

ChatGPT - DSA Mastery - Python



Graph Mastery DSA

Below is your **GRAPH MASTERY DSA GUIDE** — crafted for your **DSA Mastery project style**, extremely structured, patterns-first, with Python templates, intuition shortcuts, and interview-oriented patterns.

★ GRAPH MASTERY (0 → 100 Roadmap)

Everything you need to solve **95% of graph questions** in interviews & competitive programming.

🎯 1. Graph Theory Essentials (What every DSA candidate must know)

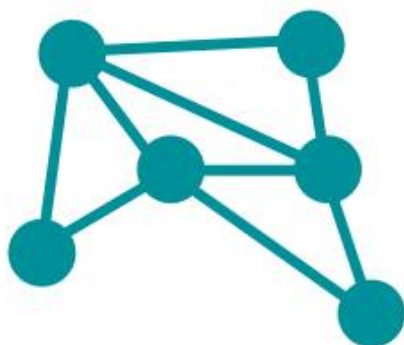
Unweighted Edge



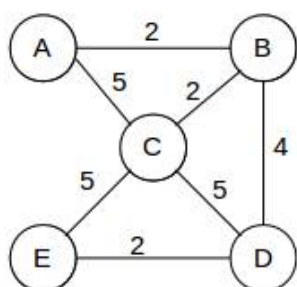
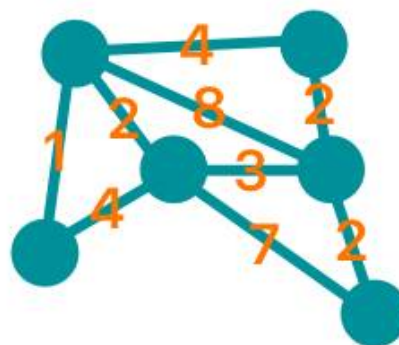
Weighted Edge



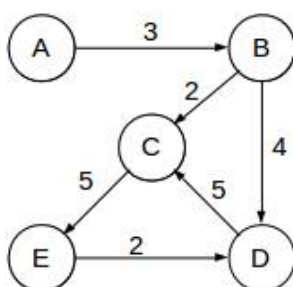
Undirected Graph



Directed Graph



Undirected



Directed

Graph Types

Type	Meaning	Uses
Undirected Graph	edges are bi-directional	networks, friendships
Directed Graph (Digraph)	edges have direction	dependencies, tasks
Weighted Graph	edges have weights	shortest path, cost
DAG (Directed Acyclic Graph)	no cycles	DP on graphs, topo sort
Tree	connected acyclic graph	hierarchy

Type	Meaning	Uses
Forest	collection of trees	divide & conquer

2. Representation Templates (Interview-ready)

Adjacency List (MOST USED)

```
python

graph = {i: [] for i in range(n)}
for u, v in edges:
    graph[u].append(v)
    graph[v].append(v) # remove for directed
```

Adjacency Matrix

```
python

mat = [[0]*n for _ in range(n)]
for u, v in edges:
    mat[u][v] = mat[v][u] = 1
```

Edge List

```
python

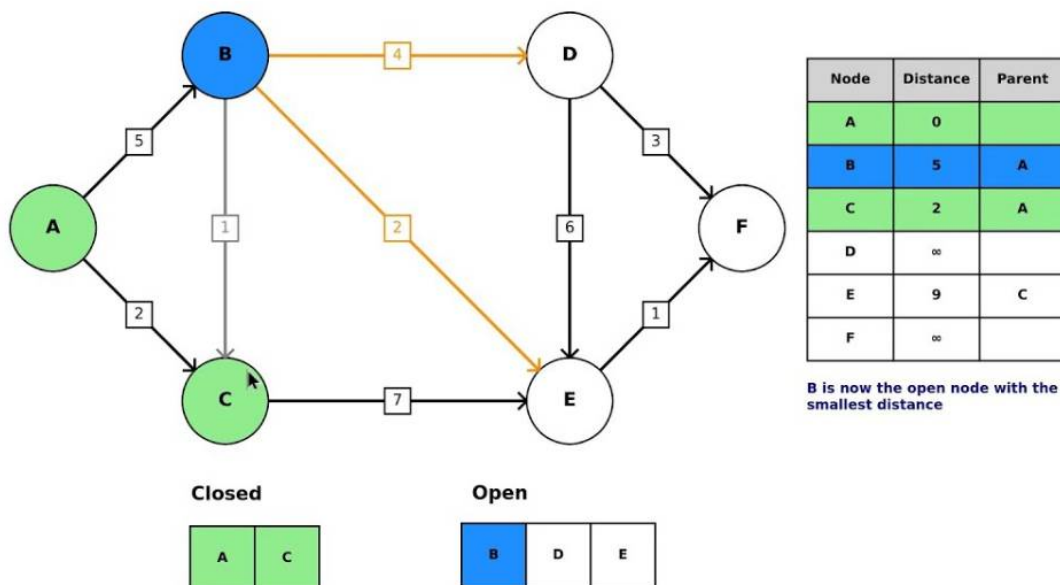
edges = [(u, v), ...]
```

★ 3. MAJOR GRAPH PATTERNS YOU MUST MASTER

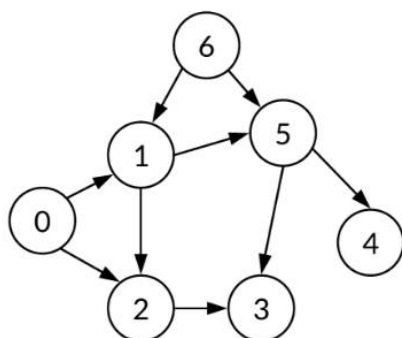


https://www.researchgate.net/publication/224202469/figure/fig4/AS%3A825156905750530%401573744145189/Flow-Chart-for-BFS-Algorithm-III-GENERAL-ALGORITHM-Without-applying-graph-theory-there.png?utm_source=chatgpt.com

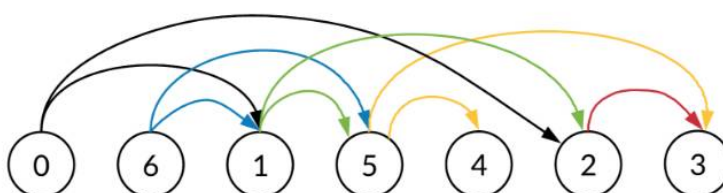
Dijkstra's Algorithm



Unsorted graph



Topologically sorted graph



Below is the **pattern-first master list** used in FAANG interviews.

PATTERN 1: Graph Traversal — DFS / BFS (Super Core Pattern)

When to use

- Reachability
- Component counting
- Cycle detection
- Grid problems (treated as graph)

DFS TEMPLATE (recursive)

python

```
def dfs(node, parent):
    visited.add(node)
    for nei in graph[node]:
        if nei != parent and nei not in visited:
            dfs(nei, node)
```

BFS TEMPLATE

python

```
from collections import deque

def bfs(start):
    q = deque([start])
    visited = {start}
    while q:
        node = q.popleft()
        for nei in graph[node]:
            if nei not in visited:
                visited.add(nei)
                q.append(nei)
```

PATTERN 2: Connected Components

Used in:

- Number of islands
- Provinces
- Graph clusters

python

```
count = 0
for node in graph:
    if node not in visited:
        dfs(node, -1)
        count += 1
```

PATTERN 3: Cycle Detection (Directed & Undirected)

Undirected

python

```
def hasCycle(u, parent):
    visited.add(u)
    for v in graph[u]:
        if v not in visited:
            if hasCycle(v, u):
```

```

        return True
    elif v != parent:
        return True
    return False

```

Directed (Use 3-color or recursion stack)

python

```

WHITE, GRAY, BLACK = 0, 1, 2
color = [0]*n

def dfs(u):
    color[u] = GRAY
    for v in graph[u]:
        if color[v] == GRAY:
            return True
        if color[v] == WHITE and dfs(v):
            return True
    color[u] = BLACK
    return False

```

PATTERN 4: Topological Sorting (Kahn + DFS)

Used only in **DAGs** for:

- course schedule
- build order
- dependency resolution

Kahn's Algorithm Template

python

```

from collections import deque

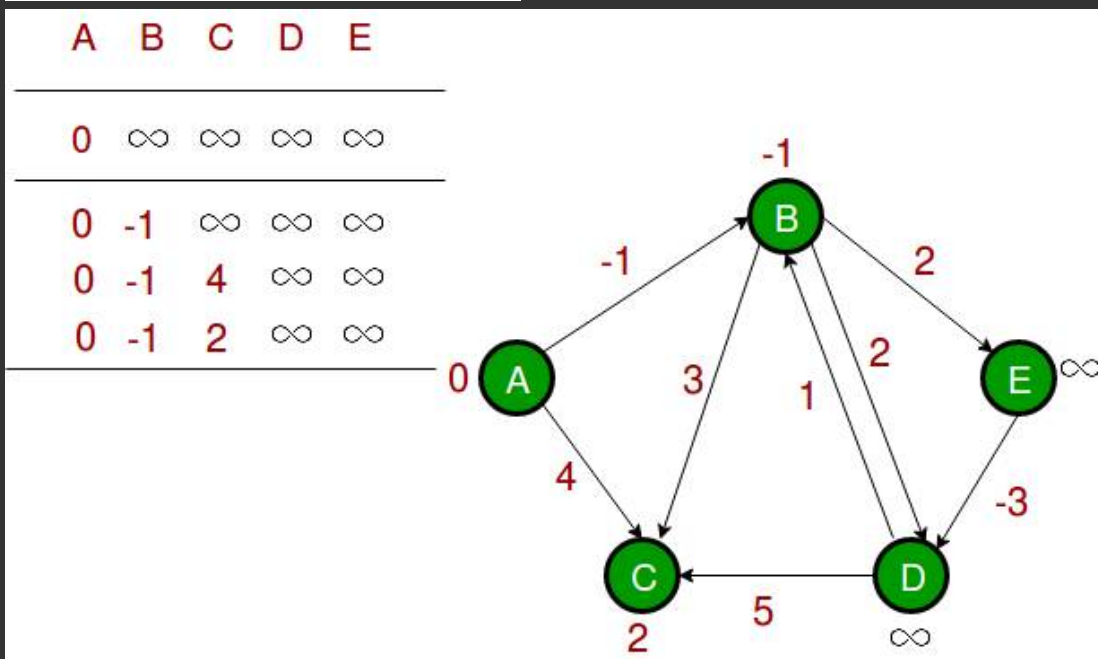
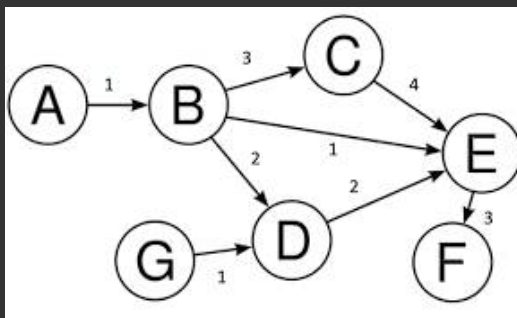
indeg = [0]*n
for u in range(n):
    for v in graph[u]:
        indeg[v] += 1

q = deque([i for i in range(n) if indeg[i] == 0])
order = []

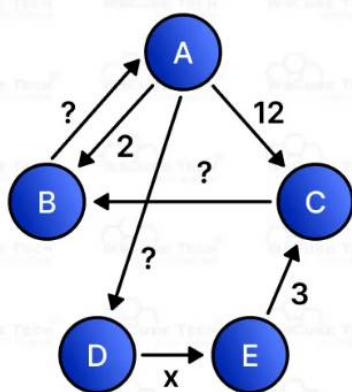
while q:
    u = q.popleft()
    order.append(u)
    for v in graph[u]:
        indeg[v] -= 1
        if indeg[v] == 0:
            q.append(v)

```

PATTERN 5: Shortest Path Algorithms



Floyd Warshall Algorithm



	A	B	C	D	E
A	0	2	10	5	7
B	3	0	13	8	10
C	7	4	0	12	14
D	12	9	5	0	X
E	10	7	3	15	0

When to use which?

Graph	Use
Unweighted	BFS
Weighted (no negative weights)	Dijkstra

Graph	Use
Negative weights	Bellman-Ford
All-pairs shortest	Floyd-Warshall

Dijkstra Template

```
python

import heapq

def dijkstra(start):
    dist = [float('inf')] * n
    dist[start] = 0
    pq = [(0, start)]

    while pq:
        d, u = heapq.heappop(pq)
        if d > dist[u]:
            continue
        for v, w in graph[u]:
            if dist[v] > d + w:
                dist[v] = d + w
                heapq.heappush(pq, (dist[v], v))
    return dist
```

PATTERN 6: Union-Find / DSU

Used for:

- Cycle detection
- Kruskal MST
- Connected components

```
python

class DSU:
    def __init__(self, n):
        self.p = list(range(n))
        self.r = [0]*n

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

    def union(self, a, b):
        pa, pb = self.find(a), self.find(b)
        if pa == pb: return False
        if self.r[pa] < self.r[pb]:
            pa, pb = pb, pa
        self.p[pb] = pa
        if self.r[pa] == self.r[pb]:
            self.r[pa] += 1
        return True
```


PATTERN 7: Minimum Spanning Tree — Prim & Kruskal

Kruskal Template

```
python

edges.sort(key=lambda x: x[2])
dsu = DSU(n)
mst = 0

for u, v, w in edges:
    if dsu.union(u, v):
        mst += w
```

Prim Template

```
python

import heapq
visited = set()
pq = [(0, 0)] # (cost, node)
total = 0

while pq:
    w, u = heapq.heappop(pq)
    if u in visited: continue
    visited.add(u)
    total += w
    for v, wt in graph[u]:
        heapq.heappush(pq, (wt, v))
```

PATTERN 8: Grid Graphs (VERY IMPORTANT for LeetCode)

Treat (i, j) cells as nodes.

Template

```
python

DIR = [(1,0), (-1,0), (0,1), (0,-1)]

def bfs(i, j):
    q = deque([(i,j)])
    visited.add((i,j))
    while q:
        x, y = q.popleft()
        for dx, dy in DIR:
            nx, ny = x+dx, y+dy
            if in_bounds(nx, ny) and (nx,ny) not in visited and grid[nx][ny] == '1':
                visited.add((nx,ny))
                q.append((nx,ny))
```

PATTERN 9: Bipartite Graph Check

Used in:

- Graph coloring
- Possible bipartition

python

```
color = {}
def bfs(start):
    color[start] = 0
    q = deque([start])
    while q:
        u = q.popleft()
        for v in graph[u]:
            if v not in color:
                color[v] = color[u] ^ 1
                q.append(v)
            elif color[v] == color[u]:
                return False
    return True
```

PATTERN 10: Advanced Patterns

★ Tarjan Algorithms

- SCC (Kosaraju/Tarjan)
- Bridges
- Articulation Points

★ Multi-source BFS

Used in

- Rotten oranges
- Walls & gates

★ Topo-based DP

Longest path in DAG

Course schedule problems

★ Meet-in-the-middle graph problems

Used in complex state graph BFS.

GRAPH PROBLEM MASTER LIST (90% of LC Graph Qs)

Core:

- BFS/DFS
- Connected components
- Cycle detection

- Topological sort
- BFS shortest path
- Dijkstra
- Grid traversal
- Bipartite
- MST
- DSU

Advanced:

- Tarjan SCC
- Bridges & articulation points
- Multi-source BFS
- 0/1 BFS
- Dial's Algorithm
- Floyd Warshall
- Bellman Ford

★ Want me to generate:

- ✓ Graph Cheat-sheet PDF
- ✓ 100 Graph Problems checklist (LC tags)
- ✓ Python templates folder
- ✓ Graph Visualization Notes
- ✓ Graph Pattern Flashcards

Just tell me **which one you want next**.

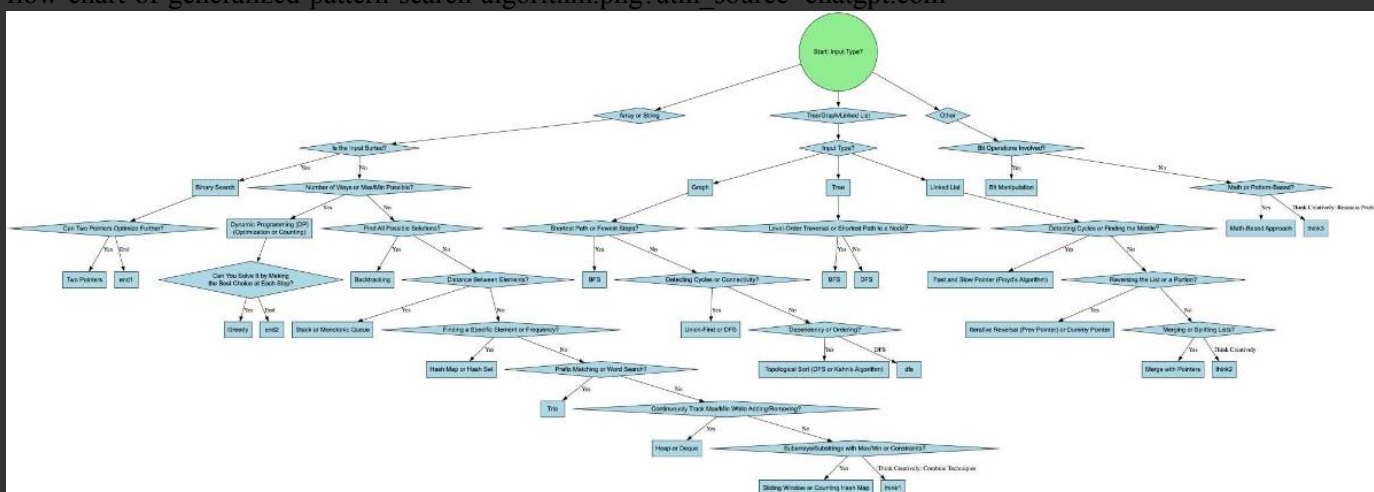


What patterns and when it is used, how to identify the questions and proper cheat-sheet

Perfect — here is your **Ultimate Graph Patterns Cheat-Sheet** with:

- ✓ Patterns
- ✓ When to use
- ✓ How to identify the question type instantly

★ GRAPH PATTERNS — IDENTIFY & APPLY (The Master Sheet)



✓ When to Use

- Explore neighbors
- Count connected components
- Reachability / Can we go from $A \rightarrow B$?
- Flood fill
- Tree/graph traversal

Problem mentions:

- "Find all connected..."
- "Reachable"
- "Number of islands"
- "Count regions / groups"
- Grid questions

✓ Template

python

```
def bfs(start):
    q = deque([start])
    visited = {start}
    while q:
        u = q.popleft()
        for v in graph[u]:
            if v not in visited:
                visited.add(v)
                q.append(v)
```

🎯 PATTERN 2 — Connected Components

✓ When to Use

- Count independent groups
- Cluster detection
- Provinces, islands, friendships

✓ How to Identify

Look for:

- "How many groups?"
- "Connected regions?"
- "Number of ... components?"

✓ Template

python

```
count = 0
for node in graph:
    if node not in visited:
        dfs(node)
        count += 1
```

🎯 PATTERN 3 — Cycle Detection

✓ When to Use

- Is graph cyclic?
- Is this a valid tree?
- Detect loops

✓ How to Identify

Problem mentions:

- "Contains cycle?"
- "Check if tree"
- "Prerequisites contain a cycle?"

✓ Template (Undirected)

```
python

def dfs(u, parent):
    for v in graph[u]:
        if v not in visited:
            visited.add(v)
            if dfs(v, u): return True
        elif v != parent:
            return True
    return False
```

PATTERN 4 — Bipartite Check

✓ When to Use

- Graph coloring
- Teams / partition
- Opposite groups

✓ How to Identify

Key words:

- "Can we divide into two groups?"
- "No edges between same group"

✓ Template

```
python

color = {}
def bfs(start):
    q = deque([start])
    color[start] = 0
    while q:
        u = q.popleft()
        for v in graph[u]:
            if v not in color:
                color[v] = 1 - color[u]
                q.append(v)
            elif color[v] == color[u]:
```

```

return True
return False

```

PATTERN 5 — Topological Sort (Kahn or DFS)

✓ When to Use

- DAG
- Course scheduling
- Dependency resolution

✓ How to Identify

Look for:

- "Order of tasks/prerequisites"
- "Dependencies"
- "Directed graph, no cycles allowed"

✓ Template (Kahn)

```

python

indeg = [0]*n
for u in graph:
    for v in graph[u]: indeg[v] += 1

q = deque([i for i in range(n) if indeg[i]==0])
order = []

while q:
    u = q.popleft()
    order.append(u)
    for v in graph[u]:
        indeg[v] -= 1
        if indeg[v] == 0: q.append(v)

```

PATTERN 6 — Shortest Path

✓ Subpatterns & When to Use

Algorithm	When
BFS	Unweighted shortest path
Dijkstra	Weighted (positive)
Bellman-Ford	Negative weights
Floyd-Warshall	All-pairs shortest path

✓ How to Identify

- “Minimum cost / shortest distance / least steps”
- Weighted edges → Dijkstra
- No weights → BFS

✓ Template (Dijkstra)

python

```

pq = [(0, start)]
dist = [inf]*n
dist[start] = 0

while pq:
    d, u = heapq.heappop(pq)
    for v, w in graph[u]:
        if d + w < dist[v]:
            dist[v] = d + w
            heapq.heappush(pq, (dist[v], v))

```

PATTERN 7 — Union-Find / DSU

✓ When to Use

- Check if an edge creates cycle
- Kruskal MST
- Merge components

✓ How to Identify

Words like:

- “Is this a valid tree?”
- “Redundant connection”
- “Friend circles”
- “Merge sets, find groups”

✓ Template

python

```

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

```



```
def union(a, b):
    pa, pb = find(a), find(b)
    if pa == pb: return False
    parent[pb] = pa
    return True
```

PATTERN 8 — Minimum Spanning Tree (Prim/Kruskal)

✓ When to Use

- Minimum cost to connect everything
- Network wiring problems

✓ How to Identify

Terms like:

- “Minimum cost to connect all nodes”
- “Minimum wiring/network cost”

✓ Template (Kruskal)

```
python

edges.sort(key=lambda x: x[2])
for u, v, w in edges:
    if union(u, v):
        cost += w
```

PATTERN 9 — Grid Graph (BFS/DFS on matrix)

✓ When to Use

Any matrix problem is a graph problem.

✓ How to Identify

- “Islands”
- “Regions”
- “Shortest path in grid”
- “Flood fill”

✓ Template

```
python
```

```
DIR = [(1,0), (-1,0), (0,1), (0,-1)]
```

PATTERN 10 — Advanced Graph Patterns

✓ Tarjan Algorithms

- SCC (Kosaraju/Tarjan) → decomposition
- Bridges & articulation points → critical edges

✓ When to Identify

Look for:

- “Strongly connected components”
- “Critical edges/nodes”
- “Remove one edge and graph disconnects?”

THE ULTIMATE PATTERN IDENTIFICATION TABLE (PRINT THIS!)

Question Mentions	Pattern
Count regions/groups	Connected Components
Reachable / path exists?	DFS/BFS
Shortest path	BFS / Dijkstra
Weighted graph	Dijkstra / MST
Negative weights	Bellman-Ford
All pairs shortest	Floyd-Warshall
Course schedule	Topological Sort
Valid tree?	Cycle detection / DSU
Can divide into two groups	Bipartite
Minimum cost to connect all	MST (Prim/Kruskal)
Rotting oranges / multi-source	Multi-source BFS
Islands / grid traversal	Grid BFS/DFS

★ COMPLETE CHEAT-SHEET: Pattern → Identifier → Template → Use Case

If you want, I can produce:

- ✂ **One-page Graph Patterns PDF**
- ✂ **100 Graph Problems Mapped to Patterns**
- ✂ **Pattern Detection Flowchart**
- ✂ **Interview Flashcards**
- ✂ **Folder of Python Templates (ready to paste in LC)**

Which one should I generate next?



100 Graph Problems Mapped to Patterns.

Here is your **ULTIMATE LIST: 100 GRAPH PROBLEMS MAPPED TO PATTERNS** — categorized exactly by **which pattern solves them**, so you instantly know **what approach + what template** to use.

This is structured for **DSA Mastery**, interview prep, and quick pattern recall.

★ 100 GRAPH PROBLEMS → PATTERN MAPPING

(Every problem mapped to the correct pattern)

■ PATTERN 1 — DFS/BFS Traversal (Core Exploration)

Used for: reachability, count areas, simple graph search.

1. Clone Graph
2. Number of Provinces
3. Keys and Rooms
4. Reorder Routes to Make All Paths Lead to City 0
5. Evaluate Division
6. Number of Operations to Make Network Connected
7. Max Area of Island
8. Number of Closed Islands
9. Surrounded Regions
10. Counting Components in Undirected Graph
11. Word Ladder (BFS)
12. Word Ladder II
13. Race Car (BFS)
14. Shortest Path in Binary Matrix

15. Rotting Oranges (multi-source BFS)
 16. As Far From Land As Possible
 17. Remove Invalid Parentheses (graph BFS)
 18. Jump Game IV
 19. Graph Valid Tree (DFS version)
 20. Water Jug Problem (state graph BFS)
-

PATTERN 2 — Connected Components

Used for: counting groups, clustering, region identification.

21. Number of Islands
 22. Number of Distinct Islands
 23. Most Stones Removed with Same Row or Column
 24. Friend Circles / Number of Provinces
 25. Connected Components in Matrix
 26. Island Perimeter
 27. Count Sub-Islands
 28. Find Eventual Safe States
 29. Path With Maximum Probability
 30. Accounts Merge (via DFS)
-

PATTERN 3 — Cycle Detection

Used for: validate tree, detect loops in directed/undirected.

Undirected

31. Graph Valid Tree
32. Redundant Connection
33. Redundant Connection II
34. Detect Cycle in Undirected Graph
35. Largest Cycle in Graph

Directed

36. Course Schedule
37. Course Schedule II
38. Find Eventual Safe States

39. Minimum Height Trees (cycle removal idea)

40. Detect Cycle in Directed Graph

PATTERN 4 — Bipartite Graph Check

Used for: 2-coloring, team division.

41. Is Graph Bipartite?

42. Possible Bipartition

43. Divide Players into Teams

44. Strange Printer II

45. Check if Graph is Two-Colorable

PATTERN 5 — Topological Sort (Kahn / DFS)

Used for: DAG ordering, scheduling, resolver ordering.

46. Course Schedule

47. Course Schedule II

48. Alien Dictionary

49. Minimum Time to Finish All Tasks With Dependencies

50. Sort Characters by Frequency (induced topological idea)

51. Sequence Reconstruction

52. Find Order of Characters

53. Parallel Courses

54. Longest Increasing Path in Matrix (DAG)

55. Shortest Path in DAG

PATTERN 6 — Shortest Path (BFS, Dijkstra, Bellman-Ford, Floyd)

Unweighted → BFS

56. Word Ladder

57. Minimum Steps to Get Food

58. Shortest Path in Binary Matrix

59. Snakes and Ladders

60. Zero-One BFS: Shortest Path with Obstacles Elimination

Weighted → Dijkstra

- 61. Network Delay Time
- 62. Path With Minimum Effort
- 63. Path With Maximum Probability
- 64. Cheapest Flights Within K Stops
- 65. Find City With Smallest Number of Neighbors
- 66. Dijkstra on Grid (min-cost path)

Negative Weights → Bellman-Ford

- 67. Detect Negative Cycle
- 68. Cheapest Flights (relaxed via Bellman-Ford)

All-Pairs → Floyd-Warshall

- 69. Find the City With the Smallest Reach
 - 70. Graph Diameter for Weighted Graph
 - 71. Minimum Fuel Cost for All Pairs
-

PATTERN 7 — Union-Find / DSU

Used for: merging sets, detect cycles, component tracking.

- 72. Redundant Connection
 - 73. Redundant Connection II
 - 74. Most Stones Removed
 - 75. Number of Provinces
 - 76. Accounts Merge
 - 77. Graph Valid Tree
 - 78. Connecting Cities With Minimum Cost
 - 79. Satisfiability of Equality Equations
 - 80. Check if All Paths Are Valid After Restrictions
-

PATTERN 8 — Minimum Spanning Tree (Prim / Kruskal)

Used for: connecting everything with minimum cost.

- 81. Kruskal MST Problem
- 82. Prim MST Problem

- 83. Minimum Cost to Connect All Points (Manhattan MST)
- 84. Connecting Cities With Minimum Cost
- 85. Optimize Water Distribution in a Village
- 86. Min Cost to Supply Electricity

PATTERN 9 — Grid-Based Graph Problems

Treat matrix as graph

- 87. Number of Islands
- 88. Max Area of Island
- 89. Shortest Path in Binary Matrix
- 90. Rotting Oranges
- 91. Walls and Gates (multi-source BFS)
- 92. Pacific Atlantic Water Flow
- 93. Count Enclaves
- 94. Remove Islands
- 95. Gas Station Grid Path
- 96. Fire Spread Problem

PATTERN 10 — Tarjan Algorithms (Advanced Graph Theory)

SCC (Strongly Connected Components)

- 97. Critical Connections in a Network (Bridges)
- 98. Articulation Points Problem
- 99. Tarjan SCC → Number of SCCs
- 100. Find All Bridges in Graph

BONUS: FAST RECOGNITION TABLE (USE THIS IN INTERVIEWS)

Clue in Problem	Use Pattern
"Minimum steps", unweighted	BFS
"Minimum cost", weighted	Dijkstra
"Dependencies/order/prereq"	Topological Sort
"Divide into two groups"	Bipartite
"Minimum wiring / connect all nodes"	MST
"Cycle exists?"	DFS/Union-Find

Clue in Problem	Use Pattern
"How many groups?"	Connected Components
"Grid: islands, regions"	DFS/BFS Grid
"Strongly connected / critical edges"	Tarjan
"Merge accounts / groups"	DSU

★ WANT THE NEXT LEVEL?

I can generate:

- ✓ PDF version of this 100-problem map
- ✓ A full Problem → Pattern Flowchart
- ✓ A 1-page Graph Pattern Formula Sheet
- ✓ Python Template Pack for all patterns
- ✓ A 150-problem expanded version

Which one should I generate next?



DFS and BFS Patterns

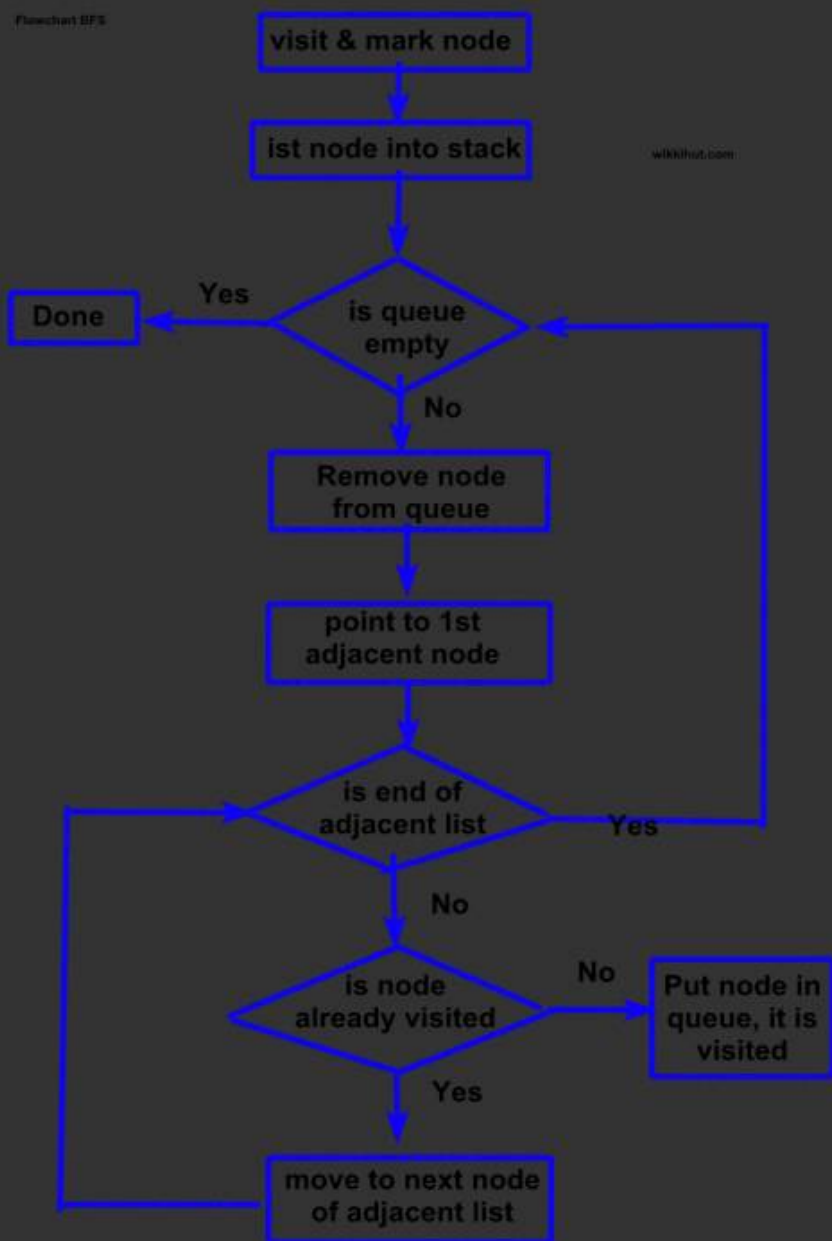
Below is your **DSA Mastery: DFS & BFS PATTERNS CHEAT-SHEET** — clean, structured, pattern-first, and optimized for **fast identification + fast coding** in interviews.

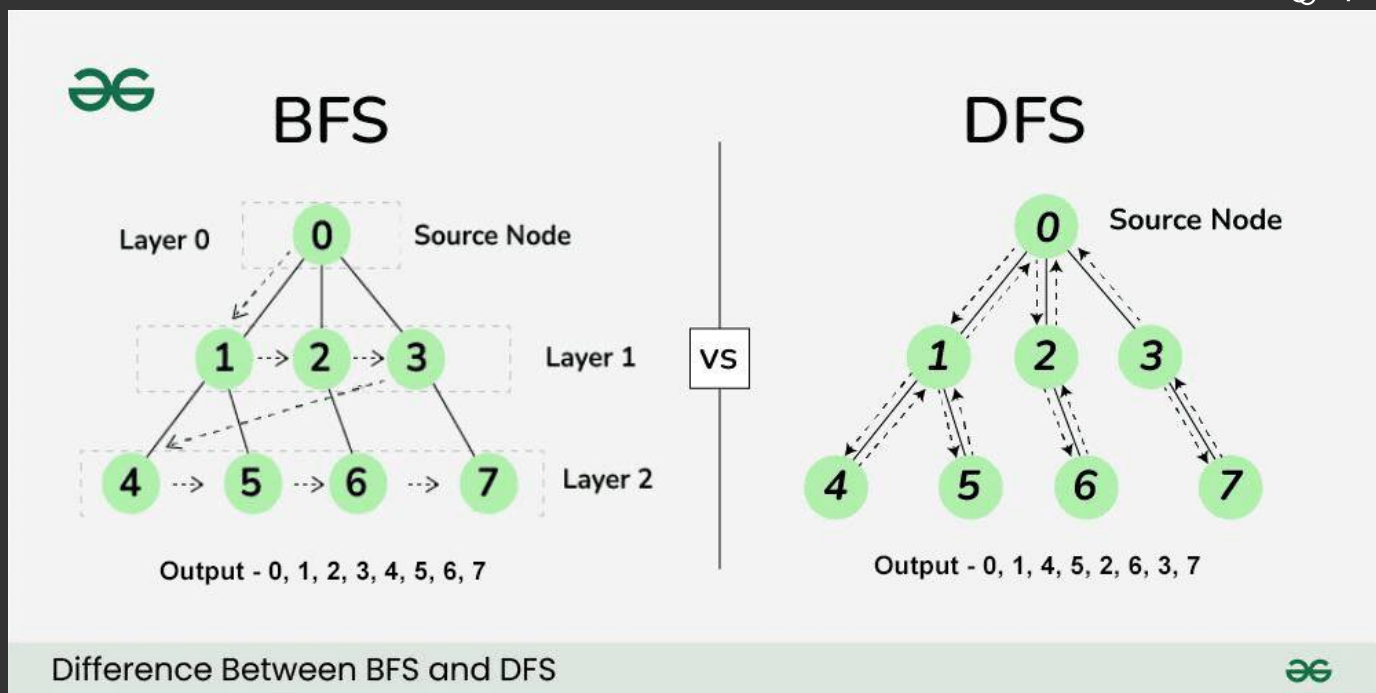
This is the **ultimate guide** to every DFS/BFS pattern you'll ever use.

★ DFS & BFS MASTER PATTERN CHEAT-SHEET



https://www.researchgate.net/publication/224202469/figure/fig4/AS%3A825156905750530%401573744145189/Flow-Chart-for-BFS-Algorithm-III-GENERAL-ALGORITHM-Without-applying-graph-theory-there.png?utm_source=chatgpt.com





DFS and BFS solve **70%+ of graph + tree + grid problems**.

Below are the **12 core patterns** grouped by:

- ✓ **What the pattern is**
- ✓ **When to use it**
- ✓ **How to instantly identify the question**
- ✓ **Minimal Python template**

PATTERN 1 — Basic DFS (Depth-first Search)

✓ **When to Use**

- Full exploration
- Recursive structure
- Backtracking
- Searching all paths

✓ **Identify From Problem:**

- "Find all possible..."
- "Explore region..."
- "Flood fill"
- "Count areas/islands"

✓ **Template**

python

```
def dfs(u):
    visited.add(u)
    for v in graph[u]:
        if v not in visited:
            dfs(v)
```

PATTERN 2 — Basic BFS (Breadth-first Search)

✓ When to Use

- Shortest path in unweighted graph
- Level-by-level expansion
- Multi-source traversal

✓ Identify

- "Minimum steps"
- "Shortest distance"
- "Minimum operations to reach X"

✓ Template

python

```
from collections import deque

def bfs(start):
    q = deque([start])
    visited = {start}

    while q:
        u = q.popleft()
        for v in graph[u]:
            if v not in visited:
                visited.add(v)
                q.append(v)
```

PATTERN 3 — Connected Components (DFS/BFS)

✓ When to Use

- Count islands/groups
- Find independent clusters

✓ Identify

- "How many groups/regions/components?"

✓ Template

```
python

count = 0
for node in graph:
    if node not in visited:
        dfs(node)
        count += 1
```

■ PATTERN 4 — Cycle Detection (DFS)

✓ When to Use

- Check if tree
- Detect loops
- Validate connections

✓ Identify

- “Contains cycle?”
- “Is this graph a valid tree?”

Undirected Cycle Detection

```
python

def dfs(u, parent):
    for v in graph[u]:
        if v not in visited:
            visited.add(v)
            if dfs(v, u): return True
        elif v != parent:
            return True
    return False
```

Directed Cycle Detection

Use **coloring** or **recursion stack**.

```
python

WHITE, GRAY, BLACK = 0, 1, 2
color = [WHITE] * n

def dfs(u):
    color[u] = GRAY
    for v in graph[u]:
        if color[v] == GRAY:
            return True
        if color[v] == WHITE and dfs(v):
            return True
```

```
color[u] = BLACK
return False
```

PATTERN 5 — Topological DFS (DFS Ordering)

✓ When to Use

- DAG longest/shortest path
- Task scheduling

✓ Identify

- "Order of tasks"
- "Prerequisites"

✓ Template

```
python

order = []

def dfs(u):
    visited.add(u)
    for v in graph[u]:
        if v not in visited:
            dfs(v)
    order.append(u)
```

PATTERN 6 — Grid DFS/BFS (Matrix Graph)

✓ When to Use

Matrix problems = graph problems.

✓ Identify

- "Islands"
- "Regions"
- "Path through grid"

✓ Template

```
python

DIR = [(1,0),(-1,0),(0,1),(0,-1)]

def dfs(i, j):
    visited.add((i,j))
    for dx, dy in DIR:
        x, y = i+dx, j+dy
```

```
if in_bounds(x, y) and (x,y) not in visited and grid[x][y] == 1:
    dfs(x, y)
```

PATTERN 7 — Multi-source BFS

✓ When to Use

- BFS starting from multiple points simultaneously
- Spread problems

✓ Identify

- “All rotten oranges rot others...”
- “All gates spread distance to rooms”

✓ Template

```
python

q = deque(all_sources)
visited = set(all_sources)

while q:
    x = q.popleft()
    for each neighbor:
        if not visited:
            visited.add(nei)
            q.append(nei)
```

PATTERN 8 — Shortest Path in Unweighted Graph (BFS)

✓ When to Use

- Minimum steps
- Minimum depth
- Word ladder
- Snakes and ladders

✓ Identify

- “Minimum moves/steps”

✓ Template

```
python

dist = {start: 0}
q = deque([start])

while q:
```

```
u = q.popleft()
for v in graph[u]:
    if v not in dist:
        dist[v] = dist[u] + 1
        q.append(v)
```

PATTERN 9 — State Graph BFS (VERY IMPORTANT)

✓ When to Use

State = node, transitions = edges.

Used in:

- Word Ladder
- Sliding Puzzle
- Shortest path in state space
- BFS on transformations

✓ Identify

- "Operations allowed: +1, -1, *2..."
- "Transform X to Y"

✓ Template

python

```
q = deque([start_state])
visited = {start_state}

while q:
    state = q.popleft()
    for next_state in generate(state):
        if next_state not in visited:
            visited.add(next_state)
            q.append(next_state)
```

PATTERN 10 — Backtracking DFS (Path Construction)

✓ When to Use

- Find all paths
- Word search
- N-Queens
- Maze paths

✓ Identify

- "Find all solutions"
- "Find any valid path"

✓ Template

```
python

def dfs(path, node):
    if goal(node):
        result.append(path[:])
        return

    for v in graph[node]:
        path.append(v)
        dfs(path, v)
        path.pop()
```

PATTERN 11 — Tree DFS (Special Case)

Tree = acyclic = simpler DFS.

✓ When to Use

- Subtree size
- LCA
- Path sums
- Diameter

✓ Template

```
python

def dfs(node):
    if not node:
        return
    dfs(node.left)
    dfs(node.right)
```

PATTERN 12 — Iterative DFS / BFS

✓ When to Use

- Avoid recursion depth
- Very large graphs

✓ Template (stack DFS)

python

```
stack = [start]
while stack:
    u = stack.pop()
    if u in visited: continue
    visited.add(u)
    for v in graph[u]:
        stack.append(v)
```

DFS & BFS Pattern Recognition Table (Print This)

Problem Statement Says	Pattern
"Minimum steps"	BFS
"Reachable"	DFS/BFS
"Number of regions"	Connected components
"Cycle exists?"	DFS cycle detection
"Islands / grid"	Grid DFS/BFS
"All sources spread outward"	Multi-source BFS
"Transform X to Y"	State graph BFS
"Find all paths"	DFS backtracking
"Tree depth/path sum"	Tree DFS

★ Want the next piece?

I can generate:

- ✓ DFS/BFS 1-page PDF cheat-sheet
- ✓ 50 Problems mapped to DFS/BFS patterns
- ✓ Python template pack for all patterns
- ✓ Flowchart: "DFS or BFS?" decision maker
- ✓ Advanced DFS (Tarjan: SCC, Bridges, AP) patterns

Which one should I generate next?