DATA STRUCTURE

Chapter 7: Graphs

Problems and Solutions (Exam Oriented): Set 3

This is the third set of solved problems for the chapter, again on fresh graphs. It adds problem types that are often asked in examinations but were not covered earlier: graphical degree sequences, the complement of a graph, the incidence matrix, iterative DFS with an explicit stack, a real-life minimum cost network, Floyd–Warshall on four vertices, topological sorting by DFS finish times, true or false questions with reasons, and complete C programs. It ends with an algorithm summary table and practice problems with answers.

Part A: Terminology and Properties

Problem 1. For the directed graph of Fig. 1 (arcs 1→2, 1→3, 2→4, 3→4, 3→5, 4→6, 5→6), find the in-degree, out-degree, successors and predecessors of each vertex. Identify the source and the sink, list all paths from 1 to 6 with their lengths, and state whether the graph has a cycle.

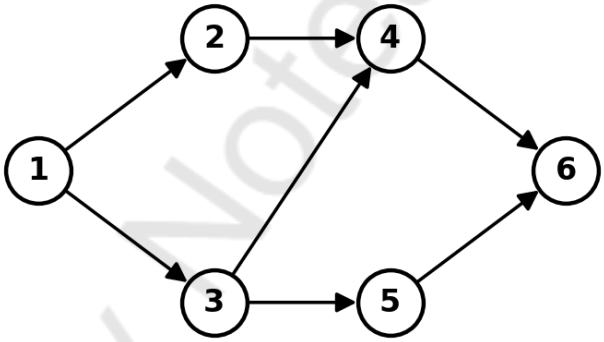

Fig. 1: Directed graph for Problems 1 and 18

Vertex	In-degree	Out-degree	Successors	Predecessors
1	0	2	2, 3	—
2	1	1	4	1
3	1	2	4, 5	1
4	2	1	6	2, 3
5	1	1	6	3
6	2	0	—	4, 5

- Check: total in-degree = 7 = total out-degree = number of arcs.
- The **source** (in-degree 0) is vertex 1 and the **sink** (out-degree 0) is vertex 6.
- Paths from 1 to 6: 1→2→4→6, 1→3→4→6 and 1→3→5→6. All three have **length 3**.
- No path returns to its starting vertex, so there is **no cycle**: the graph is a directed acyclic graph (DAG).

Problem 2. Decide whether a simple undirected graph can have the degree sequence (a) 3, 3, 2, 2, 2, (b) 4, 4, 3, 1, (c) 3, 3, 3, 1. Justify each answer.

Solution: (a) **Possible.** The sum is 12, so there are 6 edges. One graph: vertices a, b (degree 3) and c, d, e (degree 2) with edges ab, ac, ad, bc, be, de. Check: deg a = 3, b = 3, c = 2, d = 2, e = 2.

(b) **Not possible.** In a simple graph with 4 vertices the largest possible degree is 3, but the sequence contains 4 (a vertex cannot be adjacent to itself or twice to the same vertex).

(c) **Not possible.** The sum is 10 (even), but the three vertices of degree 3 must each be adjacent to all three other vertices. Then the vertex of degree 1 would be adjacent to all three of them and would have degree 3, not 1. (Equivalently, the sequence is not graphical.)

Problem 3. (a) A simple graph has 8 vertices and 10 edges. How many edges does its complement have? (b) What is the maximum number of edges in a disconnected simple graph with 10 vertices? (c) What is the minimum number of edges in a connected graph with 10 vertices?

Solution: (a) The graph and its complement together make the complete graph K8, which has 8 × 7 / 2 = 28 edges. So the complement has 28 − 10 = **18 edges**.

(b) A disconnected graph has at least two components. The number of edges is largest when one component is a complete graph on 9 vertices and the other is a single isolated vertex: 9 × 8 / 2 = **36 edges**. Any other split gives fewer (for example 8 + 2 vertices gives 28 + 1 = 29).

(c) A connected graph on n vertices needs at least n − 1 edges (a tree), so the minimum is 10 − 1 = **9 edges**.

Part B: Representation of Graphs

Problem 4. A weighted directed graph has vertices A, B, C, D and arcs A→B = 3, A→C = 8, B→D = 2, C→B = 1, D→C = 4. Write the weighted adjacency matrix and the adjacency list, and find the length of the path A → B → D → C.

Solution: Row = tail of the arc, column = head. The diagonal is 0 and missing arcs are ∞. The matrix is not symmetric because the graph is directed.

	A	**B**	**C**	**D**
A	0	3	8	∞
B	∞	0	∞	2
C	∞	1	0	∞
D	∞	∞	4	0

Vertex	**Adjacency list (vertex, weight)**
A	(B, 3) → (C, 8)
B	(D, 2)
C	(B, 1)
D	(C, 4)

The length of the path A → B → D → C is the sum of its arc weights: 3 + 2 + 4 = **9**.

Problem 5. The adjacency matrix of an undirected graph with vertices 1 to 6 is given. Find the edges, the degree of every vertex, the connected components, and the number of edges in a spanning forest.

	1	**2**	**3**	**4**	**5**	**6**
1	0	1	1	0	0	0
2	1	0	1	0	0	0
3	1	1	0	0	0	0
4	0	0	0	0	1	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

Solution: The matrix is symmetric. Reading the 1s above the diagonal, the edges are **1–2, 1–3, 2–3 and 4–5** (4 edges). Row sums give the degrees: 2, 2, 2, 1, 1, 0, and 2 + 2 + 2 + 1 + 1 + 0 = 8 = 2 × 4.

- Connected components: **{1, 2, 3}** (a triangle), **{4, 5}** and **{6}** (an isolated vertex). So there are 3 components and the graph is disconnected.
- Vertices 4 and 5 are pendant (degree 1) and vertex 6 is isolated (degree 0).
- A spanning forest has n − c = 6 − 3 = **3 edges** (two from the triangle and one for 4–5).

Problem 6. An undirected graph has vertices A, B, C, D and edges e1 = A–B, e2 = B–C, e3 = C–D, e4 = A–D, e5 = A–C. Write its incidence matrix and use it to find the degrees.

Solution: The **incidence matrix** has one row per vertex and one column per edge. An entry is 1 if the vertex is an end of that edge, otherwise 0.

	e1	**e2**	**e3**	**e4**	**e5**	**Row sum**
A	1	0	0	1	1	3
B	1	1	0	0	0	2
C	0	1	1	0	1	3
D	0	0	1	1	0	2
Column sum	2	2	2	2	2	

Every column sums to 2 because each edge has exactly two ends. The row sums give the degrees: A = 3, B = 2, C = 3, D = 2. Their total 10 is twice the number of edges (5). The incidence matrix has size V × E, so it needs more space than the adjacency matrix when E is larger than V.

Problem 7. An undirected graph has 10,000 vertices and every vertex has an average degree of 5. Which representation would you use and why? Compare the storage.

Solution: Number of edges = 10,000 × 5 / 2 = 25,000. Adjacency matrix: 10,000 × 10,000 = **10^8 cells**; even at 1 byte per cell this is about 100 MB. Adjacency list: 2 × 25,000 = 50,000 list nodes plus 10,000 head pointers = **60,000 entries**, about 1667 times fewer. The graph is extremely sparse (average degree 5 out of a possible 9,999), so the **adjacency list** is the right choice, and a traversal on it costs O(V + E) instead of O(V^2).

Part C: Graph Traversal

Use the undirected graph of Fig. 2 for Problems 8 to 11. Neighbours are visited in increasing numerical order.

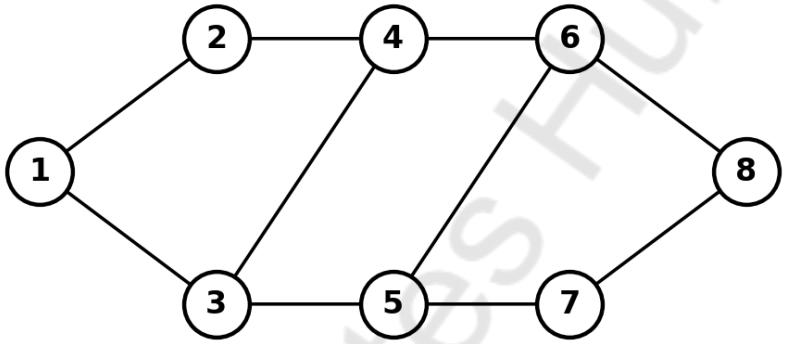

Fig. 2: Undirected graph for Problems 8 to 11

Vertex	Adjacency list
1	2, 3
2	1, 4
3	1, 4, 5
4	2, 3, 6
5	3, 6, 7
6	4, 5, 8
7	5, 8
8	6, 7

Problem 8. Perform BFS on the graph of Fig. 2 starting from vertex 4. Show the queue at each step and find the level of every vertex.

Solution: Insert vertex 4, then repeatedly remove the front vertex and insert its unvisited neighbours.

Step	Vertex removed	Inserted	Queue after step
1	4	2, 3, 6	2, 3, 6
2	2	1	3, 6, 1
3	3	5	6, 1, 5
4	6	8	1, 5, 8
5	1	none	5, 8
6	5	7	8, 7
7	8	none	7
8	7	none	empty

BFS order: 4, 2, 3, 6, 1, 5, 8, 7. Levels: level 0 = {4}, level 1 = {2, 3, 6}, level 2 = {1, 5, 8}, level 3 = {7}. So vertex 7 is at distance 3 from vertex 4.

Problem 9. How many different shortest paths (fewest edges) are there from vertex 4 to vertex 7 in Fig. 2? List them.

Solution: From the BFS levels of Problem 8, the shortest length is 3. Vertex 7 is adjacent to 5 and 8, both at level 2. Vertex 5 can be reached from 3 or 6 (2 ways) and vertex 8 only from 6 (1 way). Hence there are 2 + 1 = **3 shortest paths**: 4–3–5–7, 4–6–5–7 and 4–6–8–7.

Problem 10. Perform DFS (recursive) on the same graph starting from vertex 1 and give the DFS order and the DFS tree edges.

Step	Visit	Action
1	1	Neighbours 2, 3. Go to 2
2	2	Neighbours 1 (visited), 4. Go to 4
3	4	Neighbours 2 (visited), 3, 6. Go to 3
4	3	Neighbours 1, 4 (visited), 5. Go to 5
5	5	Neighbours 3 (visited), 6, 7. Go to 6
6	6	Neighbours 4, 5 (visited), 8. Go to 8
7	8	Neighbours 6 (visited), 7. Go to 7
8	7	Neighbours 5, 8 visited. Backtrack to 1; nothing left

DFS order: 1, 2, 4, 3, 5, 6, 8, 7. DFS tree edges: 1–2, 2–4, 4–3, 3–5, 5–6, 6–8, 8–7. The DFS tree is one long path, whereas the BFS tree is short and wide.

Problem 11. Perform DFS on the same graph from vertex 1 without recursion, using an explicit stack. Show the stack at each step.

Solution: Push the start vertex. Repeatedly **pop** a vertex; if it is already visited, skip it, otherwise visit it and push all its unvisited neighbours in **reverse** order (so the smallest neighbour ends up on top). The stack below is written from bottom to top.

Step	Pop	Action	Stack after step
1	1	Visit; push 3, 2	3, 2
2	2	Visit; push 4	3, 4
3	4	Visit; push 6, 3	3, 6, 3
4	3	Visit; push 5	3, 6, 5
5	5	Visit; push 7, 6	3, 6, 7, 6
6	6	Visit; push 8	3, 6, 7, 8
7	8	Visit; push 7	3, 6, 7, 7
8	7	Visit; nothing new	3, 6, 7
9	7	Already visited, skip	3, 6
10	6	Already visited, skip	3
11	3	Already visited, skip	empty

DFS order: 1, 2, 4, 3, 5, 6, 8, 7, the same as the recursive DFS of Problem 10. The stack may contain a vertex more than once, which is why the visited check is done when the vertex is popped.

Part D: Spanning Trees

Use the weighted graph of Fig. 3 for Problems 12 and 13.

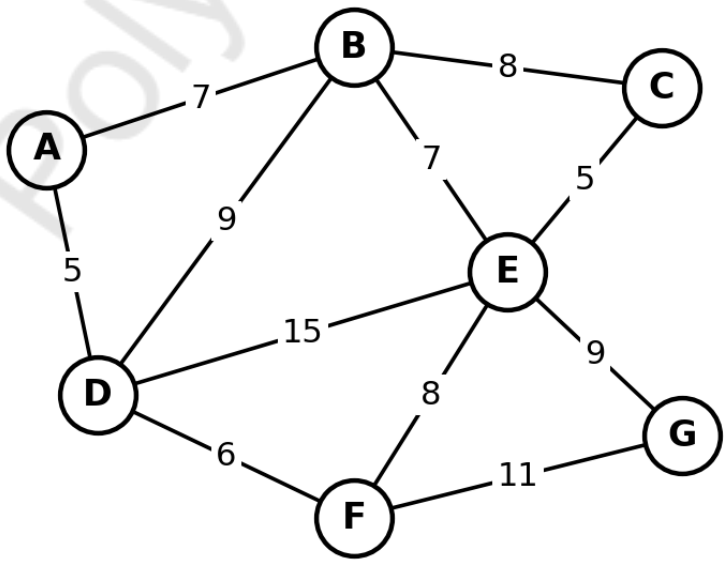

Fig. 3: Weighted graph for Problems 12 and 13

Problem 12. Find the minimum spanning tree of Fig. 3 using Kruskal's algorithm.

Solution: Sorted edges: A–D (5), C–E (5), D–F (6), A–B (7), B–E (7), B–C (8), E–F (8), B–D (9), E–G (9), F–G (11), D–E (15).

Step	Edge	Weight	Decision
1	A–D	5	Accept
2	C–E	5	Accept
3	D–F	6	Accept (component {A, D, F})
4	A–B	7	Accept (component {A, B, D, F})
5	B–E	7	Accept (joins {C, E} with {A, B, D, F})
6	B–C	8	Reject (B and C already connected)
7	E–F	8	Reject (E and F already connected)
8	B–D	9	Reject (cycle)
9	E–G	9	Accept (adds G); 6 edges chosen

Now n − 1 = 6 edges are selected, so stop. **MST edges: A–D, C–E, D–F, A–B, B–E, E–G. Total weight = 5 + 5 + 6 + 7 + 7 + 9 = 39.** The tree is shown in bold in Fig. 4.

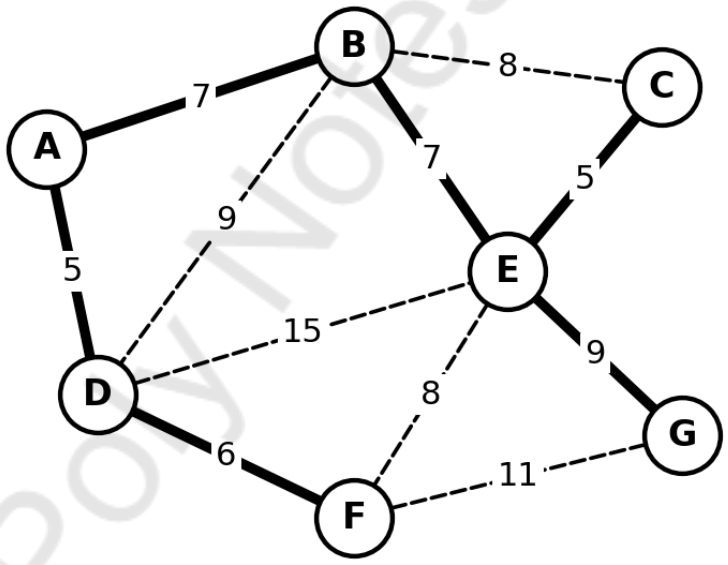

Fig. 4: Minimum spanning tree of Fig. 3 (bold edges), total weight 39

Problem 13. Find the minimum spanning tree of Fig. 3 using Prim's algorithm starting from vertex A.

Step	Tree vertices	Candidate edges	Edge chosen
1	A	AB (7), AD (5)	AD (5)
2	A, D	AB (7), BD (9), DE (15), DF (6)	DF (6)
3	A, D, F	AB (7), BD (9), DE (15), EF (8), FG (11)	AB (7)
4	A, B, D, F	BC (8), BE (7), DE (15), EF (8), FG (11)	BE (7)
5	A, B, D, E, F	BC (8), CE (5), EG (9), FG (11)	CE (5)
6	A, B, C, D, E, F	EG (9), FG (11)	EG (9)

Total weight = 5 + 6 + 7 + 7 + 5 + 9 = 39, the same tree as Kruskal's. The MST weight is always the same, even though the order in which edges are added differs between the two algorithms.

Problem 14. A company wants to connect five offices A, B, C, D, E with optical cable. The possible links and their costs (in lakh rupees) are A–B 12, A–C 5, A–D 9, B–C 6, B–E 4, C–D 8, C–E 10, D–E 3. Find the cheapest network that connects all offices, and its cost. How much is saved compared with laying every link?

Solution: Every office must be reachable from every other, and extra links only add cost, so the answer is a minimum spanning tree. Sort the links: D–E (3), B–E (4), A–C (5), B–C (6), C–D (8), A–D (9), C–E (10), A–B (12). Apply Kruskal: accept D–E, accept B–E (component {B, D, E}), accept A–C (component {A, C}), accept B–C (joins the two components). Now 4 = n − 1 links are chosen, so stop.

Cheapest network: D–E, B–E, A–C, B–C with cost 3 + 4 + 5 + 6 = 18 lakh rupees. Laying every link costs 12 + 5 + 9 + 6 + 4 + 8 + 10 + 3 = 57 lakh, so the saving is 57 − 18 = **39 lakh rupees**.

Part E: Shortest Path

Problem 15. Apply Dijkstra's algorithm to the directed graph of Fig. 5 with source S. Find the shortest distance to every vertex and the shortest path to T, using a predecessor array.

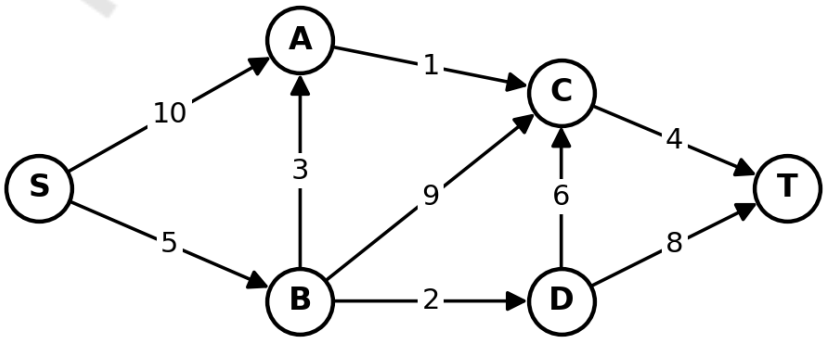

Fig. 5: Directed weighted graph for Problem 15

Solution: Set dist[S] = 0 and the others to ∞. Select the unvisited vertex with the smallest distance and relax its outgoing arcs.

Vertex selected	dist[A]	dist[B]	dist[C]	dist[D]	dist[T]
Initial	∞	∞	∞	∞	∞
S (0)	10	5	∞	∞	∞
B (5)	8	5	14	7	∞
D (7)	8	5	13	7	15
A (8)	8	5	9	7	15
C (9)	8	5	9	7	13
T (13)	8	5	9	7	13

Whenever a distance improves, the predecessor is updated. The final predecessor array is:

Vertex	A	B	C	D	T
Predecessor	B	S	A	B	C
Shortest distance	8	5	9	7	13

Tracing back from T: T ← C ← A ← B ← S, so the shortest path is **S → B → A → C → T with length 5 + 3 + 1 + 4 = 13**. Note how the route through B and A (8) beats the direct arc S → A (10), and the tentative distance of C improved twice (14, 13, then 9).

Problem 16. Find all-pairs shortest distances using the Floyd–Warshall algorithm for the directed graph with vertices 1 to 4 and arcs 1→2 = 3, 1→4 = 7, 2→1 = 8, 2→3 = 2, 3→1 = 5, 3→4 = 1, 4→1 = 2.

Solution: For k = 1, 2, 3, 4 in turn, set D[i][j] = min(D[i][j], D[i][k] + D[k][j]). Start with D0, which has 0 on the diagonal and ∞ where there is no arc.

D0 (initial):

	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

D1 (via vertex 1): D[2][4] = 8 + 7 = 15, D[3][2] = 5 + 3 = 8, D[4][2] = 2 + 3 = 5.

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	5	∞	0

D2 (via vertex 2): D[1][3] = 3 + 2 = 5, D[4][3] = 5 + 2 = 7.

	1	**2**	**3**	**4**
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

D3 (via vertex 3): D[1][4] = 5 + 1 = 6, D[2][1] = 2 + 5 = 7, D[2][4] = 2 + 1 = 3.

	1	**2**	**3**	**4**
1	0	3	5	6
2	7	0	2	3
3	5	8	0	1
4	2	5	7	0

D4 (via vertex 4): D[2][1] = 3 + 2 = 5, D[3][1] = 1 + 2 = 3, D[3][2] = 1 + 5 = 6. This is the final matrix:

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

Check: the shortest distance from 3 to 2 is 6 through 3 → 4 → 1 → 2 (1 + 2 + 3), and from 2 to 1 is 5 through 2 → 3 → 4 → 1 (2 + 1 + 2). Time complexity $O(V^3)$.

Problem 17. State whether each statement is true or false, with a reason.

1. **Dijkstra's algorithm works on undirected graphs with non-negative weights.** True. An undirected edge can be treated as two arcs of equal weight, and no negative weight is present.
2. **BFS always finds the shortest path in a weighted graph.** False. BFS minimises the number of edges, not the total weight. It is correct only when all edges have the same weight.
3. **Adding the same constant to every edge weight never changes the shortest path.** False. Paths with more edges are penalised more. Example: a direct edge of weight 5 and a two-edge route 2 + 2 = 4; after adding 2 to every edge the direct edge costs 7 and the two-edge route costs 8, so the shortest path changes.
4. **A graph with n vertices and n − 1 edges is always a tree.** False. It must also be connected. A triangle plus an isolated vertex has 4 vertices and 3 edges but is not a tree.
5. **Every spanning tree of a connected graph with n vertices has exactly n − 1 edges.** True. A spanning tree is connected and acyclic on all n vertices.

6. **If all edge weights are distinct, the minimum spanning tree is unique.** True. When weights are distinct, each rejected edge is the strict maximum of some cycle, so no alternative tree of equal weight exists.

Part F: Application Problem

Problem 18. Find a topological ordering of the DAG of Fig. 1 using the DFS finish-time method. Also apply the in-degree (Kahn) method and compare.

Solution: Run DFS (neighbours in increasing order) and record the order in which vertices **finish**, i.e. when all their successors have been fully explored. A topological order is the **reverse** of the finish order.

Event	Vertex	Note
Visit	1	Go to successor 2
Visit	2	Go to successor 4
Visit	4	Go to successor 6
Finish 1st	6	No successors
Finish 2nd	4	All successors done
Finish 3rd	2	All successors done. Back at 1, go to successor 3
Visit	3	Successor 4 visited; go to 5
Visit	5	Successor 6 visited
Finish 4th	5	Done
Finish 5th	3	Done
Finish 6th	1	Done

Finish order: 6, 4, 2, 5, 3, 1. **Reversed: 1, 3, 5, 2, 4, 6**, which is a valid topological order (every arc goes from an earlier to a later vertex: 1→2, 1→3, 2→4, 3→4, 3→5, 4→6, 5→6 all hold).

In-degree method: in-degrees are 1 = 0, 2 = 1, 3 = 1, 4 = 2, 5 = 1, 6 = 2. Remove 1 (then 2 and 3 become 0), remove 2 (4 drops to 1), remove 3 (4 and 5 become 0), remove 4 (6 drops to 1), remove 5 (6 becomes 0), remove 6. This gives **1, 2, 3, 4, 5, 6**. Both orders are valid: a DAG usually has more than one topological order. The longest path (1 → 2 → 4 → 6) has 3 arcs, so with unlimited parallel workers the tasks need a minimum of 4 stages.

Part G: Programming Problems in C

Problem 19. Write a complete C program that reads an undirected graph (n vertices, e edges), builds its adjacency list and prints the list of every vertex.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20

struct node {
    int vertex;
    struct node *next;
};
struct node *adj[MAX];            /* all heads are NULL initially */

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

int main()
{
    int n, e, i, u, v;
    struct node *p;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &n, &e);
    for (i = 0; i < e; i++) {
        scanf("%d %d", &u, &v);
        addEdge(u, v);            /* undirected: add both ways */
        addEdge(v, u);
    }
    for (i = 0; i < n; i++) {
        printf("%d:", i);
        for (p = adj[i]; p != NULL; p = p->next)
            printf(" %d", p->vertex);
        printf("\n");
    }
    return 0;
}
```

Vertices are numbered 0 to n − 1. Because each new node is inserted at the head, every list shows the neighbours in the reverse of the order in which the edges were entered.

Problem 20. Using the adjacency list of Problem 19, write C functions for BFS and DFS.

```
int visited[MAX];

void bfs(int start)
{
    int queue[MAX], front = 0, rear = -1, v;
    int seen[MAX] = {0};
    struct node *p;
    seen[start] = 1;
    queue[++rear] = start;
    while (front <= rear) {
        v = queue[front++];
        printf("%d ", v);
        for (p = adj[v]; p != NULL; p = p->next)
            if (!seen[p->vertex]) {
                seen[p->vertex] = 1;
                queue[++rear] = p->vertex;
            }
    }
}

void dfs(int v)
{
    struct node *p;
    visited[v] = 1;
    printf("%d ", v);
    for (p = adj[v]; p != NULL; p = p->next)
        if (!visited[p->vertex])
            dfs(p->vertex);
}
```

Both functions run in O(V + E) time because every vertex is processed once and every list node is examined once. BFS marks a vertex as seen when it is inserted into the queue, so no vertex enters the queue twice.

Problem 21. Write a C function for Kruskal's algorithm. The edges are stored in an array of structures and cycles are detected with a parent array.

```
struct edge { int u, v, w; };
int parent[MAX];

int find(int x)
{
    while (parent[x] != x)
        x = parent[x];
    return x;
}

int kruskal(struct edge e[], int n, int m)   /* n vertices, m edges */
{
    int i, j, a, b, count = 0, total = 0;
    struct edge t;
    for (i = 0; i < n; i++)
        parent[i] = i;                       /* each vertex alone */
    for (i = 0; i < m - 1; i++)              /* sort by weight    */
        for (j = 0; j < m - 1 - i; j++)
            if (e[j].w > e[j + 1].w) {
                t = e[j];  e[j] = e[j + 1];  e[j + 1] = t;
            }
    for (i = 0; i < m && count < n - 1; i++) {
        a = find(e[i].u);
        b = find(e[i].v);
        if (a != b) {                        /* different sets */
            parent[a] = b;                   /* union */
            printf("%d - %d : %d\n", e[i].u, e[i].v, e[i].w);
            total += e[i].w;
            count++;
        }
    }
    return total;                            /* MST weight */
}
```

Two vertices are in the same component exactly when find() returns the same root, so an edge with a == b would form a cycle and is skipped. The bubble sort makes this version $O(E^2)$; using qsort() gives the standard O(E log E).

Problem 22. Write a C function for Warshall's algorithm to compute the path (reachability) matrix of a directed graph from its adjacency matrix.

```
void warshall(int p[][MAX], int n)   /* p = adjacency matrix */
{
    int i, j, k;
    for (k = 0; k < n; k++)          /* k must be outermost */
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                p[i][j] = p[i][j] || (p[i][k] && p[k][j]);
}
```

After the function returns, p[i][j] = 1 exactly when there is a path from i to j. The loop over the intermediate vertex k must be the **outermost** loop, otherwise the result can be wrong. Time complexity is $O(V^3)$.

Quick Revision: Algorithms at a Glance

Algorithm	Idea in one line	Time
BFS	Queue; visit level by level; mark a vertex when it is inserted	O(V + E)
DFS	Stack or recursion; go deep first, then backtrack	O(V + E)
Kruskal	Sort edges; add an edge only if it joins two different components	O(E log E)
Prim	Grow one tree by the cheapest edge leaving it	$O(V^2)$
Dijkstra	Pick the nearest unvisited vertex, relax its edges (no negative weights)	$O(V^2)$
Bellman–Ford	Relax all edges V − 1 times; a V-th improvement means a negative cycle	O(VE)
Floyd–Warshall	Try every vertex k as an intermediate vertex for all pairs	$O(V^3)$
Warshall	Same idea using OR and AND to find reachability	$O(V^3)$
Topological sort	Repeatedly remove in-degree 0 vertices, or reverse the DFS finish order	O(V + E)

Practice Problems with Answers

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

No.	Problem	Answer
1	A graph has 9 vertices and 20 edges. How many edges does its complement have?	16
2	Maximum number of edges in a disconnected simple graph with 7 vertices.	15
3	Sum of the degrees of all vertices of a tree with 10 vertices.	18
4	Maximum number of arcs in a directed acyclic graph with 5 vertices.	10
5	A directed graph has 40 arcs. How many nodes does its adjacency list contain?	40
6	Time complexity of BFS when the graph is stored as an adjacency matrix.	$O(V^2)$
7	Number of spanning trees of K3.	3
8	Time complexity of Dijkstra's algorithm using a binary heap.	O((V + E) log V)
9	Number of minimum spanning trees when all edge weights are distinct.	1
10	Time complexity of the Bellman–Ford algorithm.	O(VE)