify each with complexity. (16 marks)**

  Adjacency list (sparse: average degree 5). (a) one DFS/BFS, O(V + E), count restarts. (b) BFS, O(V + E). (c) Dijkstra with a heap, O(E log V); not Floyd-Warshall ($V^3 \approx 8 \times 10^{15}$). (d) MST: Kruskal O(E log E) or Prim with heap. (spans 5.2, 5.3, 5.5, 5.6, 5.7)

Glossary (terms used in this lesson)

Adjacency list	One list of neighbours per vertex; Θ(V + E) space.
Adjacency matrix	V × V array with M[i][j] = 1 or the weight when edge i→j exists; Θ(V²) space.

BFS	Breadth-first search: queue-based traversal by increasing distance in edges.
Connected component	Maximal set of mutually reachable vertices in an undirected graph.
DAG	Directed acyclic graph; the only digraphs with a topological order.
Degree	Number of edges at a vertex; in-degree and out-degree for digraphs.
DFS	Depth-first search: stack or recursion based traversal that goes deep before wide.
Floyd-Warshall	All-pairs shortest paths by dynamic programming over intermediate vertices; $\Theta(V^3)$.
Handshaking lemma	$\Sigma \deg(v) = 2\|E\|$.
Kahn's algorithm	Topological sort by repeatedly removing in-degree-0 vertices.
Kruskal's algorithm	MST by taking edges in weight order and skipping those that close a cycle.
Path	Walk with no repeated vertex; its length is its edge count or weight sum.
Spanning tree	Subgraph that is a tree and contains every vertex; $\|V\| - 1$ edges.
Strongly connected	Every vertex reaches every other by a directed path.
Topological order	Vertex sequence with every edge pointing forward.
Weighted graph	Graph with a number w(u, v) on each edge.

Sources and further reading

Textbooks

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*, Part VI "Graph Algorithms" (elementary graph algorithms, minimum spanning trees, single-source and all-pairs shortest paths). The standard reference for the proofs in 5.6 to 5.8.
- Mark Allen Weiss, *Data Structures and Algorithm Analysis in C*, chapter "Graph Algorithms" (topological sort, shortest paths, minimum spanning tree, DFS applications). Closest to the way many syllabi phrase this unit.
- Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed, *Fundamentals of Data Structures in C*, chapter on graphs (representations, traversals, spanning trees, shortest paths).
- Aaron M. Tenenbaum, Yedidyah Langsam and Moshe J. Augenstein, *Data Structures Using C*, chapter on graphs.
- Robert Sedgewick and Kevin Wayne, *Algorithms*, 4th edition, chapter 4 "Graphs" (undirected graphs, directed graphs, MSTs, shortest paths), with free online companion material.

Open web resources (opened while writing)

- VisuAlgo, "Graph Traversal (DFS/BFS)": animated BFS, DFS, topological sort and component finding on your own graphs, free. https://visualgo.net/en/dfsbfs
- CP-Algorithms, "Floyd-Warshall algorithm": loop order, negative-cycle handling and path reconstruction, with code. https://cp-algorithms.com/graph/all-pair-shortest-path-floyd-warshall.html
- Wikipedia, "Dijkstra's algorithm": history (conceived 1956, published 1959), the heap-based complexities quoted in 5.7, and pointers to A* and Johnson's algorithm. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Syllabi: the universal syllabus and a range of university course documents.

Built for the University free n easy YouTube channel from the universal syllabus of Data Structures (Oct 2026). Written with top universities' syllabi and standard textbooks; check a formula against your prescribed book before an exam, and report mistakes on the channel.