



Graph Live Class



**INTERVIEW
KICKSTART**

Instructor Introduction

Chao Zhang

Tech Lead & Manager at Google (Bay Area)

Ex: Workday, Salesforce, LinkedIn

10+ yrs experience

Engineering, tech lead, team management



**INTERVIEW
KICKSTART**

Summary of preclass videos

The first part of the video series introduces basic graph theory by taking the example of the Eulerian Path/Cycle problem which is then analyzed, and basic terminology and patterns of reasoning about Graphs are introduced. Once the solution is obtained in theory, then we look at how Graphs are represented in memory - using adjacency lists, maps and matrices. Their tradeoffs are discussed, and finally the solution to the problem is implemented in pseudocode.

Summary of preclass videos

Most interview problems on Graphs are on BFS/DFS, we focus on them and look at the BFS and DFS trees that get created as a result of running BFS/DFS on an undirected graph. Their time and space complexities are discussed, and we finally look at how to determine the number of connected components in an undirected graph by adding an outer loop that triggers BFS/DFS multiple times until the whole graph is explored. Between iterative stack DFS and recursive DFS, we prefer recursive DFS.

Agenda

Graph Introduction and Background

Topics and Scope of the Class

Undirected Graph

- DFS, BFS

Directed Graph

- DFS, BFS, Topological Sort, Shortest Path

Final Q&A

Remote class best practices

Keep your video camera on

Keep track of time

Mute your mics when not speaking

Participants for interactions, speak up once you are asked

Sudo code for coding examples, focus on logic not syntax

Post class Q&A

IK Graphs curriculum

Basic Graph Theory

Graph representations

General graph traversal

BFS

DFS

Shortest paths

1. Traversals of undirected and directed graphs
2. Finding connected components
3. Detecting cycles
4. Bipartite

top.sort

SCC
etc.

Dijkstra

Prim

Best-first

A*

Bellman-Ford
Floyd-Warshall

Kruskal



**INTERVIEW
KICKSTART**

323. Number of Connected Components in an Undirected Graph

Medium

👍 465

💬 14

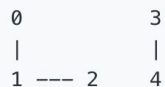
🤍 Favorite

🔗 Share

Given n nodes labeled from 0 to $n - 1$ and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

Example 1:

Input: $n = 5$ and $\text{edges} = [[0, 1], [1, 2], [3, 4]]$



Output: 2

Example 2:

Input: $n = 5$ and $\text{edges} = [[0, 1], [1, 2], [2, 3], [3, 4]]$



Output: 1



INTERVIEW
KICKSTART

Live Coding Session

Live Walk Through and Sample Code

261. Graph Valid Tree

Medium

👍 720

💬 22

♡ Favorite

🔗 Share

Given n nodes labeled from 0 to $n-1$ and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

Example 1:

Input: $n = 5$, and $edges = [[0,1], [0,2], [0,3], [1,4]]$

Output: `true`

Example 2:

Input: $n = 5$, and $edges = [[0,1], [1,2], [2,3], [1,3], [1,4]]$

Output: `false`

Note: you can assume that no duplicate edges will appear in `edges`. Since all edges are undirected, $[0,1]$ is the same as $[1,0]$ and thus will not appear together in `edges`.




**INTERVIEW
KICKSTART**

Live Coding Session

Live Walk Through and Sample Code

785. Is Graph Bipartite?

Medium  698  89  Favorite  Share

Given an undirected `graph`, return `true` if and only if it is bipartite.

Recall that a graph is *bipartite* if we can split its set of nodes into two independent subsets A and B such that every edge in the graph has one node in A and another node in B.

The graph is given in the following form: `graph[i]` is a list of indexes `j` for which the edge between nodes `i` and `j` exists. Each node is an integer between `0` and `graph.length - 1`. There are no self edges or parallel edges: `graph[i]` does not contain `i`, and it doesn't contain any element twice.

Example 1:

Input: `[[1,3], [0,2], [1,3], [0,2]]`

Output: `true`

Explanation:

The graph looks like this:

0----1

| |

| |

3----2

We can divide the vertices into two groups: `{0, 2}` and `{1, 3}`.



**INTERVIEW
KICKSTART**

Live Coding Session

Live Walk Through and Sample Code

909. Snakes and Ladders

Medium

137

404

Favorite

Share

On an $N \times N$ board, the numbers from 1 to $N*N$ are written *boustrophedonically* starting from the bottom left of the board, and alternating direction each row. For example, for a 6 x 6 board, the numbers are written as follows:

36	35	34	33	32	31
25	26	27	28	29	30
24	23	22	21	20	19
13	14	15	16	17	18
12	11	10	9	8	7
1	2	3	4	5	6

You start on square 1 of the board (which is always in the last row and first column). Each move, starting from square x , consists of the following:

- You choose a destination square s with number $x+1$, $x+2$, $x+3$, $x+4$, $x+5$, or $x+6$, provided this number is $\leq N*N$.
 - (This choice simulates the result of a standard 6-sided die roll: ie., there are always **at most 6 destinations, regardless of the size of the board**.)
- If s has a snake or ladder, you move to the destination of that snake or ladder. Otherwise, you move to s .

A board square on row r and column c has a "snake or ladder" if $\text{board}[r][c] \neq -1$. The destination of that snake or ladder is $\text{board}[r][c]$.

Note that you only take a snake or ladder at most once per move: if the destination to a snake or ladder is the start of another snake or ladder, you do **not** continue moving. (For example, if the board is `[[4,-1],[-1,3]]`, and on the first move your destination square is '2', then you finish your first move at '3', because you do **not** continue moving to '4'.)

Return the least number of moves required to reach square $N*N$. If it is not possible, return -1.



INTERVIEW
KICKSTART

Live Coding Session

Live Walk Through and Sample Code

210. Course Schedule II

Medium

👍 1033

💬 75

🤍 Favorite

📄 Share

There are a total of n courses you have to take, labeled from 0 to $n-1$.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1 , which is expressed as a pair: $[0,1]$

Given the total number of courses and a list of prerequisite **pairs**, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

Example 1:

Input: 2, $[[1,0]]$

Output: $[0,1]$

Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is $[0,1]$.

Example 2:

Input: 4, $[[1,0],[2,0],[3,1],[3,2]]$

Output: $[0,1,2,3]$ or $[0,2,1,3]$

Explanation: There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is $[0,1,2,3]$. Another correct ordering is $[0,2,1,3]$.



INTERVIEW
KICKSTART

Live Coding Session

Live Walk Through and Sample Code

Slides Created by:
Omkar Deshpande