ABHYASMITRA.IN

COMPREHENSIVE UNIVERSITY STUDY NOTES

DATA STRUCTURES

Unit V — Non-Linear Data Structures: Graphs, MST, Shortest Paths & Topological Sort

---

Topics Covered in this Unit

Graph Fundamentals & Terminology: Vertices, Edges, Directed vs. Undirected, Handshaking Lemma Proof, Connectivity

Storage Representations: Adjacency Matrix vs. Adjacency List (Density, Space Complexity, Query Efficiency Trade-offs)

Graph Traversals: Depth-First Search (DFS Edge Classification: Tree, Back, Forward, Cross) & Breadth-First Search (BFS)

Minimum Spanning Tree (MST): Spanning Tree Properties, Cut Property, Prim's Greedy Algorithm & Kruskal's with Disjoint-Set

Single Source Shortest Path: Dijkstra's Algorithm, Greedy Edge Relaxation, Trace Table, Negative Edge Limitations

All-Pairs Shortest Path: Floyd-Warshall Dynamic Programming Matrix Sequence $D^{(k)}$, Recurrence Relation & Negative Cycle Check

Topological Ordering / Sorting: Directed Acyclic Graphs (DAG), Kahn's In-Degree BFS Algorithm & DFS Stack-based Algorithm

High-Yield University Exam Question Bank with Detailed Model Answers & Unit Summary

Compiled in a structured, easy-to-understand, textbook style format
Specially curated for B.Tech / B.E. / BCA / MCA / B.Sc Computer Science & Engineering Students

COURSE SYLLABUS & MAPPING

[UNIT V SYLLABUS — PRESCRIBED UNIVERSITY CURRICULUM]

This section contains the official syllabus for Unit V. Verify with your university curriculum mapping.

Unit V Syllabus Breakdown:

Unit V Non-linear Data Structure: Graph: Graph: Introduction of graph, storage representation, Adjacency matrix, adjacency list, DFS, BFS, Minimum spanning Tree - Prims and Kruskal Algorithms, Dijkstra's Single source shortest path, All pairs shortest paths- FlyodWarshall Algorithm, Topological ordering.

• Course Code: _____ • Semester / Year: _____

TABLE OF CONTENTS

5.1 Graph Fundamentals, Terminology & Handshaking Lemma

Formal Definition: Graph G = (V, E)

A Graph G is a non-linear data structure defined as an ordered pair G = (V, E), where:

- *V is a non-empty finite set of elements called Vertices (or Nodes).*
- *E is a set of pairs of vertices called Edges (or Arcs) representing relationships or connections between vertices.*

For an Undirected Graph, edges are unordered pairs {u, v}. For a Directed Graph (Digraph), edges are ordered pairs (u, v) directed from u to v.

Key Graph Terminology

- **1. Degree of a Vertex:** In undirected graphs, the number of edges incident on vertex v. In digraphs: In-Degree (number of incoming edges) and Out-Degree (number of outgoing edges). Total degree = in-degree + out-degree.
- **2. Isolated & Pendant Vertices:** An Isolated Vertex has degree 0 (no incident edges). A Pendant Vertex has degree 1.
- **3. Complete Graph (K_n):** A simple undirected graph of n vertices where every pair of distinct vertices is connected by an edge. Total edges |E| = n(n - 1) / 2.
- **4. Path, Cycle & Connectivity:** A Path is an alternating sequence of vertices and edges with distinct vertices. A Cycle is a closed path (start and end vertices are identical). A graph is Connected if there is a path between every pair of vertices.
- **5. Strongly vs. Weakly Connected Digraphs:** A digraph is Strongly Connected if there exists a directed path between every pair of vertices in both directions. It is Weakly Connected if replacing directed edges with undirected edges yields a connected graph.

Handshaking Lemma & Proof

Theorem: In any undirected graph G = (V, E), the sum of degrees of all vertices is equal to twice the number of edges:

$$\sum_{v \in V} deg(v) = 2 \cdot |E|$$

Proof:

Every single edge e = {u, v} connects two vertices u and v, contributing exactly +1 to deg(u) and +1 to deg(v).

Thus, every edge is counted twice when summing the degrees over all vertices.

Therefore: $\sum deg(v) = 2 \cdot |E|$ [Q.E.D.]

Corollary: The number of vertices with ODD degree in any graph is ALWAYS EVEN.

5.2 Storage Representations: Adjacency Matrix vs. Adjacency List

1. Adjacency Matrix Representation

An Adjacency Matrix for a graph with n vertices is a 2D square matrix A of dimension n × n, where A[i][j] = 1 (or edge weight w) if there is an edge from vertex i to vertex j, and 0 (or ∞) otherwise.

- • **Undirected Graph:** The adjacency matrix is always SYMMETRIC across the main diagonal (A[i][j] = A[j][i]).
- • **Directed Graph:** Row sum of row i gives Out-Degree of vertex i; Column sum of column j gives In-Degree of vertex j.
- • **Complexity:** Space: $O(|V|^2)$; Edge lookup (u, v): O(1); Vertex neighbors search: O(|V|).

2. Adjacency List Representation

An array (or vector) of Linked Lists of size |V|, where the i-th linked list stores all vertices adjacent to vertex i.

- • **Complexity:** Space: O(|V| + |E|) for Digraphs; O(|V| + 2|E|) for Undirected graphs. Highly space-efficient for sparse graphs ($|E| \ll |V|^2$).

STORAGE REPRESENTATIONS VISUALIZATION

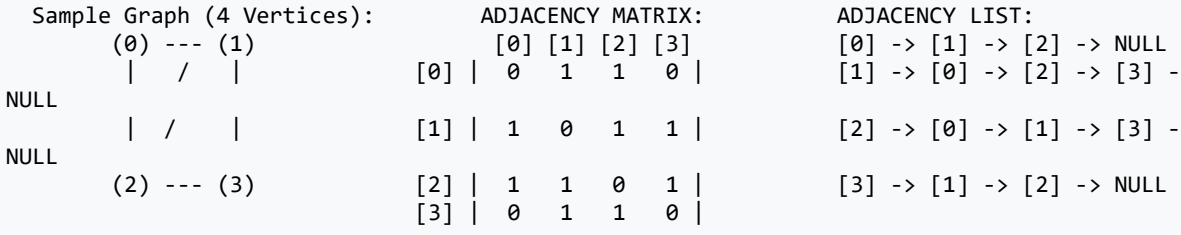

Figure: Comparison of Adjacency Matrix (dense matrix) and Adjacency List (linked lists)

Master Comparison Table: Adjacency Matrix vs. Adjacency List

Evaluation Parameter	Adjacency Matrix	Adjacency List
Memory Space Complexity	$O(\|V\|^2)$ — Inefficient for sparse graphs	$O(\|V\| + \|E\|)$ — Optimal for sparse graphs
Edge Existence Query (u, v)	O(1) (Direct index lookup A[u][v])	O(deg(u)) (Linear scan of list u)
Find All Neighbors of Vertex	$O(\|V\|)$ (Scan entire row of size $\|V\|$)	O(deg(u)) (Traverse linked list u)
Add a New Vertex	$O(\|V\|^2)$ (Must resize n × n matrix)	O(1) (Append to array of lists)
Suitability	Dense Graphs where $\|E\| \approx \|V\|^2$	Sparse Graphs where $\|E\| \ll \|V\|^2$ (Most real networks)

5.3 Graph Traversals: Depth-First & Breadth-First Search

1. Depth-First Search (DFS)

DFS explores as deep as possible along each branch before backtracking, utilizing a LIFO Stack (or system recursion stack) and a boolean visited[] array.

- **Edge Classification in DFS:**
  - • **Tree Edges:** Edges traversed to discover an unvisited vertex for the first time.
  - • **Back Edges:** Edges connecting to an ancestor in the DFS tree. CRUCIAL: The presence of a back edge indicates a CYCLE in the graph!
  - • **Forward Edges:** Nontree edges connecting to a descendant in the DFS tree.
  - • **Cross Edges:** Edges connecting vertices in different subtrees with no ancestor-descendant relation.
- **Time & Space:** Time: $O(|V| + |E|)$ with adjacency list; Space: $O(|V|)$ recursion stack.

2. Breadth-First Search (BFS)

BFS visits vertices level by level radiating outwards from a start vertex, utilizing a FIFO Queue and visited[] array.

- **Shortest Path Property:** BFS is guaranteed to find the SHORTEST path (minimum number of edges) from the source to all reachable vertices in an unweighted graph.
- **Time & Space:** Time: $O(|V| + |E|)$ with adjacency list; Space: $O(|V|)$ queue buffer.

5.4 Minimum Spanning Tree (MST): Prim's & Kruskal's Algorithms

Definition: Minimum Spanning Tree (MST)

For a connected, edge-weighted, undirected graph G = (V, E), a Minimum Spanning Tree (MST) is a spanning subgraph T = (V, E') that connects all |V| vertices using exactly |V| - 1 edges without any cycles, such that the Total Edge Weight ∑ w(e) is MINIMIZED.

1. Prim's Algorithm (Vertex-Growing Greedy Approach)

Maintains a growing tree T starting from an arbitrary source vertex. At each step, finds the minimum-weight edge connecting a vertex in T to a vertex outside T, and adds that edge and vertex to T.

- **Time Complexity:** $O(|V|^2)$ with Adjacency Matrix; $O(|E| \log |V|)$ using Min-Heap / Priority Queue with Adjacency List. Best for dense graphs.

2. Kruskal's Algorithm (Edge-Growing Greedy Approach)

Sorts all edges in non-decreasing order of weights. Iteratively picks the smallest edge; if adding it does not create a cycle, it is included in the MST. Cycle detection is handled via Disjoint-Set / Union-Find (with Path Compression and Union by Rank).

- **Time Complexity:** Sorting edges: $O(|E| \log |E|) = O(|E| \log |V|)$. Union-Find operations: $O(|E| \cdot \alpha(|V|))$ where α is the inverse Ackermann function. Overall time: $O(|E| \log |V|)$. Best for sparse graphs.

Comparison: Prim's vs. Kruskal's Algorithm

Comparison Criteria	Prim's Algorithm	Kruskal's Algorithm
Core Strategy	Grows a single connected tree vertex by vertex.	Grows a forest of trees by selecting smallest global edges.
Graph Precondition	Must be connected graph.	Can work on disconnected components (forms Minimum Spanning Forest).
Cycle Detection	Implicitly avoided by picking edges to outside vertices.	Explicitly detected using Disjoint-Set (Union-Find) structure.
Time Complexity	$O(\|V\|^2)$ or $O(\|E\| \log \|V\|)$ with Min-Heap	$O(\|E\| \log \|E\|) = O(\|E\| \log \|V\|)$
Optimal Graph Type	Dense Graphs ($\|E\| \approx \|V\|^2$)	Sparse Graphs ($\|E\| \ll \|V\|^2$)

5.5 Shortest Path Algorithms: Dijkstra & Floyd-Warshall

1. Dijkstra's Single Source Shortest Path Algorithm

Dijkstra's Algorithm & Edge Relaxation

Dijkstra's algorithm computes the shortest path from a single source vertex s to all other vertices in a weighted graph with NON-NEGATIVE edge weights.

Relaxation Condition:

For directed edge (u, v) with weight w(u, v):

```
if (dist[u] + w(u, v) < dist[v]) {
   dist[v] = dist[u] + w(u, v);
   parent[v] = u;
}
```

NOTE / EXAM POINTER

CRITICAL LIMITATION OF DIJKSTRA'S ALGORITHM: Dijkstra's greedy choice assumes that once a vertex is marked visited (extracted from min-priority queue), its shortest distance is permanently finalized. This assumption FAILS on graphs with NEGATIVE edge weights! For graphs with negative weights, the Bellman-Ford algorithm must be used.

2. Floyd-Warshall All-Pairs Shortest Path Algorithm

Floyd-Warshall Dynamic Programming Recurrence

Computes shortest paths between ALL pairs of vertices (i, j) in a directed weighted graph.

Let D^(k)[i][j] be the shortest path from vertex i to vertex j using only intermediate vertices from the set {1, 2, ..., k}:

$$D^{(k)}[i][j] = \min \{ D^{(k-1)}[i][j],\ D^{(k-1)}[i][k] + D^{(k-1)}[k][j] \}$$

Base Case: D^(0)[i][j] = weight matrix W[i][j] (with D^(0)[i][i] = 0 and non-edges = ∞).

Time Complexity = $\Theta(|V|^3)$ via 3 nested loops. Space Complexity = $\Theta(|V|^2)$.

- **Negative Cycle Detection in Floyd-Warshall:** If after computing D^(n), any diagonal entry D^(n)[i][i] < 0, then a negative weight cycle exists reachable from vertex i.

5.6 Topological Ordering / Sorting in DAGs

Definition: Topological Sorting

A Topological Sort of a Directed Acyclic Graph (DAG) is a linear ordering of its vertices such that for every directed edge (u, v), vertex u appears BEFORE vertex v in the ordering.

• Condition: Graph MUST be a DAG (Directed Acyclic Graph). If a graph contains a cycle, no topological ordering exists!

1. Kahn's Algorithm (In-Degree BFS Approach)

1. Compute in-degree for all vertices. 2. Enqueue all vertices with in-degree == 0 into a Queue. 3. While Queue is not empty: Dequeue vertex u, append to topological order list. For each neighbor v of u, decrement in-degree[v] by 1. If in-degree[v] becomes 0, enqueue v. 4. Cycle Check: If count of visited vertices < |V|, the graph contains a cycle!

- **Time Complexity:** O(|V| + |E|), Space: O(|V|).

2. DFS-Based Topological Sort

Execute DFS traversal. When a vertex finishes exploring all its adjacent descendants (on finish time), PUSH it onto a Stack. When all vertices are processed, popping the stack yields the valid topological order.

5.7 High-Yield University Exam Question Bank

Part A: Short Answer Questions (2 to 5 Marks)

Q1. State Handshaking Lemma. Why is the number of vertices with odd degree always even? [3 Marks]

- **Answer:** Handshaking Lemma states $\sum \deg(v) = 2|E|$. Since $2|E|$ is always an even number, the total sum of degrees must be even. Summing degrees of even-degree vertices yields an even number. For the total sum to remain even, the sum of degrees of odd-degree vertices must also be even, which is mathematically only possible if the number of odd-degree vertices is EVEN.

Q2. Why does Dijkstra's algorithm fail for negative edge weights? [3 Marks]

- **Answer:** Dijkstra's algorithm relies on a greedy strategy where once a vertex is chosen with minimum tentative distance and marked visited, its distance is assumed final. A subsequent negative

weight edge can provide a shorter path to an already-finalized vertex, which Dijkstra's algorithm cannot revisit or correct.

Q3. What is Topological Sorting, and what is the necessary condition for its existence? [3 Marks]

- **Answer:** Topological sorting is a linear arrangement of vertices in a directed graph such that for every edge u -> v, u precedes v. The mandatory condition is that the graph MUST be a Directed Acyclic Graph (DAG). If a directed cycle exists (e.g., A -> B -> C -> A), no linear ordering can satisfy the precedence dependencies.

Part B: Long Answer Questions (10 to 15 Marks Outline Guides)

- **Q4. Compare Adjacency Matrix and Adjacency List with diagrams and time/space complexity for insertion, deletion, and search. [10 Marks]**
- **Q5. Explain Depth-First Search (DFS) and Breadth-First Search (BFS) with algorithms. Trace both on a 6-vertex graph. Discuss classification of edges in DFS. [12 Marks]**
- **Q6. Find the Minimum Spanning Tree (MST) of a weighted graph using: (a) Prim's Algorithm, (b) Kruskal's Algorithm with Union-Find. Compare their complexities. [14 Marks]**
- **Q7. Write Dijkstra's algorithm. Find shortest paths from source vertex A to all other vertices in a 5-vertex directed weighted graph showing the distance array at every step. [12 Marks]**
- **Q8. Explain Floyd-Warshall All-Pairs Shortest Path algorithm. Compute the sequence of matrices D^(0), D^(1), D^(2), D^(3) for a 3-vertex directed graph. How are negative cycles detected? [14 Marks]**
- **Q9. Write Kahn's algorithm for Topological Sort using in-degrees. Trace the algorithm on a Directed Acyclic Graph of 6 tasks. [10 Marks]**

5.8 Unit Summary & Key Takeaways

- Graphs represent many-to-many non-linear relationships G = (V, E); Handshaking lemma proves $\sum$ deg(v) = 2|E|.
- Adjacency Matrix offers O(1) edge queries at O($|V|^2$) space; Adjacency List optimizes memory to O(|V| + |E|) for sparse graphs.
- DFS (stack/recursion) explores depth-first and identifies cycles via back edges; BFS (queue) finds shortest paths in unweighted graphs.
- Minimum Spanning Trees connect all vertices with |V|-1 edges and minimum weight, solved via Prim's (vertex-greedy) and Kruskal's (edge-greedy with Union-Find).

- Dijkstra's greedy algorithm finds single-source shortest paths in non-negative weighted graphs in O((|V|+|E|) log |V|) time.
- Floyd-Warshall dynamic programming computes All-Pairs Shortest Paths in $\Theta(|V|^3)$ time via recurrence D^(k)[i,j] = min(D^(k-1)[i,j], D^(k-1)[i,k] + D^(k-1)[k,j]).
- Topological sort linearly orders DAG vertices via Kahn's in-degree BFS or DFS finish-time stack, resolving dependencies in task scheduling.