DATA STRUCTURE

Chapter 7: Graphs

Problems and Solutions (Exam Oriented): Set 4

This is the fourth set of solved problems for the chapter. It concentrates on question types that were not covered in Sets 1 to 3: testing whether a graph is a tree, leaves of a tree, tournaments, the inverse adjacency list, the path matrix from powers of the adjacency matrix, traversal on a graph given as a matrix, BFS parent arrays, DFS discovery and finish times with arc classification, Prim's algorithm with a key array, counting and proving facts about minimum spanning trees, shortest paths in a DAG, negative-cycle detection, graph colouring, and more C programs. It ends with comparison tables and practice problems with answers.

Part A: Properties of Graphs

Problem 1. Decide whether each graph is a tree. (a) 6 vertices with edges 1–2, 2–3, 3–1, 4–5, 5–6. (b) 6 vertices with edges 1–2, 1–3, 3–4, 3–5, 5–6.

Solution: An undirected graph with n vertices is a **tree** exactly when it is **connected and has n − 1 edges** (equivalently, connected and acyclic).

- (a) There are 5 edges = n − 1, but the graph is **not connected**: it consists of the triangle {1, 2, 3} and the path {4, 5, 6}. The triangle is a cycle. So it is **not a tree**. Having n − 1 edges alone is not enough.
- (b) There are 5 edges = n − 1 and every vertex can be reached from vertex 1 (1–2, 1–3, 3–4, 3–5, 5–6), so the graph is connected. Hence it **is a tree**.

Problem 2. A tree has 2 vertices of degree 4, 1 vertex of degree 3, and all other vertices are leaves (degree 1). Find the number of leaves and the total number of vertices.

Solution: Let L be the number of leaves. Then the number of vertices is n = 3 + L and the number of edges is n − 1 = L + 2. The sum of the degrees is twice the number of edges: 2(4) + 3 + L × 1 = 2(L + 2). So 11 + L = 2L + 4, which gives **L = 7**. The total number of vertices is 3 + 7 = **10**. Check: 8 + 3 + 7 = 18 = 2 × 9 edges.

Problem 3. A tournament is a directed graph in which there is exactly one arc between every pair of vertices. For a tournament on 8 vertices find (a) the number of arcs, (b) the sum of all out-degrees, and (c) the in-degree of a vertex whose out-degree is 7.

Solution: (a) There is one arc for each pair of vertices, so the number of arcs is 8 × 7 / 2 = **28**. (b) Every arc leaves exactly one vertex, so the sum of the out-degrees is also **28**. (c) A

vertex with out-degree 7 has an arc to each of the other 7 vertices, so no arc can enter it: its in-degree is **0** (it is a source).

Part B: Representation of Graphs

Problem 4. An undirected graph has the adjacency lists 1: 2, 5; 2: 1, 3, 4; 3: 2, 4; 4: 2, 3, 5; 5: 1, 4. Write its adjacency matrix, list the edges, and name a cycle.

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

Solution: Put 1 in row i, column j for every j in the list of i. The matrix is symmetric, as it must be for an undirected graph. The edges are **1–2, 1–5, 2–3, 2–4, 3–4 and 4–5** (6 edges). The degrees (list lengths or row sums) are 2, 3, 2, 3, 2, and 2 + 3 + 2 + 3 + 2 = 12 = 2 × 6. A cycle is **1** → **2** → **4** → **5** → **1** (length 4); another is 2 → 3 → 4 → 2 (length 3).

Problem 5. A directed graph has vertices 1 to 4 and arcs 1→2, 2→3, 3→2, 3→4. Find the path matrix P = A ∨ A² ∨ A³ ∨ A⁴ using Boolean powers of the adjacency matrix A.

Solution: Entry (i, j) of A^k is 1 if there is a walk of length k from i to j. Row i of A^(k+1) is the OR of those rows of A that row i of A^k points to.

A:

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

A² (walks of length 2):

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

A^3 (walks of length 3):

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

A^4 (walks of length 4) is the same as A^2. The OR of A, A^2, A^3 and A^4 gives the path matrix:

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

Vertices 1, 2 and 3 can each reach 2, 3 and 4 (vertices 2 and 3 reach themselves through the cycle $2 \to 3 \to 2$), and vertex 4 reaches nothing. Only powers up to n = 4 are needed, because any path in a graph with n vertices has length at most n. Warshall's algorithm gives the same matrix faster, in $O(V^3)$ instead of $O(V^4)$.

Problem 6. A directed graph has the adjacency lists 1: 2, 3; 2: 3; 3: 4; 4: 2. Find the in-degree of every vertex from the lists and construct the inverse adjacency list.

Solution: In an adjacency list the out-degree of a vertex is the length of its list, but the in-degree can be found only by scanning **all** the lists and counting how often the vertex appears. In the **inverse adjacency list**, the list of a vertex contains its predecessors (the vertices with an arc into it), so its length is the in-degree.

Vertex	Adjacency list (successors)	Inverse list (predecessors)	In-degree
1	2, 3	NULL	0
2	3	1, 4	2
3	4	1, 2	2
4	2	3	1

The in-degrees add up to 5, which equals the number of arcs ($1\to2$, $1\to3$, $2\to3$, $3\to4$, $4\to2$). Computing all in-degrees from the ordinary lists takes $O(V + E)$ time.

Part C: Graph Traversal

Problem 7. The adjacency matrix of an undirected graph with vertices 1 to 6 is given. Perform BFS and DFS starting from vertex 6, scanning each row from left to right.

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

Solution: BFS: insert 6. At each step remove the front vertex, scan its row and insert the unvisited vertices whose entry is 1.

Step	**Vertex removed**	**Unvisited 1s in its row**	**Queue after step**
1	6	4, 5	4, 5
2	4	2, 3	5, 2, 3
3	5	none	2, 3
4	2	1	3, 1
5	3	none	1
6	1	none	empty

BFS order: 6, 4, 5, 2, 3, 1.

DFS: visit 6; its row has its first unvisited 1 at column 4, so go to 4. From 4 go to 2 (column 2 is the first unvisited 1). From 2 go to 1. From 1 the first unvisited neighbour is 3. From 3 go to 5. Vertex 5 has no unvisited neighbour (3 and 6 are done), so backtrack to the start. **DFS order: 6, 4, 2, 1, 3, 5.** Because every row must be scanned fully, both traversals cost $O(V^2)$ with a matrix.

Problem 8. For the graph of Problem 7, use the BFS from vertex 6 to fill a parent array and hence find a shortest path from 6 to 1. What is the distance from 6 to 3?

Solution: Whenever BFS inserts a vertex into the queue, record the vertex it was reached from as its parent.

Vertex	**6**	**4**	**5**	**2**	**3**	**1**
Parent	— (start)	6	6	4	4	2

Follow the parents back from vertex 1: 1 ← 2 ← 4 ← 6. So the shortest path is **6 → 4 → 2 → 1, of length 3**. Vertex 3 has parent 4 and 4 has parent 6, so the distance from 6 to 3 is **2** (path 6 → 4 → 3).

Problem 9. Perform DFS on the directed graph of Fig. 1 starting from vertex 1 (successors in increasing order). Record the discovery time d and finish time f of every vertex, classify every arc as tree, back, forward or cross, and say whether the graph has a cycle.

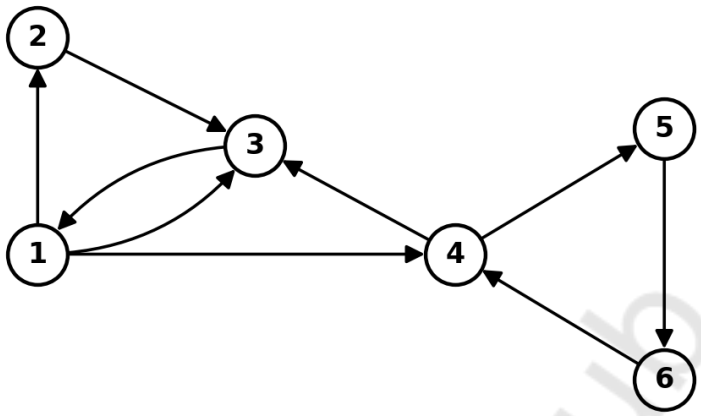

Fig. 1: Directed graph for Problem 9

Solution: Successor lists: 1: 2, 3, 4; 2: 3; 3: 1; 4: 3, 5; 5: 6; 6: 4. A clock ticks once at every discovery and every finish.

Vertex	**1**	**2**	**3**	**4**	**5**	**6**
Discovery d	1	2	3	6	7	8
Finish f	12	5	4	11	10	9

Arc	**Type**	**Reason**
1→2, 2→3, 1→4, 4→5, 5→6	Tree	Used by DFS to discover a new vertex
1→3	Forward	3 is a descendant of 1 that was already finished
3→1	Back	1 is an ancestor of 3 that is still active
6→4	Back	4 is an ancestor of 6 that is still active
4→3	Cross	3 finished (f = 4) before 4 was discovered (d = 6); neither is an ancestor of the other

All 9 arcs are classified (5 tree, 1 forward, 2 back, 1 cross). **A directed graph has a cycle exactly when DFS finds a back edge.** Here there are two back edges, so cycles exist: 1 → 2 → 3 → 1 and 4 → 5 → 6 → 4. The graph is also **strongly connected**: DFS from 1 reaches every vertex, and every vertex can get back to 1 (3→1, 2→3→1, 4→3→1, 6→4→3→1, 5→6→4→3→1).

Part D: Spanning Trees

Problem 10. Find the minimum spanning tree of the graph in Fig. 2 using Prim's algorithm with a key array, starting from vertex 1. The cost matrix uses ∞ where there is no edge.

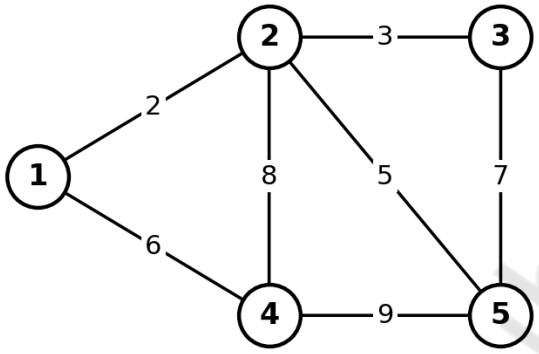

Fig. 2: Weighted graph for Problem 10

	1	2	3	4	5
1	0	2	∞	6	∞
2	2	0	3	8	5
3	∞	3	0	∞	7
4	6	8	∞	0	9
5	∞	5	7	9	0

Solution: Keep key[v], the cheapest known edge connecting v to the tree, and parent[v], the tree vertex of that edge. Repeatedly add the outside vertex with the smallest key and update the keys of its neighbours. A dash means the vertex is already in the tree.

Vertex added	key[2]	key[3]	key[4]	key[5]	Edge added
1 (start)	2	∞	6	∞	—
2	—	3	6	5	1–2 (2)
3	—	—	6	5	2–3 (3)
5	—	—	6	—	2–5 (5)
4	—	—	—	—	1–4 (6)

Parents: parent[2] = 1, parent[3] = 2, parent[4] = 1, parent[5] = 2. **MST edges: 1–2, 2–3, 2–5, 1–4. Total weight = 2 + 3 + 5 + 6 = 16.** Check with Kruskal's order (2, 3, 5, 6 accepted; 3–5 (7), 2–4 (8) and 4–5 (9) all close cycles): the same tree and the same weight. With a key array and a matrix, Prim's algorithm runs in $O(V^2)$.

Problem 11. All edges of a connected graph have the same weight. How many minimum spanning trees does it have if the graph is (a) a triangle, (b) a cycle of 4 vertices, (c) the complete graph K4?

Solution: If all weights are equal, every spanning tree has the same total weight (n − 1 times that weight), so **every spanning tree is a minimum spanning tree**. So count the

spanning trees. (a) A triangle has 3 edges and a spanning tree keeps any 2 of them: **3** trees. (b) A 4-cycle has 4 edges, and removing any one edge gives a spanning tree: **4** trees. (c) By Cayley's formula K4 has 4^2 = **16** spanning trees. This shows that an MST need not be unique when weights are repeated.

Problem 12. Prove that if all edge weights of a connected graph are distinct, the edge of smallest weight belongs to every minimum spanning tree.

Solution: Let e = (u, v) be the unique lightest edge and suppose some MST T does not contain e. Because T is a spanning tree, adding e to T creates exactly one cycle, and this cycle contains some other edge f of T. All weights are distinct and e is the lightest, so w(e) < w(f). Remove f from T + e: the result T′ = T + e − f is again a spanning tree (the cycle is broken and the graph stays connected), and its weight is w(T) − w(f) + w(e), which is **less than** w(T). This contradicts the fact that T is a minimum spanning tree. Hence every MST contains e. ■

Part E: Shortest Path

Problem 13. The road network of Fig. 3 joins six towns; the numbers are distances in km. Use Dijkstra's algorithm to find the shortest distance from P to every town and the shortest route from P to U.

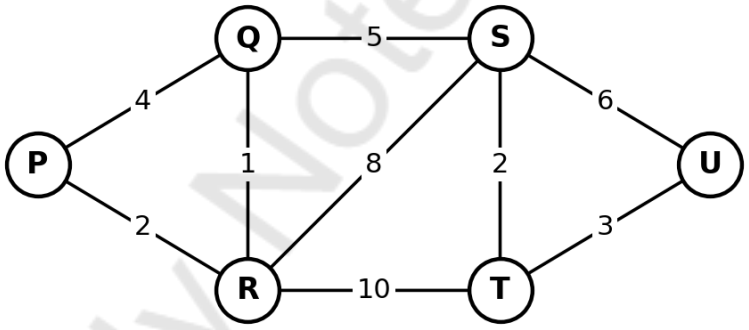

Fig. 3: Road network for Problem 13

Solution: Start with dist[P] = 0 and all others ∞. Select the nearest unvisited town and relax its roads.

Town selected	dist[Q]	dist[R]	dist[S]	dist[T]	dist[U]
Initial	∞	∞	∞	∞	∞
P (0)	4	2	∞	∞	∞
R (2)	3	2	10	12	∞
Q (3)	3	2	8	12	∞
S (8)	3	2	8	10	14
T (10)	3	2	8	10	13
U (13)	3	2	8	10	13

Town	Shortest distance	Route
Q	3	P → R → Q
R	2	P → R
S	8	P → R → Q → S
T	10	P → R → Q → S → T
U	13	P → R → Q → S → T → U

The shortest route from P to U is **P → R → Q → S → T → U, 13 km** (2 + 1 + 5 + 2 + 3). The direct road P–Q (4 km) is not used, because going through R makes Q reachable in 3 km, and reaching U through S (8 + 6 = 14) is longer than through T (10 + 3 = 13).

Problem 14. Find the shortest distance from S to every vertex of the weighted DAG of Fig. 4 by processing the vertices in topological order. Note that one arc has a negative weight.

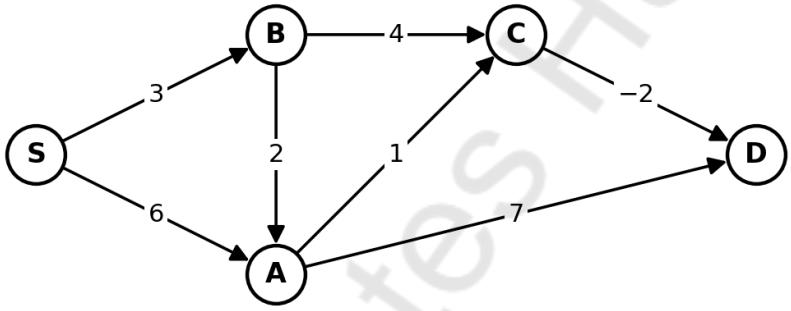

Fig. 4: Weighted DAG for Problem 14

Solution: A topological order of the DAG is S, B, A, C, D. Process the vertices in this order and relax the outgoing arcs of each. Since no cycle exists, a vertex's distance is final when its turn comes, so even a negative weight causes no trouble.

Vertex processed	dist[B]	dist[A]	dist[C]	dist[D]
Initial (dist[S] = 0)	∞	∞	∞	∞
S	3	6	∞	∞
B	3	5	7	∞
A	3	5	6	12
C	3	5	6	4
D	3	5	6	4

Final distances: B = 3, A = 5 (S → B → A), C = 6 (S → B → A → C), D = 4 (S → B → A → C → D, using the arc C → D of weight −2). The method takes only O(V + E) time, faster than Dijkstra, and it works with negative weights because the graph is acyclic.

Problem 15. Apply the Bellman–Ford algorithm to the directed graph with arcs (in this order) 1→2 = 1, 2→3 = −1, 3→4 = −1, 4→2 = −1 and source 1. Does the graph contain a negative cycle?

Solution: There are V = 4 vertices, so V − 1 = 3 passes should be enough if no negative cycle exists. In each pass relax all arcs in the given order. Values are shown at the end of each pass.

Pass	dist[2]	dist[3]	dist[4]	Change?
Initial	∞	∞	∞	—
1	−2	0	−1	Yes
2	−5	−3	−4	Yes
3	−8	−6	−7	Yes
4 (check)	arc 2→3 gives −9 < −6			Still improves

In pass 1, for example, 1→2 gives dist[2] = 1, then 2→3 gives 0, 3→4 gives −1 and 4→2 improves dist[2] to −2. After the required V − 1 = 3 passes the distances are still decreasing, and the extra V-th pass can still relax an arc. Therefore the graph **contains a negative cycle**, namely 2 → 3 → 4 → 2 of total weight −3, which is reachable from the source. No shortest path exists to vertices 2, 3 and 4, because every trip around the cycle makes the distance smaller.

Part F: Application Problems

Problem 16. A college must schedule the examinations of six subjects S1 to S6. Two subjects cannot be held in the same slot if some student takes both. Conflicting pairs: S1–S2, S1–S3, S2–S3, S3–S4, S4–S5, S4–S6, S5–S6. Use graph colouring to find a timetable with the fewest slots.

Solution: Draw a graph with a vertex for each subject and an edge for each conflict. Giving each vertex a colour (a slot) so that adjacent vertices get different colours is **graph colouring**. Colour the vertices in order, giving each the smallest colour not used by an adjacent earlier vertex.

Subject	Adjacent earlier subjects (colours)	Colour (slot)
S1	none	1
S2	S1 (1)	2
S3	S1 (1), S2 (2)	3
S4	S3 (3)	1
S5	S4 (1)	2
S6	S4 (1), S5 (2)	3

Timetable: slot 1 = {S1, S4}, slot 2 = {S2, S5}, slot 3 = {S3, S6}. None of the pairs inside a slot conflicts. Three slots are also the minimum, because S1, S2, S3 are mutually in conflict

(a triangle), so they need three different slots. The greedy method does not always give the minimum in general graphs, but here it does.

Problem 17. Four web pages A, B, C, D have the links A→B, A→C, B→C, C→A, D→A, D→C. Find the in-degree and out-degree of every page and say which page is linked to most.

Solution: Model the pages as vertices and the links as arcs. The in-degree counts how many pages link to a page.

Page	In-degree (linked from)	Out-degree (links to)
A	2 (C, D)	2 (B, C)
B	1 (A)	1 (C)
C	3 (A, B, D)	1 (A)
D	0	2 (A, C)

The total in-degree and the total out-degree are both 6, the number of links. **Page C has the highest in-degree (3)**, so it is the most linked page, and page D is a source that nobody links to. Search engines refine this idea (PageRank): a link from an important page counts for more than a link from an unimportant one.

Part G: Programming Problems in C

The structures node and adj[] of the adjacency-list representation (struct node with fields vertex and next) are the same as in the earlier sets.

Problem 18. Write a C function that prints the out-degree and in-degree of every vertex of a directed graph stored as adjacency lists.

Solution: The out-degree is the length of a list. To get the in-degrees, count how many times each vertex appears in any list.

```
void degrees(int n)
{
    int indeg[MAX] = {0};
    int i, outdeg;
    struct node *p;
    for (i = 0; i < n; i++) {
        outdeg = 0;
        for (p = adj[i]; p != NULL; p = p->next) {
            outdeg++;                  /* arc i -> p->vertex */
            indeg[p->vertex]++;
        }
        printf("Vertex %d: out = %d\n", i, outdeg);
    }
    for (i = 0; i < n; i++)
        printf("Vertex %d: in = %d\n", i, indeg[i]);
}
```

Every list node is visited once, so the time complexity is O(V + E).

Problem 19. Write a recursive C function that returns 1 if there is a path from vertex u to vertex v in a graph stored as an adjacency matrix, and 0 otherwise.

```
int a[MAX][MAX], visited[MAX];

int hasPath(int u, int v, int n)
{
    int w;
    if (u == v)
        return 1;
    visited[u] = 1;
    for (w = 0; w < n; w++)
        if (a[u][w] == 1 && !visited[w] && hasPath(w, v, n))
            return 1;
    return 0;
}
```

Before every call the array visited[] must be cleared to 0 (for example with memset). The function is a DFS that stops as soon as v is reached; the worst-case time is $O(V^2)$.

Problem 20. Write a C function for topological sorting of a directed graph stored as an adjacency matrix using the in-degree method. It should also report a cycle.

```
void topoSort(int n)
{
    int indeg[MAX] = {0}, queue[MAX];
    int front = 0, rear = -1, count = 0, i, j, v;
    for (i = 0; i < n; i++)          /* in-degree = column sum */
        for (j = 0; j < n; j++)
            indeg[j] += a[i][j];
    for (i = 0; i < n; i++)
        if (indeg[i] == 0)
            queue[++rear] = i;
    while (front <= rear) {
        v = queue[front++];
        printf("%d ", v);
        count++;
        for (j = 0; j < n; j++)
            if (a[v][j] == 1 && --indeg[j] == 0)
                queue[++rear] = j;
    }
    if (count < n)
        printf("\nGraph has a cycle");
}
```

A vertex enters the queue only when all its predecessors have been printed. If fewer than n vertices are printed, the remaining vertices lie on or behind a cycle and no topological order exists. Time complexity is $O(V^2)$ with a matrix.

Problem 21. Write a C function for the Bellman–Ford algorithm using an array of edges. It should return 0 if a negative cycle is reachable from the source.

```
#define INF 9999
struct edge { int u, v, w; };

int bellmanFord(struct edge e[], int n, int m, int src, int dist[])
{
    int i, j;
    for (i = 0; i < n; i++)
        dist[i] = INF;
    dist[src] = 0;
    for (i = 0; i < n - 1; i++)            /* n - 1 passes */
        for (j = 0; j < m; j++)
            if (dist[e[j].u] != INF &&
                dist[e[j].u] + e[j].w < dist[e[j].v])
                dist[e[j].v] = dist[e[j].u] + e[j].w;
    for (j = 0; j < m; j++)                /* extra pass: check */
        if (dist[e[j].u] != INF &&
            dist[e[j].u] + e[j].w < dist[e[j].v])
            return 0;                      /* negative cycle */
    return 1;                              /* dist[] is valid */
}
```

The test dist[u] != INF stops the program from relaxing an arc out of an unreachable vertex. The nested loops give a time complexity of O(VE).

Quick Revision: Comparison Tables

Point	Kruskal	Prim
Approach	Picks the cheapest edge of the whole graph that does not form a cycle	Grows one tree by the cheapest edge leaving it
Works on	Edge list (sorted)	Adjacency matrix or list with keys
Intermediate result	A forest that may be disconnected	Always a single connected tree
Time	O(E log E)	$O(V^2)$ or O(E log V)
Best for	Sparse graphs	Dense graphs

Point	Dijkstra	Bellman–Ford	Floyd–Warshall
Problem solved	Single source	Single source	All pairs
Negative weights	Not allowed	Allowed	Allowed (no negative cycle)
Negative cycle	Not detected	Detected	Shows a negative diagonal entry
Time	$O(V^2)$ or O((V + E) log V)	O(VE)	$O(V^3)$
Technique	Greedy	Repeated relaxation	Dynamic programming

Arc type in DFS of a directed graph	Meaning
Tree arc	Leads to a newly discovered vertex
Back arc	Leads to an ancestor still being explored; signals a cycle
Forward arc	Leads to an already finished descendant
Cross arc	Leads to a finished vertex in another branch

Practice Problems with Answers

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

No.	Problem	Answer
1	A tree has 3 vertices of degree 3 and all other vertices are leaves. How many leaves does it have?	5
2	Number of arcs in a tournament on 10 vertices.	45
3	Number of minimum spanning trees of a 5-cycle whose edges all have equal weight.	5
4	Minimum number of colours needed to colour the complete graph K5.	5
5	Maximum number of passes Bellman–Ford needs on a graph with 12 vertices (no negative cycle).	11
6	Time complexity of shortest paths in a DAG using topological order.	O(V + E)
7	The sum of the in-degrees of a directed graph is 25. How many arcs does it have?	25
8	In DFS of a directed graph, an arc to a vertex that is still on the recursion stack is called a ___ arc.	Back
9	Maximum number of edges in a bipartite graph with 10 vertices.	25
10	A connected graph has 12 vertices and 11 edges. Is it a tree?	Yes