DATA STRUCTURE

Chapter 7: Graphs

Problems and Solutions (Exam Oriented)

This sheet contains solved problems on every topic of the chapter, the way they are normally asked in university and diploma examinations: degree and counting problems, matrix and list representation, BFS and DFS tracing, minimum spanning trees, shortest paths, applications and C programs. It ends with a formula sheet and unsolved practice problems with answers.

How to score well in graph problems

- Always draw the graph first (or redraw it neatly from the given matrix or edge list); examiners give marks for the diagram.
- For traversals, show a step table (queue or stack after every step) and then state the final order clearly.
- For Kruskal, Prim and Dijkstra, show every step in a table and write the final total weight or distances.
- When the choice of neighbour is not specified, visit neighbours in alphabetical or numerical order and state this assumption.
- Quote the time complexity at the end, for example $O(V + E)$ or $O(V^2)$.

Part A: Terminology, Degree and Counting

Problem 1. An undirected graph has 5 vertices with degrees 2, 3, 3, 4 and 2. Find the number of edges.

Solution: Sum of degrees = 2 + 3 + 3 + 4 + 2 = 14. By the handshaking property, Σ deg(v) = 2e, so 2e = 14 and **e = 7**.

Problem 2. Can a simple undirected graph have 5 vertices, each of degree 3? Justify your answer.

Solution: The sum of the degrees would be 5 × 3 = 15. The sum of degrees must equal 2e, which is always even. Since 15 is odd, **no such graph exists**.

Problem 3. For the directed graph in Fig. 1 (arcs 1→2, 1→3, 2→3, 3→1, 3→4, 4→2), find the in-degree, out-degree, successors and predecessors of every vertex. Also write one path and one cycle with their lengths.

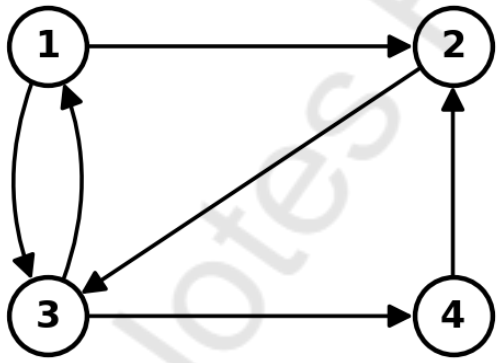

Fig. 1: Directed graph for Problems 3 and 7

Solution: The out-degree is the number of arcs leaving a vertex and the in-degree is the number of arcs entering it.

Vertex	In-degree	Out-degree	Successors	Predecessors
1	1	2	2, 3	3
2	2	1	3	1, 4
3	2	2	1, 4	1, 2
4	1	1	2	3

- Check: total in-degree = 6 = total out-degree = number of arcs.
- Path: 1 → 3 → 4 → 2 has **length 3**.
- Cycle: 1 → 3 → 1 has **length 2** (another cycle is 2 → 3 → 4 → 2 of length 3).

Problem 4. Find the maximum number of edges in a simple undirected graph and in a simple directed graph with 8 vertices.

Solution: Undirected: $n(n-1)/2 = 8 \times 7 / 2 =$ **28**. Directed: every ordered pair of distinct vertices can be an arc, so $n(n-1) = 8 \times 7 =$ **56**.

Problem 5. Find the number of edges and the degree of each vertex in (a) the complete graph K6 and (b) the complete bipartite graph K3,4.

Solution: (a) K6 has $6 \times 5 / 2 =$ **15 edges** and every vertex has degree 5. (b) In K3,4 each of the 3 vertices of the first set is joined to each of the 4 vertices of the second set, so there are $3 \times 4 =$ **12 edges**. The 3 vertices have degree 4 and the 4 vertices have degree 3. Check: $3(4) + 4(3) = 24 = 2 \times 12$.

Problem 6. Prove that in any undirected graph the number of vertices of odd degree is even.

Solution: Let the vertices be split into those of even degree and those of odd degree. Since $\Sigma \deg(v) = 2e$ is even, and the sum of the even degrees is even, the sum of the odd degrees must also be even. A sum of odd numbers is even only when the count of those numbers is even. Hence the number of odd-degree vertices is even. ■

Part B: Representation of Graphs

Problem 7. Write the adjacency matrix and the adjacency list of the directed graph in Fig. 1. Verify the degrees using the matrix.

Solution: Put 1 at row i, column j if there is an arc i → j.

	1	**2**	**3**	**4**	**Row sum (out)**
1	0	1	1	0	2
2	0	0	1	0	1
3	1	0	0	1	2
4	0	1	0	0	1
Column sum (in)	1	2	2	1	

Vertex	**Adjacency list**
1	2 → 3 → NULL
2	3 → NULL
3	1 → 4 → NULL
4	2 → NULL

Row sums give the out-degrees (2, 1, 2, 1) and column sums give the in-degrees (1, 2, 2, 1), which agree with Problem 3.

Problem 8. A weighted undirected graph has vertices A, B, C, D and edges A–B = 5, A–D = 10, B–C = 3, C–D = 1. Write its weighted adjacency matrix and adjacency list.

Solution: In the weighted matrix, an entry is the weight of the edge, 0 on the diagonal, and ∞ where no edge exists. The matrix is symmetric because the graph is undirected.

	A	**B**	**C**	**D**
A	0	5	∞	10
B	5	0	3	∞
C	∞	3	0	1
D	10	∞	1	0

Vertex	**Adjacency list (vertex, weight)**
A	(B, 5) → (D, 10)
B	(A, 5) → (C, 3)
C	(B, 3) → (D, 1)
D	(A, 10) → (C, 1)

Problem 9. The adjacency matrix of an undirected graph with vertices 1 to 5 is given below. Find the edges, the degree of each vertex, and state whether the graph is connected. Name a cycle and a pendant vertex.

	1	**2**	**3**	**4**	**5**
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	0
5	0	0	1	0	0

Solution: The matrix is symmetric, so the graph is undirected. Reading the 1s above the diagonal gives the edges **1–2, 1–4, 2–3, 3–4 and 3–5** (5 edges). Row sums give the degrees: deg(1) = 2, deg(2) = 2, deg(3) = 3, deg(4) = 2, deg(5) = 1. Check: 2 + 2 + 3 + 2 + 1 = 10 = 2 × 5.

- The graph is **connected**: every vertex can be reached from vertex 1 (1–2–3–5 and 1–4).
- Cycle: 1 – 2 – 3 – 4 – 1 (length 4).
- Pendant vertex: 5, because its degree is 1.

Problem 10. An undirected graph has 1000 vertices and 3000 edges. Compare the storage needed by an adjacency matrix and an adjacency list. Which is better?

Solution: Adjacency matrix: 1000 × 1000 = **10,00,000 cells**. Adjacency list: each undirected edge is stored twice, so 2 × 3000 = 6000 list nodes, plus 1000 head pointers, i.e. about **7000 entries**. The matrix needs roughly 140 times more space. The maximum possible number of

edges is 1000 × 999 / 2 = 4,99,500, so the graph has only about 0.6% of them. It is a **sparse graph**, and the adjacency list (space O(V + E)) is the better choice.

Part C: Graph Traversal (BFS and DFS)

Use the undirected graph of Fig. 2 with edges 1–2, 1–3, 2–4, 2–5, 3–5, 3–6, 4–7, 5–7. Neighbours are visited in increasing numerical order.

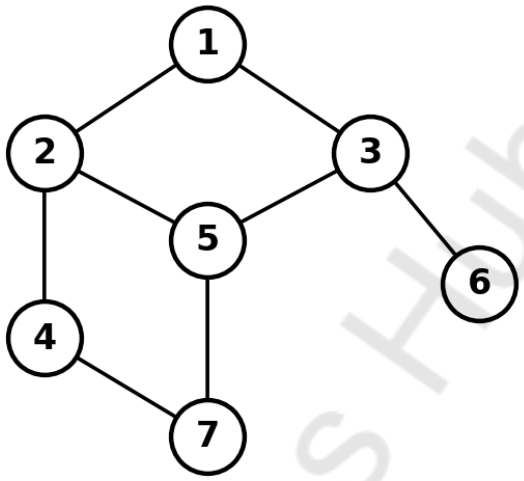

Fig. 2: Graph for Problems 11 to 13

Vertex	Adjacency list
1	2, 3
2	1, 4, 5
3	1, 5, 6
4	2, 7
5	2, 3, 7
6	3
7	4, 5

Problem 11. Perform Breadth First Search on the graph of Fig. 2 starting from vertex 1. Show the queue at every step, and give the level of each vertex.

Solution: Insert 1 into the queue and mark it visited. Then repeatedly remove the front vertex and insert all its unvisited neighbours.

Step	Vertex removed	Inserted	Queue after step
1	1	2, 3	2, 3
2	2	4, 5	3, 4, 5
3	3	6	4, 5, 6
4	4	7	5, 6, 7
5	5	none	6, 7
6	6	none	7
7	7	none	empty

BFS order: 1, 2, 3, 4, 5, 6, 7. Levels: level 0 = {1}, level 1 = {2, 3}, level 2 = {4, 5, 6}, level 3 = {7}. BFS tree edges: 1–2, 1–3, 2–4, 2–5, 3–6, 4–7. Time complexity O(V + E) with an adjacency list.

Problem 12. Perform Depth First Search on the same graph starting from vertex 1 and give the DFS order.

Solution: Visit a vertex, then go to its first unvisited neighbour; backtrack when none is left.

Step	Visit	Action
1	1	Neighbours 2, 3. Go to 2
2	2	Neighbours 1 (visited), 4, 5. Go to 4
3	4	Neighbours 2 (visited), 7. Go to 7
4	7	Neighbours 4 (visited), 5. Go to 5
5	5	Neighbours 2 (visited), 3, 7 (visited). Go to 3
6	3	Neighbours 1 (visited), 5 (visited), 6. Go to 6
7	6	Neighbour 3 visited. Backtrack 6 → 3 → 5 → 7 → 4 → 2 → 1; nothing left

DFS order: 1, 2, 4, 7, 5, 3, 6. DFS tree edges: 1–2, 2–4, 4–7, 7–5, 5–3, 3–6. Time complexity O(V + E) with an adjacency list.

Problem 13. Using the same graph, find the path from vertex 1 to vertex 5 given by the BFS tree and by the DFS tree. Which one is the shortest path?

Solution: In the BFS tree, the parent of 5 is 2 and the parent of 2 is 1, so the path is **1 → 2 → 5 (length 2)**. In the DFS tree the path is **1 → 2 → 4 → 7 → 5 (length 4)**. BFS gives the

shortest path in terms of the number of edges in an unweighted graph; DFS does not guarantee this.

Problem 14. A graph has 8 vertices (1 to 8) and edges 1–2, 2–3, 3–4, 1–4, 5–6, 7–8. Find the number of connected components using DFS, and the number of edges in a spanning forest.

Solution: Start DFS at 1: it visits 1, 2, 3, 4 (component 1). The first unvisited vertex is 5: DFS visits 5, 6 (component 2). The next unvisited vertex is 7: DFS visits 7, 8 (component 3). All vertices are now visited, so there are **3 connected components** and the graph is disconnected (it has no spanning tree). A spanning forest has V − c = 8 − 3 = **5 edges** (3 + 1 + 1).

Part D: Spanning Trees

Problem 15. (a) A connected graph has 7 vertices and 10 edges. How many edges must be removed to obtain a spanning tree? (b) How many spanning trees do K4 and K5 have?

Solution: (a) A spanning tree has n − 1 = 6 edges, so 10 − 6 = **4 edges** must be removed (one from each independent cycle). (b) By Cayley's formula, Kn has n^(n−2) spanning trees: K4 has 4^2 = **16** and K5 has 5^3 = **125**.

For Problems 16 and 17 use the weighted graph of Fig. 3.

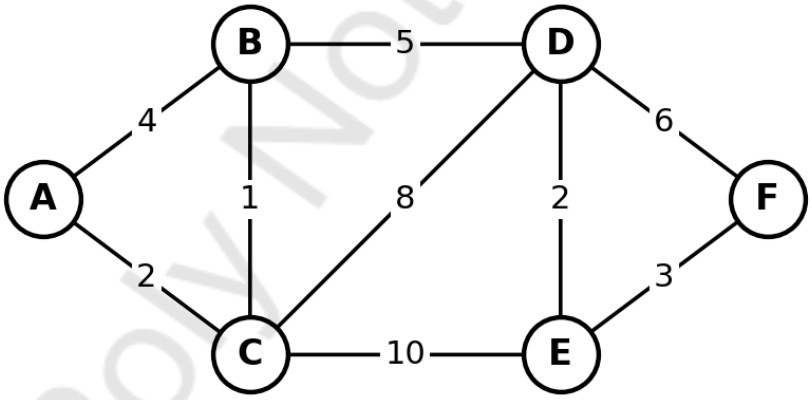

Fig. 3: Weighted graph for Problems 16 and 17

Problem 16. Find the minimum spanning tree of the graph in Fig. 3 using Kruskal's algorithm.

Solution: Sort the edges by weight: BC(1), AC(2), DE(2), EF(3), AB(4), BD(5), DF(6), CD(8), CE(10). Take them in order and reject any edge that forms a cycle.

Step	Edge	Weight	Decision
1	B–C	1	Accept
2	A–C	2	Accept (A joins {B, C})
3	D–E	2	Accept
4	E–F	3	Accept (component {D, E, F})
5	A–B	4	Reject (A and B already connected, cycle A–B–C)
6	B–D	5	Accept (joins {A, B, C} and {D, E, F})

Now 5 edges = n − 1 have been chosen, so stop. **MST edges: BC, AC, DE, EF, BD. Total weight = 1 + 2 + 2 + 3 + 5 = 13.** The MST is drawn with bold lines in Fig. 4. Complexity of Kruskal: O(E log E).

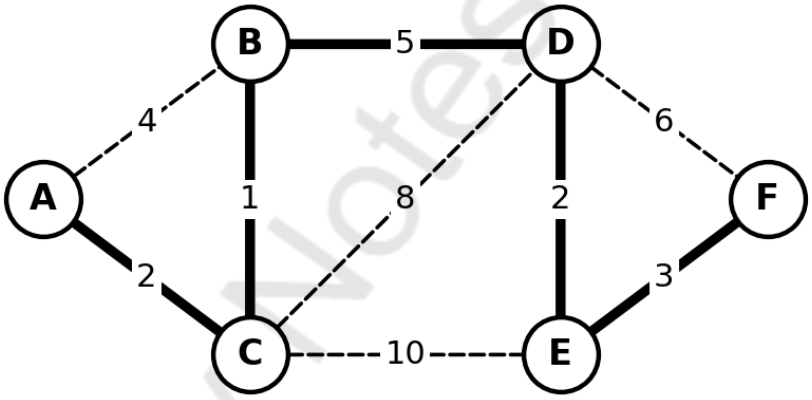

Fig. 4: Minimum spanning tree (bold edges), total weight 13

Problem 17. Find the minimum spanning tree of the graph in Fig. 3 using Prim's algorithm starting from vertex A.

Solution: Grow the tree from A, always adding the cheapest edge that connects a tree vertex to a vertex outside the tree.

Step	Tree vertices	Candidate edges	Edge chosen
1	A	AB(4), AC(2)	AC (2)
2	A, C	AB(4), BC(1), CD(8), CE(10)	BC (1)
3	A, B, C	BD(5), CD(8), CE(10)	BD (5)
4	A, B, C, D	CE(10), DE(2), DF(6)	DE (2)
5	A, B, C, D, E	EF(3), DF(6)	EF (3)

Total weight = 2 + 1 + 5 + 2 + 3 = 13, the same tree as Kruskal's. The MST is unique here because every rejected edge is strictly heavier than the other edges of the cycle it would close. Complexity of Prim: $O(V^2)$ with a matrix, $O(E \log V)$ with a heap.

Problem 18. Which algorithm, Kruskal or Prim, is preferred for (a) a sparse graph and (b) a dense graph? Why?

Solution: (a) **Kruskal** for sparse graphs, since its cost $O(E \log E)$ depends on the number of edges, which is small. (b) **Prim** (with an adjacency matrix, $O(V^2)$) for dense graphs, since E is close to V^2 and sorting all edges becomes expensive.

Part E: Shortest Path

Problem 19. Apply Dijkstra's algorithm to the directed weighted graph of Fig. 5 with source A. Find the shortest distance and path to every vertex.

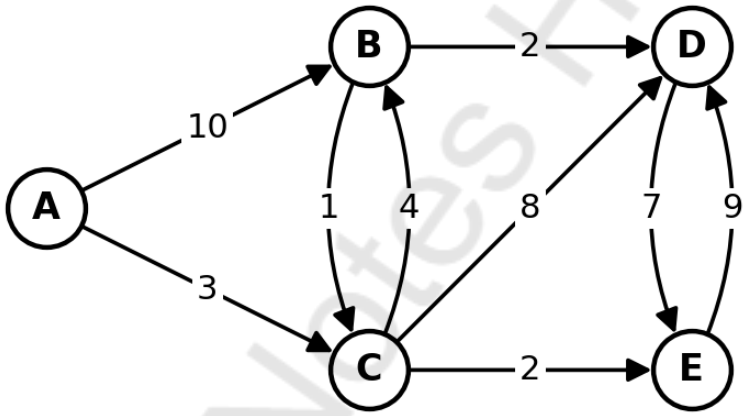

Fig. 5: Directed weighted graph for Problem 19

Solution: Start with dist[A] = 0 and all others ∞. Repeatedly pick the unvisited vertex with the smallest distance and relax its outgoing edges.

Vertex selected	dist[B]	dist[C]	dist[D]	dist[E]
Initial	∞	∞	∞	∞
A (0)	10	3	∞	∞
C (3)	7	3	11	5
E (5)	7	3	11	5
B (7)	7	3	9	5
D (9)	7	3	9	5

Destination	Shortest distance	Path
B	7	A → C → B
C	3	A → C
D	9	A → C → B → D
E	5	A → C → E

Note that the direct edge A → B (10) is longer than the route through C (3 + 4 = 7), and D is best reached through B (7 + 2 = 9) rather than directly from C (3 + 8 = 11).

Problem 20. Show with an example that Dijkstra's algorithm can give a wrong answer when a negative edge is present, and show how Bellman–Ford solves it. Graph: A→B = 2, A→C = 5, C→B = −4, source A.

Solution: Dijkstra: from A, B (2) is closer than C (5), so B is selected and finalised with dist[B] = 2. Later C is selected and the edge C → B would give 5 − 4 = 1, but B is already finalised, so Dijkstra reports 2. The true shortest distance to B is A → C → B = 1, so the answer is **wrong**.

Bellman–Ford relaxes every edge V − 1 = 2 times. Start with dist[A] = 0.

Pass	Edge relaxed	dist[B]	dist[C]
Initial	—	∞	∞
1	A→B	2	∞
1	A→C	2	5
1	C→B (5 − 4 = 1)	1	5
2	no change	1	5

Bellman–Ford gives the correct result dist[B] = 1 in time O(VE).

Problem 21. Find all-pairs shortest distances using the Floyd–Warshall algorithm for a directed graph with vertices 1, 2, 3 and arcs 1→2 = 4, 1→3 = 11, 2→3 = 2, 3→1 = 3.

Solution: Use the rule D[i][j] = min(D[i][j], D[i][k] + D[k][j]) for k = 1, 2, 3 in turn. The initial matrix D0 has 0 on the diagonal and ∞ where there is no arc.

D0 (initial):

	1	**2**	**3**
1	0	4	11
2	∞	0	2
3	3	∞	0

D1 (via vertex 1): D[3][2] = D[3][1] + D[1][2] = 3 + 4 = 7.

	1	**2**	**3**
1	0	4	11
2	∞	0	2
3	3	7	0

D2 (via vertex 2): D[1][3] = D[1][2] + D[2][3] = 4 + 2 = 6, which is less than 11.

	1	2	3
1	0	4	6
2	∞	0	2
3	3	7	0

D3 (via vertex 3): D[2][1] = D[2][3] + D[3][1] = 2 + 3 = 5. No other entry improves.

	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

D3 is the final answer. For example, the shortest distance from 2 to 1 is 5 (path 2 → 3 → 1). Time complexity $O(V^3)$.

Part F: Application Problem

Problem 22. Six tasks 0 to 5 have prerequisites given by the directed graph with arcs 5→2, 5→0, 4→0, 4→1, 2→3, 3→1 (an arc u → v means u must be done before v). Find a valid order of doing all the tasks (topological order).

Solution: Use the in-degree method. In-degrees: task 0 = 2, task 1 = 2, task 2 = 1, task 3 = 1, task 4 = 0, task 5 = 0. Repeatedly remove a vertex of in-degree 0 and reduce the in-degree of its successors.

Step	Vertex removed	In-degrees reduced	Ready (in-degree 0)
1	4	0 → 1, 1 → 1	5
2	5	2 → 0, 0 → 0	2, 0
3	2	3 → 0	0, 3
4	0	none	3
5	3	1 → 0	1
6	1	none	none

One valid order: 4, 5, 2, 0, 3, 1. All six tasks were removed, so the graph has no cycle (it is a DAG). If some vertices could never reach in-degree 0, the graph would contain a cycle and no topological order would exist. More than one valid order can exist; for example, 5, 4, 2, 3, 0, 1 is also valid.

Part G: Programming Problems in C

Problem 23. Write a C function to print the in-degree and out-degree of every vertex of a directed graph stored as an adjacency matrix.

Solution: Row i sum gives the out-degree and column i sum gives the in-degree.

```
void degrees(int a[][MAX], int n)
{
    int i, j, in, out;
    for (i = 0; i < n; i++) {
        in = out = 0;
        for (j = 0; j < n; j++) {
            out += a[i][j];     /* row sum    */
            in  += a[j][i];     /* column sum */
        }
        printf("Vertex %d: in = %d, out = %d\n", i, in, out);
    }
}
```

Problem 24. Write a C function for Breadth First Search using an adjacency matrix.

```
#define MAX 20
int a[MAX][MAX], visited[MAX], queue[MAX];
int front = 0, rear = -1;

void bfs(int start, int n)
{
    int v, w;
    visited[start] = 1;
    queue[++rear] = start;
    while (front <= rear) {
        v = queue[front++];
        printf("%d ", v);
        for (w = 0; w < n; w++)
            if (a[v][w] == 1 && !visited[w]) {
                visited[w] = 1;
                queue[++rear] = w;
            }
    }
}
```

Every vertex enters the queue at most once, so a queue of size MAX is enough. Time complexity is $O(V^2)$ because each row of the matrix is scanned.

Problem 25. Write a recursive C function for Depth First Search using an adjacency matrix, and show how to count the connected components of an undirected graph.

```
void dfs(int v, int n)
{
    int w;
    visited[v] = 1;
    printf("%d ", v);
    for (w = 0; w < n; w++)
        if (a[v][w] == 1 && !visited[w])
            dfs(w, n);
}

/* counting components */
int count = 0;
for (i = 0; i < n; i++)
    if (!visited[i]) {
        dfs(i, n);
        count++;
    }
```

Each call of dfs from the loop explores one complete component, so count holds the number of connected components. Time complexity is $O(V^2)$.

Problem 26. Write a C function to insert an edge u → v in an adjacency-list representation. How will you use it for an undirected graph?

```
struct node {
    int vertex;
    struct node *next;
};
struct node *adj[MAX];

void addEdge(int u, int v)
{
    struct node *p = (struct node *)malloc(sizeof(struct node));
    p->vertex = v;
    p->next = adj[u];      /* insert at the head */
    adj[u] = p;
}
```

For an undirected graph call both addEdge(u, v) and addEdge(v, u), because each edge appears in the lists of both end vertices. Insertion at the head takes O(1) time.

Quick Revision: Formulas and Complexities

Result	Formula
Handshaking property (undirected)	Σ deg(v) = 2e
Directed graph	Σ in-degree = Σ out-degree = e
Maximum edges, simple undirected graph	n(n − 1)/2
Maximum edges, simple directed graph	n(n − 1)
Edges in complete bipartite graph Km,n	m × n
Edges in a tree / spanning tree	n − 1
Spanning trees of complete graph Kn	n^(n−2)
Edges in a spanning forest with c components	n − c

Topic	Method	Time	Space / note
Representation	Adjacency matrix	Edge check O(1)	$O(V^2)$
Representation	Adjacency list	Neighbours O(degree)	O(V + E)
Traversal	BFS (queue)	O(V + E) list, $O(V^2)$ matrix	Shortest path, unweighted
Traversal	DFS (stack)	O(V + E) list, $O(V^2)$ matrix	Cycles, components
MST	Kruskal	O(E log E)	Sparse graphs
MST	Prim	$O(V^2)$ or O(E log V)	Dense graphs
Shortest path	Dijkstra	$O(V^2)$ or O((V + E) log V)	No negative weights
Shortest path	Bellman–Ford	O(VE)	Negative weights allowed
Shortest path	Floyd–Warshall	$O(V^3)$	All pairs

Practice Problems with Answers

Try these on your own first, then check the answers.

No.	Problem	Answer
1	Number of edges in the complete graph K8.	28
2	A simple graph has 6 vertices, each of degree 3. How many edges does it have?	9
3	A directed graph has 10 arcs. What is the sum of all out-degrees?	10
4	A connected graph has 9 vertices and 15 edges. How many edges must be removed to get a spanning tree?	7
5	Number of edges in the complete bipartite graph K4,5.	20
6	An undirected graph has 50 vertices and 120 edges. How many list nodes does its adjacency list contain?	240
7	Number of spanning trees of K5.	125
8	Maximum number of arcs in a simple directed graph with 6 vertices.	30
9	Which traversal gives the fewest-edge path in an unweighted graph?	BFS
10	Which shortest-path algorithm handles negative edge weights (without negative cycles) from a single source?	Bellman–Ford