DATA STRUCTURE

Chapter 7: Graphs

Problems and Solutions (Exam Oriented): Set 2

This is the second set of solved problems for the chapter. It uses new graphs and covers the same topics again with a little more variety: degree and counting problems, matrix and list conversion, matrix powers, BFS and DFS tracing, cycle detection, minimum spanning trees (including a changed-weight case and a maximum spanning tree), Dijkstra, Bellman–Ford, Warshall's path matrix, topological ordering, bipartite testing and C programs. It ends with common mistakes to avoid and practice problems with answers.

Part A: Terminology, Degree and Counting

Problem 1. For the directed graph in Fig. 1 (arcs P→Q, P→R, Q→S, R→S, S→T, T→P), find the in-degree, out-degree, successors and predecessors of every vertex. Find a path from P to T, a cycle, and say whether every vertex can reach every other vertex.

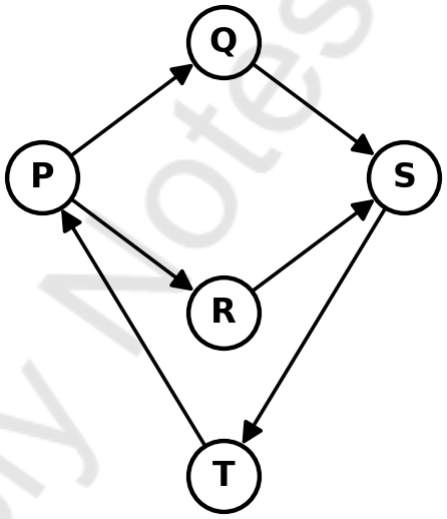

Fig. 1: Directed graph for Problem 1 and Problem 13

Solution: Count the arcs entering and leaving each vertex.

Vertex	In-degree	Out-degree	Successors	Predecessors
P	1	2	Q, R	T
Q	1	1	S	P
R	1	1	S	P
S	2	1	T	Q, R
T	1	1	P	S

- Check: total in-degree = 6 = total out-degree = number of arcs.
- Path from P to T: P → Q → S → T, **length 3** (P → R → S → T is another).
- Cycle: P → Q → S → T → P, **length 4**.

- Every vertex lies on a cycle through P, so each vertex can reach every other vertex: the graph is **strongly connected**.

Problem 2. An undirected graph has 21 edges. It has 3 vertices of degree 4 and all other vertices have degree 3. Find the number of vertices.

Solution: Let x be the number of vertices of degree 3. Sum of degrees = 2 × 21 = 42. So 3(4) + 3x = 42, which gives 3x = 30 and x = 10. Total vertices = 3 + 10 = **13**.

Problem 3. (a) A simple complete graph has 45 edges. Find the number of vertices. (b) A regular graph of degree 4 has 10 vertices. Find the number of edges.

Solution: (a) n(n − 1)/2 = 45 gives n(n − 1) = 90, so n = **10** (10 × 9 = 90). (b) Sum of degrees = 10 × 4 = 40, so 2e = 40 and e = **20**.

Problem 4. (a) How many edges does a tree with 12 vertices have? (b) How many edges does a forest with 12 vertices and 3 trees have?

Solution: A tree with n vertices has n − 1 edges. (a) 12 − 1 = **11**. (b) The three trees have n1 + n2 + n3 = 12 vertices, so the edges are (n1 − 1) + (n2 − 1) + (n3 − 1) = 12 − 3 = **9**. In general a forest has n − c edges.

Problem 5. Which of the cycle graphs C5 (5 vertices) and C6 (6 vertices) is bipartite? Give the reason.

Solution: A graph is bipartite if its vertices can be coloured with two colours so that no edge joins two vertices of the same colour. Going around C6 the colours alternate (1, 2, 1, 2, 1, 2) and the last vertex meets the first correctly, so **C6 is bipartite**. In C5 the colours would alternate 1, 2, 1, 2, 1 and the last vertex would have the same colour as the first, which are adjacent. So **C5 is not bipartite**. A graph is bipartite exactly when it contains no cycle of odd length.

Part B: Representation of Graphs

Problem 6. A directed graph has the adjacency lists 1: 2, 3; 2: 4; 3: 4, 5; 4: 5; 5: (empty). Write its adjacency matrix, find the in-degree and out-degree of each vertex, and name the source and the sink.

Solution: Put 1 in row i, column j for every vertex j in the list of i.

	1	**2**	**3**	**4**	**5**	**Out-degree**
1	0	1	1	0	0	2
2	0	0	0	1	0	1
3	0	0	0	1	1	2
4	0	0	0	0	1	1
5	0	0	0	0	0	0
In-degree	0	1	1	2	2	

Vertex 1 has in-degree 0, so it is the **source**. Vertex 5 has out-degree 0, so it is the **sink**. The total of both degree columns is 6, equal to the number of arcs. The graph has no cycle (it is a DAG).

Problem 7. The adjacency matrix A of a directed graph with vertices 1 to 4 is given. Compute A^2 and explain what its entries mean.

A	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	1	0	0	1
4	0	0	0	0

Solution: Entry (i, j) of A^2 is the sum over k of A[i][k] × A[k][j]. Equivalently, row i of A^2 is the sum of those rows of A that i points to.

A^2	1	2	3	4
1	0	0	1	0
2	1	0	0	1
3	0	1	0	0
4	0	0	0	0

Entry (i, j) of A^2 is the **number of walks of length 2** from i to j. For example, (1, 3) = 1 is the walk 1 → 2 → 3, (2, 4) = 1 is 2 → 3 → 4, (2, 1) = 1 is 2 → 3 → 1, and (3, 2) = 1 is 3 → 1 → 2. In general, entry (i, j) of A^k gives the number of walks of length k from i to j.

Problem 8. An undirected graph has vertices A, B, C, D, E and edges A–B, A–C, B–C, C–D, D–E. Write the adjacency matrix and adjacency list, and verify the degrees.

	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	0	0
C	1	1	0	1	0
D	0	0	1	0	1
E	0	0	0	1	0

Vertex	Adjacency list
A	B → C → NULL
B	A → C → NULL
C	A → B → D → NULL
D	C → E → NULL
E	D → NULL

Solution: The matrix is symmetric. Row sums give the degrees: A = 2, B = 2, C = 3, D = 2, E = 1, and 2 + 2 + 3 + 2 + 1 = 10 = 2 × 5. The adjacency list has 10 nodes, because every undirected edge is stored twice.

Problem 9. A directed graph has 500 vertices and 1200 arcs. Compare the memory of the adjacency matrix and adjacency list, and the cost of listing all out-neighbours of one vertex in each.

Solution: Matrix: 500 × 500 = **2,50,000 cells**. List: each arc is stored once, so 1200 nodes plus 500 head pointers = **1700 entries**, about 147 times less. The maximum possible number of arcs is 500 × 499 = 2,49,500, so the graph is only about 0.5% full: it is sparse. To list the out-neighbours of a vertex, the matrix scans a whole row of 500 entries, while the list visits only the out-degree of that vertex (on average 1200 / 500 = 2.4 nodes). The adjacency list is clearly better.

Part C: Graph Traversal

Use the undirected graph of Fig. 2 for Problems 10 to 12 and 14. Its edges are A–B, A–C, A–D, B–E, C–E, C–F, D–F, E–G, F–G. Neighbours are visited in alphabetical order.

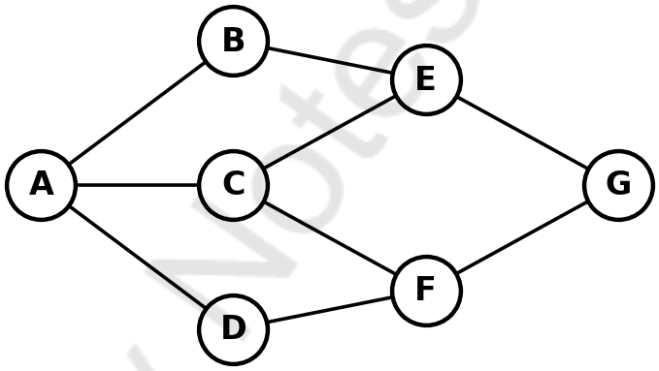

Fig. 2: Undirected graph for Problems 10 to 12 and 14

Vertex	Adjacency list
A	B, C, D
B	A, E
C	A, E, F
D	A, F
E	B, C, G
F	C, D, G
G	E, F

Problem 10. Perform BFS on the graph of Fig. 2 starting from A. Show the queue at each step and write the level of every vertex.

Solution: Insert A, then repeatedly remove the front vertex and insert its unvisited neighbours.

Step	Vertex removed	Inserted	Queue after step
1	A	B, C, D	B, C, D
2	B	E	C, D, E
3	C	F	D, E, F
4	D	none	E, F
5	E	G	F, G
6	F	none	G
7	G	none	empty

BFS order: A, B, C, D, E, F, G. Levels: level 0 = {A}, level 1 = {B, C, D}, level 2 = {E, F}, level 3 = {G}. So G is at distance 3 from A.

Problem 11. Perform DFS on the same graph starting from A and give the DFS order and the DFS tree edges.

Step	Visit	Action
1	A	Neighbours B, C, D. Go to B
2	B	Neighbours A (visited), E. Go to E
3	E	Neighbours B (visited), C, G. Go to C
4	C	Neighbours A (visited), E (visited), F. Go to F
5	F	Neighbours C (visited), D, G. Go to D
6	D	Neighbours A, F both visited. Backtrack to F
7	G	From F go to G. Neighbours E, F visited. Backtrack to A; nothing left

DFS order: A, B, E, C, F, D, G. DFS tree edges: A–B, B–E, E–C, C–F, F–D, F–G. Notice that the DFS tree is much deeper than the BFS tree.

Problem 12. Use DFS to show that the graph of Fig. 2 contains a cycle and write one cycle.

Solution: In an undirected graph, if DFS finds a neighbour that is already visited and is **not the vertex it just came from (the parent)**, the edge to that neighbour is a back edge and it closes a cycle. In the DFS of Problem 11, at vertex C (reached from E) the neighbour A is already visited and A is not the parent of C. So C–A is a back edge. Following the tree path

from A to C gives the cycle **A → B → E → C → A** (length 4). Other cycles exist too, for example A → C → F → D → A.

Problem 13. Perform BFS and DFS on the directed graph of Fig. 1 starting from P, visiting successors in alphabetical order.

Solution: In a directed graph only the outgoing arcs are followed.

Step	BFS: removed	BFS: queue after	DFS: visit
1	P	Q, R	P, go to Q
2	Q	R, S	Q, go to S
3	R	S	S, go to T
4	S	T	T (successor P visited), backtrack
5	T	empty	Back at P, go to R
6	—	—	R (successor S visited), done

BFS order: P, Q, R, S, T. DFS order: P, Q, S, T, R. Since the graph is strongly connected, both traversals reach every vertex from P.

Problem 14. How many different shortest paths (fewest edges) are there from A to G in Fig. 2? List them.

Solution: BFS levels are A (0); B, C, D (1); E, F (2); G (3), so the shortest length is 3. Count the ways to reach each level: E is reached from B or C (2 ways), F is reached from C or D (2 ways), and G is reached from E or F, giving 2 + 2 = **4 shortest paths**: A–B–E–G, A–C–E–G, A–C–F–G and A–D–F–G.

Part D: Spanning Trees

Use the weighted graph of Fig. 3 for Problems 15 to 18 and 19.

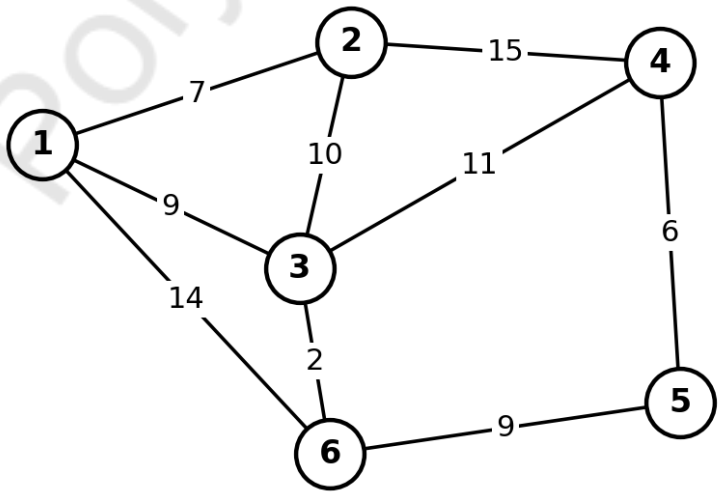

Fig. 3: Weighted graph for Problems 15 to 19

Problem 15. Find the minimum spanning tree of Fig. 3 using Kruskal's algorithm.

Solution: Sorted edges: 3–6 (2), 4–5 (6), 1–2 (7), 1–3 (9), 5–6 (9), 2–3 (10), 3–4 (11), 1–6 (14), 2–4 (15).

Step	Edge	Weight	Decision
1	3–6	2	Accept
2	4–5	6	Accept
3	1–2	7	Accept
4	1–3	9	Accept (joins {1, 2} and {3, 6})
5	5–6	9	Accept (joins {4, 5} and {1, 2, 3, 6}); all 6 vertices connected

Five edges = n − 1 are selected, so stop. **Total weight = 2 + 6 + 7 + 9 + 9 = 33.** The remaining edges 2–3, 3–4, 1–6 and 2–4 would each form a cycle. The MST is shown with bold lines in Fig. 4.

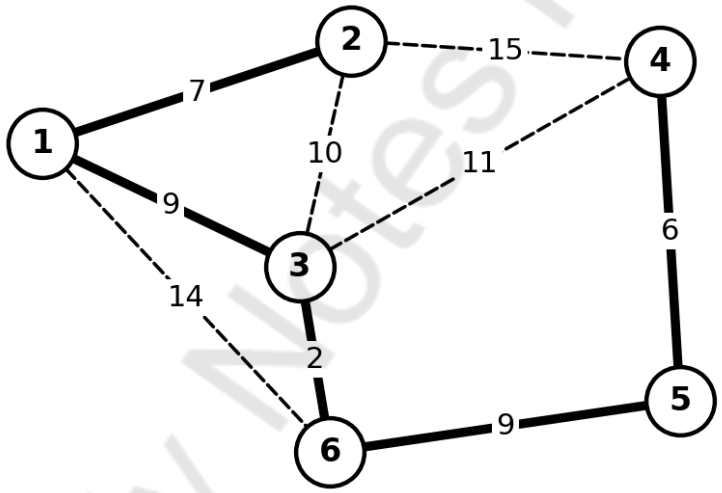

Fig. 4: Minimum spanning tree of Fig. 3 (bold edges), total weight 33

Problem 16. Find the minimum spanning tree of Fig. 3 using Prim's algorithm starting from vertex 1.

Step	Tree vertices	Candidate edges	Edge chosen
1	1	1–2 (7), 1–3 (9), 1–6 (14)	1–2 (7)
2	1, 2	1–3 (9), 1–6 (14), 2–3 (10), 2–4 (15)	1–3 (9)
3	1, 2, 3	1–6 (14), 2–4 (15), 3–4 (11), 3–6 (2)	3–6 (2)
4	1, 2, 3, 6	2–4 (15), 3–4 (11), 5–6 (9)	5–6 (9)
5	1, 2, 3, 5, 6	2–4 (15), 3–4 (11), 4–5 (6)	4–5 (6)

Total weight = 7 + 9 + 2 + 9 + 6 = 33, the same tree as Kruskal's, although the edges were added in a different order.

Problem 17. In Fig. 3 the weight of edge 4–5 is increased from 6 to 12. Does the MST change? Find the new MST weight.

Solution: The old MST used edge 4–5, so it changes. Sorted edges now: 3–6 (2), 1–2 (7), 1–3 (9), 5–6 (9), 2–3 (10), 3–4 (11), 4–5 (12), 1–6 (14), 2–4 (15). Kruskal: accept 3–6, 1–2, 1–3 (now {1, 2, 3, 6}), accept 5–6 (adds 5), reject 2–3 (cycle), **accept 3–4** (adds 4). The new MST is {3–6, 1–2, 1–3, 5–6, 3–4} with weight 2 + 7 + 9 + 9 + 11 = **38**. Edge 3–4 (11) has replaced edge 4–5; the weight rises by 5 (from 33 to 38).

Problem 18. Find a maximum spanning tree of Fig. 3 (the spanning tree of largest total weight).

Solution: Use Kruskal's algorithm but sort the edges in **decreasing** order: 2–4 (15), 1–6 (14), 3–4 (11), 2–3 (10), 1–3 (9), 5–6 (9), 1–2 (7), 4–5 (6), 3–6 (2). Accept 2–4, 1–6 and 3–4. Reject 2–3, because 2 and 3 are already connected through 4. Accept 1–3 (joins {1, 6} with {2, 3, 4}) and 5–6 (adds vertex 5). The tree {2–4, 1–6, 3–4, 1–3, 5–6} has weight 15 + 14 + 11 + 9 + 9 = **58**. (Equivalently, negate all weights and find the minimum spanning tree.)

Part E: Shortest Path and Reachability

Problem 19. Apply Dijkstra's algorithm on the graph of Fig. 3 with source vertex 1. Find the shortest distance and path to every vertex.

Solution: Start with dist[1] = 0 and all others ∞. Select the unvisited vertex of smallest distance and relax its edges.

Vertex selected	dist[2]	dist[3]	dist[4]	dist[5]	dist[6]
Initial	∞	∞	∞	∞	∞
1 (0)	7	9	∞	∞	14
2 (7)	7	9	22	∞	14
3 (9)	7	9	20	∞	11
6 (11)	7	9	20	20	11
4 (20)	7	9	20	20	11
5 (20)	7	9	20	20	11

Destination	Shortest distance	Path
2	7	1 → 2
3	9	1 → 3
4	20	1 → 3 → 4
5	20	1 → 3 → 6 → 5
6	11	1 → 3 → 6

Notice that the direct edge 1–6 (14) is longer than the route 1 → 3 → 6 (9 + 2 = 11), and that vertex 4 is reached more cheaply through 3 (9 + 11 = 20) than through 2 (7 + 15 = 22).

Problem 20. Use the Bellman–Ford algorithm on the directed graph with arcs (in this order) S→A = 4, S→B = 5, B→A = −3, A→C = 2, B→C = 6 and source S. How can a negative cycle be detected?

Solution: Set dist[S] = 0 and the others to ∞. Relax all arcs in the given order; repeat up to V − 1 = 3 times, stopping early if no distance changes.

Pass	Arc relaxed	dist[A]	dist[B]	dist[C]
Initial	—	∞	∞	∞
1	S→A	4	∞	∞
1	S→B	4	5	∞
1	B→A (5 − 3 = 2)	2	5	∞
1	A→C (2 + 2 = 4)	2	5	4
1	B→C (11, no change)	2	5	4
2	all arcs	2	5	4

Pass 2 changes nothing, so the algorithm stops. Result: **A = 2 (S → B → A), B = 5 (S → B), C = 4 (S → B → A → C)**. To detect a negative cycle, do one more (V-th) pass: if any arc can still be relaxed, the graph has a negative cycle reachable from the source and no shortest path exists.

Problem 21. A directed graph has vertices 1 to 4 and arcs 1→2, 2→3, 3→4, 4→2. Find its path (reachability) matrix using Warshall's algorithm.

Solution: Start with P0 = adjacency matrix. For k = 1 to 4, if P[i][k] = 1 then row i is OR-ed with row k (i.e. P[i][j] = P[i][j] OR (P[i][k] AND P[k][j])).

P0 (adjacency matrix). Column 1 is all zeros, so k = 1 changes nothing and P1 = P0:

	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	1	0	0

P2 (k = 2): rows 1 and 4 point to 2, so they take row 2 (which is 0 0 1 0):

	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	1
4	0	1	1	0

P3 (k = 3): rows 1, 2 and 4 point to 3, so they take row 3 (0 0 0 1):

	1	2	3	4
1	0	1	1	1
2	0	0	1	1
3	0	0	0	1
4	0	1	1	1

P4 (k = 4): every row points to 4, so every row takes row 4 (0 1 1 1). This is the final path matrix:

	1	2	3	4
1	0	1	1	1
2	0	1	1	1
3	0	1	1	1
4	0	1	1	1

Vertex 1 reaches 2, 3 and 4. Each of 2, 3 and 4 reaches all of 2, 3 and 4 (including itself, because they lie on the cycle 2 → 3 → 4 → 2). No vertex reaches vertex 1. Time complexity is $O(V^3)$.

Part F: Application Problems

Problem 22. Six courses C1 to C6 have prerequisites C1→C3, C2→C3, C3→C4, C3→C5, C4→C6, C5→C6 (an arc u → v means u must be completed before v). Find a valid order of study. If any number of independent courses can be taken together in one semester, what is the minimum number of semesters?

Solution: In-degrees: C1 = 0, C2 = 0, C3 = 2, C4 = 1, C5 = 1, C6 = 2. Repeatedly remove a course of in-degree 0.

Step	Removed	In-degrees reduced	Ready (in-degree 0)
1	C1	C3: 2 → 1	C2
2	C2	C3: 1 → 0	C3
3	C3	C4: 1 → 0, C5: 1 → 0	C4, C5
4	C4	C6: 2 → 1	C5
5	C5	C6: 1 → 0	C6
6	C6	none	none

Valid order: C1, C2, C3, C4, C5, C6. For the semester count, group the courses by level: semester 1 = {C1, C2}, semester 2 = {C3}, semester 3 = {C4, C5}, semester 4 = {C6}. The minimum is **4 semesters**, which equals the number of vertices on the longest path (C1 → C3 → C4 → C6).

Problem 23. Use BFS colouring to test whether the graph with edges 1–2, 2–3, 3–4, 4–1, 3–5 is bipartite. What happens if the edge 1–3 is added?

Solution: Colour the start vertex 1 with colour X. In BFS, give every neighbour of a vertex the opposite colour; if a neighbour already has the same colour, the graph is not bipartite.

- Vertex 1 = X. Its neighbours 2 and 4 get colour Y.
- Vertex 2 (Y): neighbour 3 gets X. Vertex 4 (Y): neighbours 1 (X) and 3 (X) are consistent.
- Vertex 3 (X): neighbours 2 (Y), 4 (Y) are consistent, and 5 gets Y.

Every edge joins different colours, so the graph **is bipartite** with sets {1, 3} and {2, 4, 5}. If edge 1–3 is added, vertices 1 and 3 are both coloured X yet are adjacent. This gives a conflict, so the graph is **not bipartite** (it now contains the odd cycle 1 – 2 – 3 – 1 of length 3).

Part G: Programming Problems in C

Problem 24. Write C functions to (a) check whether a graph given as an adjacency matrix is undirected (symmetric) and (b) count the edges of an undirected graph with no self-loops.

Solution: An undirected graph has a symmetric matrix, and the sum of all entries is twice the number of edges.

```
int isSymmetric(int a[][MAX], int n)
{
    int i, j;
    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++)
            if (a[i][j] != a[j][i])
                return 0;          /* directed graph */
    return 1;                      /* undirected     */
}

int countEdges(int a[][MAX], int n)
{
    int i, j, sum = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            sum += a[i][j];
    return sum / 2;
}
```

Problem 25. Write a C function for Prim's algorithm using a cost matrix. Vertices are numbered 0 to n − 1, and cost[i][j] = INF when there is no edge.

```
#define INF 9999

int prim(int cost[][MAX], int n)
{
    int visited[MAX] = {0};
    int i, j, u = 0, v = 0, k, min, total = 0;
    visited[0] = 1;                         /* start at vertex 0 */
    for (k = 1; k < n; k++) {               /* n - 1 edges       */
        min = INF;
        for (i = 0; i < n; i++)
            if (visited[i])
                for (j = 0; j < n; j++)
                    if (!visited[j] && cost[i][j] < min) {
                        min = cost[i][j];
                        u = i;  v = j;
                    }
        visited[v] = 1;
        total += min;
        printf("Edge %d-%d cost %d\n", u, v, min);
    }
    return total;                           /* MST weight */
}
```

Each round picks the cheapest edge from the tree to a vertex outside it. As written the function takes $O(V^3)$ time; keeping a key[] array of the cheapest known edge to each outside vertex reduces it to $O(V^2)$. The graph must be connected.

Problem 26. Write a C function for Dijkstra's algorithm using a cost matrix to find the shortest distance from a source vertex to all other vertices.

```
void dijkstra(int cost[][MAX], int n, int src, int dist[])
{
    int visited[MAX] = {0};
    int i, k, u, min;
    for (i = 0; i < n; i++)
        dist[i] = INF;
    dist[src] = 0;
    for (k = 0; k < n; k++) {
        min = INF;  u = -1;
        for (i = 0; i < n; i++)          /* nearest unvisited vertex
*/
            if (!visited[i] && dist[i] < min) {
                min = dist[i];  u = i;
            }
        if (u == -1) break;              /* rest are unreachable */
        visited[u] = 1;
        for (i = 0; i < n; i++)          /* relax edges of u */
            if (!visited[i] && cost[u][i] < INF &&
                dist[u] + cost[u][i] < dist[i])
                dist[i] = dist[u] + cost[u][i];
    }
}
```

The outer loop runs n times and each round scans n vertices, so the time complexity is $O(V^2)$. It works only for non-negative weights.

Common Mistakes to Avoid

- Forgetting that an undirected edge appears **twice** in an adjacency list and twice in the matrix (so the matrix sum is 2e).
- Using $n(n - 1)/2$ for a directed graph; the maximum number of arcs is $n(n - 1)$.
- In BFS, marking a vertex visited only when it is removed from the queue instead of when it is inserted, which puts duplicates in the queue.
- Writing a DFS order without following the stated neighbour order (alphabetical or numerical).
- In Kruskal's algorithm, accepting an edge that joins two vertices of the same component (this creates a cycle).
- In Prim's algorithm, picking the cheapest edge of the whole graph instead of the cheapest edge leaving the current tree.
- Applying Dijkstra's algorithm to a graph with negative edge weights; use Bellman–Ford there.
- Stopping a Kruskal or Prim solution without stating the total weight of the MST.

Practice Problems with Answers

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

No.	Problem	Answer
1	A graph has 12 edges and every vertex has degree 3. How many vertices does it have?	8
2	A complete graph has 66 edges. How many vertices does it have?	12
3	A forest has 20 vertices and 4 trees. How many edges does it have?	16
4	Number of edges in the complete bipartite graph K5,6.	30
5	An undirected graph has 200 vertices and 500 edges. How many nodes does its adjacency list contain?	1000
6	Number of cells in the adjacency matrix of a directed graph with 30 vertices.	900
7	Number of spanning trees of K6.	1296
8	Maximum number of arcs in a simple directed graph with 7 vertices.	42
9	Maximum number of edges in an acyclic undirected graph with 15 vertices.	14
10	Which algorithm computes the path (reachability) matrix of a directed graph?	Warshall's