Graph Patterns Playbook

High-Impact LeetCode Graph Templates

Updated October 4, 2025

Curated from Blind 75, NeetCode 150, and top company sets

Contents

1 🔗 Graph Fundamentals

Building clean adjacency representations unlocks flexible traversals and ensures you can plug in search patterns quickly.

1.1 Adjacency List Builder

Handles directed vs. undirected edges while keeping neighbors sorted for reproducibility. **Complexity:** Time $O(V + E)$ to build, Space $O(V + E)$.

```
from collections import defaultdict

def build_graph(n, edges, directed=False):
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)
        if not directed:
            graph[v].append(u)
    for node in graph:
        graph[node].sort()
    return graph
```

Example Problems:

- 733. Flood Fill (NeetCode 150)
- 323. Number of Connected Components in an Undirected Graph (NeetCode 150)
- 261. Graph Valid Tree (Top Meta/Ebay)
- 2492. Minimum Score of a Path Between Two Cities (Top Amazon)

1.2 Visited Bookkeeping Patterns

Choose between ‘set‘, ‘dict‘, or in-place marking depending on constraints. **Tip:** For dense integer labels, prefer boolean arrays; for large IDs, use hash sets. **Example Problems:**

- 200. Number of Islands (Blind 75, NeetCode 150)
- 133. Clone Graph (Blind 75)
- 886. Possible Bipartition (NeetCode 150)
- 207. Course Schedule (Blind 75, NeetCode 150)

2 Traversal Patterns

Depth-first and breadth-first search sit at the heart of graph reasoning. Master variants like multi-source BFS or iterative DFS for stack control.

2.1 Depth-First Search (Recursive & Iterative)

Recursion communicates intent; iterative variants avoid recursion limits and support custom stack seeding. **Complexity:** Time $O(V + E)$, Space $O(V)$.

```python
def dfs_recursive(graph, start):
    seen = set()

    def dfs(node):
        if node in seen:
            return
        seen.add(node)
        for nei in graph[node]:
            dfs(nei)

    dfs(start)
    return seen


def dfs_iterative(graph, start):
    stack, seen = [start], set()
    while stack:
        node = stack.pop()
        if node in seen:
            continue
        seen.add(node)
        for nei in graph[node]:
            if nei not in seen:
                stack.append(nei)
    return seen
```

Example Problems:

- 417. Pacific Atlantic Water Flow (NeetCode 150)
- 695. Max Area of Island (NeetCode 150)
- 399. Evaluate Division (Top Google)
- 684. Redundant Connection (NeetCode 150)

2.2 Breadth-First Search for Levels

Track layer depth to handle unweighted shortest path, level-order traversal, and minimum steps. **Complexity:** Time $O(V + E)$, Space $O(V)$.

```python
from collections import deque

def bfs_levels(graph, start):
    queue = deque([(start, 0)])
    seen = {start}
    while queue:
        node, depth = queue.popleft()
        yield node, depth
        for nei in graph[node]:
            if nei not in seen:
                seen.add(nei)
                queue.append((nei, depth + 1))
```

Example Problems:

- 127. Word Ladder (Blind 75, NeetCode 150)
- 752. Open the Lock (NeetCode 150)
- 433. Minimum Genetic Mutation (Top Microsoft)
- 1091. Shortest Path in Binary Matrix (NeetCode 150)

2.3 Multi-Source BFS

Seed the queue with multiple start nodes for nearest-source problems. **Complexity:** Time $O(V + E)$, Space $O(V)$.

```python
def multi_source_bfs(graph, sources):
    queue = deque((node, 0) for node in sources)
    seen = set(sources)
    while queue:
        node, depth = queue.popleft()
        yield node, depth
        for nei in graph[node]:
            if nei not in seen:
                seen.add(nei)
                queue.append((nei, depth + 1))
```

Example Problems:

- 994. Rotting Oranges (NeetCode 150)
- 286. Walls and Gates (Top Uber)
- 1765. Map of Highest Peak (Top surveyed)
- 542. 01 Matrix (Blind 75, NeetCode 150)

2.4 Bidirectional BFS

Meet in the middle to shrink branching factor on large search spaces. **Complexity:** Empirically $O(b^{d/2})$ vs $O(b^d)$.

```
def bidirectional_bfs(graph, start, target):
    if start == target:
        return 0
    front, back = {start}, {target}
    seen_front, seen_back = {start}, {target}
    depth = 0
    while front and back:
        depth += 1
        if len(front) > len(back):
            front, back = back, front
            seen_front, seen_back = seen_back, seen_front
        next_front = set()
        for node in front:
            for nei in graph[node]:
                if nei in seen_back:
                    return depth
                if nei not in seen_front:
                    seen_front.add(nei)
                    next_front.add(nei)
        front = next_front
    return -1
```

Example Problems:

- 127. Word Ladder (Blind 75)
- 752. Open the Lock (NeetCode 150)
- 773. Sliding Puzzle (Top Google)
- 847. Shortest Path Visiting All Nodes (Top Amazon)

3 🛰 Components & Connectivity

Union-Find and traversal-based component analysis solve reachability, clustering, and bipartite checks.

3.1 Connected Components Counter

Iterative DFS across all nodes, robust to disconnected graphs. **Complexity:** Time $O(V+E)$, Space $O(V)$.

```
def count_components(n, edges):
    graph = [[] for _ in range(n)]
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)

    seen, components = set(), 0
    for node in range(n):
        if node in seen:
            continue
        components += 1
        stack = [node]
        while stack:
            cur = stack.pop()
            if cur in seen:
                continue
            seen.add(cur)
            for nei in graph[cur]:
                if nei not in seen:
                    stack.append(nei)
    return components
```

Example Problems:

- 323. Number of Connected Components in an Undirected Graph (NeetCode 150)
- 547. Number of Provinces (Blind 75, NeetCode 150)
- 200. Number of Islands (Blind 75)
- 1254. Number of Closed Islands (Top Amazon)

3.2 Union-Find with Path Compression

Lightning-fast connectivity checks and Kruskal foundation. **Complexity:** Amortized $\alpha(n)$ per operation.

```
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))
        self.size = [1] * n
        self.components = n

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        rx, ry = self.find(x), self.find(y)
        if rx == ry:
            return False
        if self.size[rx] < self.size[ry]:
            rx, ry = ry, rx
        self.parent[ry] = rx
        self.size[rx] += self.size[ry]
        self.components -= 1
        return True
```

Example Problems:

- 684. Redundant Connection (NeetCode 150)
- 721. Accounts Merge (NeetCode 150)
- 990. Satisfiability of Equality Equations (Top Meta)
- 1319. Number of Operations to Make Network Connected (Top Amazon)

3.3 Union-Find with Rollback

Supports offline queries and divide-and-conquer scenarios.

```python
class UnionFindRollback:
    def __init__(self, n):
        self.parent = list(range(n))
        self.size = [1] * n
        self.history = []

    def find(self, x):
        while self.parent[x] != x:
            x = self.parent[x]
        return x

    def union(self, x, y):
        x, y = self.find(x), self.find(y)
        if x == y:
            self.history.append((-1, -1, -1))
            return False
        if self.size[x] < self.size[y]:
            x, y = y, x
        self.history.append((y, self.parent[y], self.size[x]))
        self.parent[y] = x
        self.size[x] += self.size[y]
        return True

    def snapshot(self):
        return len(self.history)

    def rollback(self, snap):
        while len(self.history) > snap:
            node, parent, size_before = self.history.pop()
            if node == -1:
                continue
            root = self.parent[node]
            self.parent[node] = parent
            self.size[root] = size_before
```

Example Problems:

- 1202. Smallest String With Swaps (NeetCode 150) *baseline DSU*
- 1627. Graph Connectivity With Threshold (Hard, offline unions)
- 1697. Checking Existence of Edge Length Limited Paths (NeetCode 150 Hard)
- Competitive programming dynamic connectivity sets (Codeforces EDU, AtCoder) for rollback drills

3.4 Bipartite Check via Coloring

Two-coloring with BFS catches odd cycles. **Complexity:** Time $O(V + E)$.

```
def is_bipartite(graph):
    color = {}
    for node in graph:
        if node in color:
            continue
        queue = deque([node])
        color[node] = 0
        while queue:
            cur = queue.popleft()
            for nei in graph[cur]:
                if nei not in color:
                    color[nei] = color[cur] ^ 1
                    queue.append(nei)
                elif color[nei] == color[cur]:
                    return False
    return True
```

Example Problems:

- 785. Is Graph Bipartite? (NeetCode 150)
- 886. Possible Bipartition (NeetCode 150)
- 1042. Flower Planting With No Adjacent (Top Google)
- 2493. Divide Nodes Into the Maximum Number of Groups (Top Meta)

4 🔃 Directed Graphs & Ordering

Directed acyclic graph techniques resolve scheduling, dependencies, and reachability with direction.

4.1 Cycle Detection (DFS with Path Set)

Track recursion stack to spot back edges in directed graphs. **Complexity:** Time $O(V+E)$.

```
def has_cycle_directed(graph):
    seen, path = set(), set()

    def dfs(node):
        if node in path:
            return True
        if node in seen:
            return False
        seen.add(node)
        path.add(node)
        for nei in graph[node]:
            if dfs(nei):
                return True
        path.remove(node)
        return False

    return any(dfs(node) for node in graph)
```

Example Problems:

- 207. Course Schedule (Blind 75)
- 802. Find Eventual Safe States (NeetCode 150)
- 1059. All Paths from Source Lead to Destination (Top Amazon)
- 2360. Longest Cycle in a Graph (Top Google)

4.2 Topological Sort (Kahn's Algorithm)

BFS with indegree tracking produces a valid order or detects cycles. **Complexity:** Time $O(V + E)$.

```python
def topo_sort(n, edges):
    indegree = [0] * n
    graph = [[] for _ in range(n)]
    for u, v in edges:
        graph[u].append(v)
        indegree[v] += 1

    queue = deque(node for node in range(n) if indegree[node] == 0)
    order = []
    while queue:
        node = queue.popleft()
        order.append(node)
        for nei in graph[node]:
            indegree[nei] -= 1
            if indegree[nei] == 0:
                queue.append(nei)
    return order if len(order) == n else []
```

Example Problems:

- 210. Course Schedule II (NeetCode 150)
- 269. Alien Dictionary (Top Google)
- 444. Sequence Reconstruction (Top Amazon)
- 1203. Sort Items by Groups Respecting Dependencies (Top Meta)

4.3 Topological Sort (DFS Post-Order)

Reverse post-order from DFS for stack-based ordering.

```python
def topo_sort_dfs(n, edges):
    graph = [[] for _ in range(n)]
    for u, v in edges:
        graph[u].append(v)

    seen, stack, path = set(), [], set()

    def dfs(node):
        if node in path:
            return False
        if node in seen:
            return True
        seen.add(node)
        path.add(node)
        for nei in graph[node]:
            if not dfs(nei):
                return False
        path.remove(node)
        stack.append(node)
        return True

    for node in range(n):
        if not dfs(node):
            return []
    return stack[::-1]
```

Example Problems:

- 2115. Find All Possible Recipes from Given Supplies (Top Amazon)
- 1203. Sort Items by Groups Respecting Dependencies (NeetCode 150 Hard)
- 1494. Parallel Courses II (Top Meta)
- 2050. Parallel Courses III (Top Meta)

4.4 Strongly Connected Components (Tarjan)

Tarjan uses a single DFS with low-link values. **Complexity:** Time $O(V + E)$.

```python
def tarjans_scc(n, edges):
    graph = [[] for _ in range(n)]
    for u, v in edges:
        graph[u].append(v)

    index = 0
    ids = [-1] * n
    low = [0] * n
    on_stack = [False] * n
    stack, components = [], []

    def dfs(at):
        nonlocal index
        ids[at] = low[at] = index
        index += 1
        stack.append(at)
        on_stack[at] = True

        for to in graph[at]:
            if ids[to] == -1:
                dfs(to)
                low[at] = min(low[at], low[to])
            elif on_stack[to]:
                low[at] = min(low[at], ids[to])

        if ids[at] == low[at]:
            component = []
            while True:
                node = stack.pop()
                on_stack[node] = False
                component.append(node)
                if node == at:
                    break
            components.append(component)

    for v in range(n):
        if ids[v] == -1:
            dfs(v)
    return components
```

Example Problems:

- 2360. Longest Cycle in a Graph (Top Google)
- 2127. Maximum Employees to Be Invited to a Meeting (Top Meta)
- 1203. Sort Items by Groups Respecting Dependencies (NeetCode 150 Hard)
- 2699. Modify Graph Edge Weights (Hard) *SCC feasibility*

5 🚀 Shortest Paths

Choose algorithms based on edge weights and restrictions: BFS for unweighted, Dijkstra for non-negative weights, Bellman-Ford for negatives, Floyd-Warshall for all-pairs.

5.1 Unweighted Shortest Path (BFS)

Return distance and optionally parent mapping.

```
def bfs_shortest_path(graph, start, target):
    queue = deque([(start, 0)])
    seen = {start}
    parent = {start: None}
    while queue:
        node, dist = queue.popleft()
        if node == target:
            path = []
            while node is not None:
                path.append(node)
                node = parent[node]
            return dist, path[::-1]
        for nei in graph[node]:
            if nei not in seen:
                seen.add(nei)
                parent[nei] = node
                queue.append((nei, dist + 1))
    return -1, []
```

Example Problems:

- 279. Perfect Squares (Blind 75)
- 1293. Shortest Path in a Grid with Obstacles Elimination (NeetCode 150)
- 847. Shortest Path Visiting All Nodes (Top Amazon)
- 1730. Shortest Path to Get Food (Top DoorDash)

5.2 0-1 BFS

Deque-based shortest path when edge weights are 0 or 1. **Complexity:** Time $O(V + E)$.

```python
from collections import deque

def zero_one_bfs(n, edges, start):
    graph = [[] for _ in range(n)]
    for u, v, w in edges:
        graph[u].append((v, w))
        graph[v].append((u, w))

    dist = [float('inf')] * n
    dist[start] = 0
    dq = deque([start])
    while dq:
        node = dq.popleft()
        for nei, weight in graph[node]:
            new_dist = dist[node] + weight
            if new_dist < dist[nei]:
                dist[nei] = new_dist
                if weight == 0:
                    dq.appendleft(nei)
                else:
                    dq.append(nei)
    return dist
```

Example Problems:

- 1368. Minimum Cost to Make at Least One Valid Path in a Grid (Top Google)
- 2290. Minimum Obstacle Removal to Reach Corner (Top Amazon)
- Competitive programming playlists (CSES Advanced Graph, Codeforces 0-1 BFS classics)

5.3 Dijkstra's Algorithm

Priority queue handles non-negative weighted edges. **Complexity:** Time $O((V + E) \log V)$.

```
import heapq
from collections import defaultdict

def dijkstra(n, edges, start):
    graph = defaultdict(list)
    for u, v, w in edges:
        graph[u].append((v, w))

    dist = [float('inf')] * n
    dist[start] = 0
    heap = [(0, start)]
    while heap:
        cur_dist, node = heapq.heappop(heap)
        if cur_dist > dist[node]:
            continue
        for nei, weight in graph[node]:
            new_dist = cur_dist + weight
            if new_dist < dist[nei]:
                dist[nei] = new_dist
                heapq.heappush(heap, (new_dist, nei))
    return dist
```

Example Problems:

- 743. Network Delay Time (Blind 75, NeetCode 150)
- 1514. Path with Maximum Probability (NeetCode 150)
- 1631. Path With Minimum Effort (NeetCode 150)
- 1786. Number of Restricted Paths From First to Last Node (Top Amazon)

5.4 Bellman-Ford

Detects negative cycles and handles edges with negative weights. **Complexity:** Time $O(V \cdot E)$.

```
def bellman_ford(n, edges, start):
    dist = [float('inf')] * n
    dist[start] = 0
    for _ in range(n - 1):
        updated = False
        for u, v, w in edges:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                updated = True
        if not updated:
            break
    for u, v, w in edges:
        if dist[u] + w < dist[v]:
            return None  # negative cycle detected
    return dist
```

Example Problems:

- 787. Cheapest Flights Within K Stops (NeetCode 150)
- 1514. Path with Maximum Probability (NeetCode 150) *via log-weights + relaxation*
- 1462. Course Schedule IV (Top Amazon)
- UVA 558. Wormholes (classic negative-cycle detection drill)

5.5 Floyd-Warshall

All-pairs shortest paths via DP, also detects negative cycles. **Complexity:** Time $O(V^3)$.

```
def floyd_warshall(n, dist):
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
    return dist
```

Example Problems:

- 1334. Find the City With the Smallest Number of Neighbors at a Threshold Distance (NeetCode 150)
- 1462. Course Schedule IV (Top Amazon)
- 2192. All Ancestors of a Node in a Directed Acyclic Graph (Top Meta)
- 2642. Design Graph With Shortest Path Calculator (Hard system design)

6 🗺️ Spanning Trees & Cuts

Minimum spanning trees and bridge detection appear frequently in infrastructure-style questions.

6.1 Kruskal's Algorithm

Sort edges by weight and union components. **Complexity:** Time $O(E \log E)$.

```
def kruskal(n, edges):
    uf = UnionFind(n)
    total_weight = 0
    for w, u, v in sorted(edges):
        if uf.union(u, v):
            total_weight += w
    return total_weight
```

Example Problems:

- 1584. Min Cost to Connect All Points (NeetCode 150)
- 1135. Connecting Cities With Minimum Cost (Top Amazon)
- 1168. Optimize Water Distribution in a Village (Top Google)
- 2492. Minimum Score of a Path Between Two Cities (Top Amazon)

6.2 Prim's Algorithm (PQ Variant)

Grow MST from a seed vertex using a heap. **Complexity:** Time $O(E \log V)$.

```
def prim(n, graph, start=0):
    seen = {start}
    edges = []
    for to, w in graph[start]:
        heapq.heappush(edges, (w, start, to))
    total = 0
    while edges and len(seen) < n:
        w, frm, to = heapq.heappop(edges)
        if to in seen:
            continue
        seen.add(to)
        total += w
        for nxt, weight in graph[to]:
            if nxt not in seen:
                heapq.heappush(edges, (weight, to, nxt))
    return total if len(seen) == n else float('inf')
```

Example Problems:

- 1584. Min Cost to Connect All Points (NeetCode 150)

- 1135. Connecting Cities With Minimum Cost (Top Amazon)
- 1168. Optimize Water Distribution in a Village (Top Google)
- 2812. Find the Safest Path in a Grid (Prim over risk graph)

6.3 Bridge Detection (Tarjan)

Low-link values identify critical edges. **Complexity:** Time $O(V + E)$.

```
def bridges(n, edges):
    graph = [[] for _ in range(n)]
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)

    ids = [-1] * n
    low = [0] * n
    time = 0
    result = []

    def dfs(node, parent):
        nonlocal time
        ids[node] = low[node] = time
        time += 1
        for nei in graph[node]:
            if nei == parent:
                continue
            if ids[nei] == -1:
                dfs(nei, node)
                low[node] = min(low[node], low[nei])
                if ids[node] < low[nei]:
                    result.append((node, nei))
            else:
                low[node] = min(low[node], ids[nei])

    for node in range(n):
        if ids[node] == -1:
            dfs(node, -1)
    return result
```

Example Problems:

- 1192. Critical Connections in a Network (Blind 75)
- 1489. Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree (NeetCode 150 Hard)
- 1568. Minimum Number of Days to Disconnect Island (Top Amazon)
- 2492. Minimum Score of a Path Between Two Cities (Top Amazon)

7 🎯 Stateful Graph Search

Many interview staples embed extra state in BFS nodes (position + keys, obstacles, etc.).

7.1 BFS with Bitmask State

Use tuples to encode location and collected keys/visited states. **Complexity:** Time $O(V \cdot 2^k)$ when tracking k bits.

```
def bfs_bitmask(start_state, next_states, goal_check):
    queue = deque([(start_state, 0)])
    seen = {start_state}
    while queue:
        state, dist = queue.popleft()
        if goal_check(state):
            return dist
        for nxt in next_states(state):
            if nxt not in seen:
                seen.add(nxt)
                queue.append((nxt, dist + 1))
    return -1
```

Example Problems:

- 864. Shortest Path to Get All Keys (NeetCode 150)
- 847. Shortest Path Visiting All Nodes (Blind 75)
- 1293. Shortest Path in a Grid with Obstacles Elimination (NeetCode 150)
- 773. Sliding Puzzle (Top Google)

7.2 Graph + Binary Search Hybrid

Binary search on answer plus connectivity check is common when answer space is monotonic. **Complexity:** $O(\log M \cdot (V + E))$ where M is answer search space.

```python
def binary_search_answer(lo, hi, feasible):
    while lo < hi:
        mid = (lo + hi) // 2
        if feasible(mid):
            hi = mid
        else:
            lo = mid + 1
    return lo
```

Example Problems:

- 1631. Path With Minimum Effort (NeetCode 150)
- 778. Swim in Rising Water (Blind 75)
- 1102. Path With Maximum Minimum Value (Top Amazon)
- 1970. Last Day Where You Can Still Cross (NeetCode 150)

8 💡 Interview Playbook

Quick checklist to choose the right graph pattern under pressure.

8.1 Decision Guide

- Need to count components or determine connectivity? Use DFS/BFS or Union-Find.
- Unweighted shortest paths or minimum moves? Use BFS; consider bidirectional or multi-source variants.
- Non-negative weights? Reach for Dijkstra or 0-1 BFS when weights are binary.
- Negative edges? Use Bellman-Ford (detect cycles) or SPFA variants in practice.
- Dependencies with no cycles? Apply topological sort (Kahn or DFS).
- Repeated connectivity queries offline? Union-Find with rollback or segment tree divide-and-conquer.
- Need all-pairs or dense graphs? Floyd-Warshall or repeated Dijkstra.
- Weighted spanning tree? Kruskal or Prim depending on input form.

8.2 Rapid Practice Sets

- **Blind 75 Graph Core:** 133, 200, 207, 210, 323, 417, 684, 695, 733, 802.
- **NeetCode 150 Graph Tier:** 127, 133, 146, 200, 207, 210, 329, 399, 542, 694, 695, 743, 752, 778, 802, 886, 994, 1020, 1192, 1514, 1631.
- **Top Company Heat Map (FAANG + Unicorns):** 269, 310, 332, 399, 490, 721, 752, 787, 934, 1091, 1192, 1293, 1584, 1631, 1976, 2493.
- **Advanced Hard Hitters:** 691, 815, 847, 1203, 1368, 1489, 1494, 1559, 1609, 1970, 2050.