📘 **Mastering Graphs in C++ (FAANG & Codeforces Level)**

Chapter 1: Introduction to Graphs

Definition: A graph is a collection of nodes (vertices) and edges (connections) that can represent networks like social media, road maps, etc.

Types of Graphs:

- Directed vs Undirected
- Weighted vs Unweighted
- Cyclic vs Acyclic
- Connected vs Disconnected

Representations:

- **Adjacency Matrix**
- **Adjacency List** (most common in CP)

Applications:

- Pathfinding (Google Maps)
- Social networks (Facebook, LinkedIn)
- Recommendation engines (YouTube, Amazon)

🛠️Chapter 2: Graph Implementation in C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, m; // nodes and edges
    cin >> n >> m;
    vector<vector<int>> adj(n + 1);
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u); // remove if directed
    }
}
```

Edge Structure (Weighted Graph):

```
vector<pair<int, int>> adj[n]; // {node, weight}
```

⚙️Chapter 3: Basic Operations

1. **Add Edge**
2. **Remove Edge**
3. **Check Adjacency**
4. **DFS Traversal**
5. **BFS Traversal**

DFS Code:

```
void dfs(int node, vector<vector<int>>& adj, vector<int>& visited) {
    visited[node] = 1;
    for (int neighbor : adj[node]) {
        if (!visited[neighbor])
            dfs(neighbor, adj, visited);
    }
}
```

BFS Code:

```
void bfs(int start, vector<vector<int>>& adj) {
    vector<bool> visited(adj.size(), false);
    queue<int> q;
    q.push(start);
    visited[start] = true;

    while (!q.empty()) {
        int node = q.front();
        q.pop();

        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                q.push(neighbor);
                visited[neighbor] = true;
            }
        }
    }
}
```

📀 Chapter 4: Important Graph Algorithms

1. **Cycle Detection** (Directed + Undirected)
2. **Topological Sort** (DFS or Kahn's Algorithm)
3. **Dijkstra's Algorithm** (Shortest Path)
4. **Bellman-Ford Algorithm**
5. **Floyd-Warshall Algorithm**
6. **Prim's Algorithm** (MST)
7. **Kruskal's Algorithm** (MST)
8. **Disjoint Set Union (DSU)**
9. **Bridge and Articulation Points**
10. **Kosaraju's Algorithm** (SCC in Directed Graph)

📌 Chapter 5: Pro Tips and Strategies

- Use **Adjacency List** for large graphs to save space.
- Prefer **DFS for tree-like recursion**, **BFS for shortest unweighted path**.
- Watch out for **0-based vs 1-based indexing**.
- For shortest path in weighted graphs, use **priority queue** with Dijkstra.
- In competitive programming, modularize your code for graph input & DFS/BFS.

🧪 Chapter 6: Practice Exercises

🕰️ Basic Level

1. Represent a graph using an adjacency list.
2. Perform BFS and DFS on a connected graph.
3. Count connected components in an undirected graph.
4. Detect cycle in an undirected graph.

🕰️ Intermediate Level

1. Check if a graph is bipartite.
2. Topological sort using DFS.
3. Find the shortest path using Dijkstra's algorithm.
4. Implement Kruskal's algorithm.

🕰️ Advanced Level

1. Tarjan's algorithm for bridges.
2. Kosaraju's algorithm for SCC.
3. Detect a negative cycle using Bellman-Ford.
4. Count number of strongly connected components.

🔗Chapter 7: Solved Examples

Q1. Detect Cycle in Undirected Graph using DFS.\ **Q2.** Dijkstra's Algorithm with priority_queue.\ **Q3.** Topological Sort using Kahn's Algorithm.\ **Q4.** Disjoint Set with Path Compression and Union by Rank.\ **Q5.** Kosaraju's Algorithm for SCC.

Chapter 8: Quick Notes and Tricks

- **DFS with visited and recursion stack = cycle detection in Directed Graph**
- **DFS Tree vs Back Edge = Useful in detecting bridges and articulation points**
- **Reverse edges in Kosaraju's first pass**
- **Always initialize `` or DSU parent properly before each test case**
- **In undirected graph, avoid revisiting parent in DFS**

📎Chapter 9: LeetCode and Practice Links

- Graph Valid Tree
- Course Schedule (Topo Sort)
- Network Delay Time (Dijkstra)
- Redundant Connection (Cycle Detection)
- Number of Connected Components
- Critical Connections (Bridges)

Let me know when you're ready for the next topic: **Trees** ✨