Graph Interview Handbook — Java

Curated set of essential graph problems for FAANG/MAANG interviews

Generated: 2025-10-25 13:37:21

Each problem: Statement, Java method signature, Example testcases, Optimized Java solution, Complexity, Source

Table of Contents

Table of Contents

Number of Islands — Medium

Number of Islands — Medium

Source: LeetCode 200 | Link: https://leetcode.com/problems/number-of-islands/

Problem statement

Given an m x n 2D grid grid of '1's (land) and '0's (water), return the number of islands. An island is formed by connecting adjacent lands horizontally or vertically. You may modify the grid in-place. Constraints: 1 <= m,n <= 300.

Java method signature

```
public int numIslands(char[][] grid)
```

Example test case(s)

```
Example 1:
Input:
["1","1","1","1","0"]
["1","1","0","1","0"]
["1","1","0","0","0"]
["0","0","0","0","0"]
Output: 1
```

Optimized Java solution (concise)

```
class Solution {
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) return 0;
        int m = grid.length, n = grid[0].length, count = 0;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == '1') {
                    dfs(grid, i, j, m, n);
                    count++;
                }
            }
        }
        return count;
    }
    private void dfs(char[][] g, int i, int j, int m, int n) {
        if (i < 0 || j < 0 || i >= m || j >= n || g[i][j] != '1') return;
        g[i][j] = '0';
        dfs(g, i+1, j, m, n);
        dfs(g, i-1, j, m, n);
        dfs(g, i, j+1, m, n);
        dfs(g, i, j-1, m, n);
    }
}
// Time: O(m * n), Space: O(m * n) recursion worst-case
```

Time & Space Complexity

Time: O(m * n). Space: O(m * n) recursion worst-case.

Shortest Path in Binary Matrix — Medium

Shortest Path in Binary Matrix — Medium

Source: LeetCode 1091 | Link: https://leetcode.com/problems/shortest-path-in-binary-matrix/

Problem statement

Given an n x n binary matrix grid where 0 denotes an empty cell and 1 denotes a blocked cell, return the length of the shortest path from (0,0) to (n-1,n-1) using 8-directional moves. Return -1 if start or end is blocked or if no path exists.

Java method signature

```
public int shortestPathBinaryMatrix(int[][] grid)
```

Example test case(s)

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

Optimized Java solution (concise)

```
class Solution {
    public int shortestPathBinaryMatrix(int[][] g) {
        int n = g.length;
        if (g[0][0] == 1 || g[n-1][n-1] == 1) return -1;
        int[][] dirs = {{1,0},{-1,0},{0,1},{0,-1},{1,1},{1,-1},{-1,1},{-1,-1}};
        boolean[][] vis = new boolean[n][n];
        java.util.ArrayDeque<int[]> q = new java.util.ArrayDeque<>();
        q.offer(new int[]{0,0,1}); vis[0][0]=true;
        while(!q.isEmpty()){
            int[] cur=q.poll(); int x=cur[0], y=cur[1], d=cur[2];
            if(x==n-1 && y==n-1) return d;
            for(int[] t:dirs){
                int nx=x+t[0], ny=y+t[1];
                if(nx>=0 && ny>=0 && nx<n && ny<n && !vis[nx][ny] && g[nx][ny]==0){
                    vis[nx][ny]=true; q.offer(new int[]{nx,ny,d+1});
                }
            }
        }
        return -1;
    }
}
// Time: O(n^2)
```

Time & Space Complexity

Time: O(n^2). Space: O(n^2).

Is Graph Bipartite? — Medium

Is Graph Bipartite? — Medium

Source: LeetCode 785 | Link: https://leetcode.com/problems/is-graph-bipartite/

Problem statement

Given an undirected graph represented as adjacency list graph where graph[i] is a list of nodes adjacent to i, determine if it is bipartite (2-colorable so that no adjacent nodes share same color).

Java method signature

```
public boolean isBipartite(int[][] graph)
```

Example test case(s)

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

Optimized Java solution (concise)

```
class Solution {
    public boolean isBipartite(int[][] g) {
        int n = g.length;
        int[] color = new int[n];
        java.util.Arrays.fill(color, -1);
        for (int i = 0; i < n; i++) {
            if (color[i] == -1) {
                java.util.ArrayDeque<Integer> q = new java.util.ArrayDeque<>();
                q.offer(i); color[i]=0;
                while(!q.isEmpty()){
                    int u=q.poll();
                    for(int v: g[u]){
                        if(color[v]==-1){ color[v]=color[u]^1; q.offer(v); }
                        else if(color[v]==color[u]) return false;
                    }
                }
            }
        }
        return true;
    }
}
// Time: O(V+E)
```

Time & Space Complexity

Time: O(V+E). Space: O(V).

Detect Cycle in Undirected Graph — Medium

Detect Cycle in Undirected Graph — Medium

Source: GeeksforGeeks | Link: https://www.geeksforgeeks.org/detect-cycle-undirected-graph/

Problem statement

Given an undirected graph of V vertices represented with adjacency lists, determine if the graph contains a cycle. Return true if any cycle exists, false otherwise.

Java method signature

```
public boolean isCycle(int V, ArrayList<ArrayList<Integer>> adj)
```

Example test case(s)

```
Example: V=5, edges: 0-1,1-2,2-0,3-4 -> Output: true
```

Optimized Java solution (concise)

```
class Solution {
    public boolean isCycle(int V, java.util.ArrayList<java.util.ArrayList<Integer>> adj) {
        boolean[] vis = new boolean[V];
        for (int i = 0; i < V; i++) if (!vis[i]) if (dfs(i, -1, adj, vis)) return true;
        return false;
    }
    private boolean dfs(int u, int p, java.util.ArrayList<java.util.ArrayList<Integer>> ad
        vis[u] = true;
        for (int v : adj.get(u)) {
            if (!vis[v]) {
                if (dfs(v, u, adj, vis)) return true;
            } else if (v != p) return true;
        }
        return false;
    }
}
// Time: O(V+E)
```

Time & Space Complexity

Time: O(V+E). Space: O(V) recursion.

Detect Cycle in Directed Graph — Medium

Detect Cycle in Directed Graph — Medium

Source: GeeksforGeeks / LeetCode 207 concept | Link: https://www.geeksforgeeks.org/detect-cycle-directed-graph/

Problem statement

Given a directed graph with V vertices and adjacency lists,
determine if the graph has a cycle. Use DFS with recursion stack
(or Kahn's algorithm for topo-sort) to detect cycles.

Java method signature

```
public boolean isCyclic(int V, ArrayList<ArrayList<Integer>> adj)
```

Example test case(s)

```
Example: V=2, edges: 0->1,1->0 -> Output: true
```

Optimized Java solution (concise)

```
class Solution {
    public boolean isCyclic(int V, java.util.ArrayList<java.util.ArrayList<Integer>> adj)
        int[] state = new int[V]; //0=unvisited,1=visiting,2=done
        for (int i = 0; i < V; i++) if (state[i] == 0) if (dfs(i, adj, state)) return true
        return false;
    }
    private boolean dfs(int u, java.util.ArrayList<java.util.ArrayList<Integer>> adj, int[
        st[u] = 1;
        for (int v : adj.get(u)) {
            if (st[v] == 0) { if (dfs(v, adj, st)) return true; }
            else if (st[v] == 1) return true;
        }
        st[u] = 2; return false;
    }
}
// Time: O(V+E)
```

Time & Space Complexity

Time: O(V+E). Space: O(V) recursion.

Snakes and Ladders — Medium

Snakes and Ladders — Medium

Source: LeetCode 909 | Link: https://leetcode.com/problems/snakes-and-ladders/

Problem statement

Given an n x n board for Snakes and Ladders where squares are numbered in a boustrophedon pattern. On each turn, roll a 6-sided die and move; if land on square with ladder/snake (board[r][c] != -1) move to that square. Return minimum moves to reach n^2 from 1, or -1 if impossible.

Java method signature

```
public int snakesAndLadders(int[][] board)
```

Example test case(s)

```
Example input: see LeetCode sample; Output: 4
```

Optimized Java solution (concise)

```
class Solution {
    public int snakesAndLadders(int[][] b) {
        int n = b.length;
        java.util.function.Function<Integer,int[]> idx = s -> {
            int r = n - 1 - (s-1)/n;
            int c = ((s-1)/n) % 2 == 0 ? (s-1)%n : n-1 - ((s-1)%n);
            return new int[]{r,c};
        };
        java.util.ArrayDeque<Integer> q = new java.util.ArrayDeque<>();
        boolean[] vis = new boolean[n*n+1];
        q.offer(1); vis[1]=true; int moves=0;
        while(!q.isEmpty()){
            for(int sz=q.size(); sz>0; sz--){
                int cur=q.poll(); if(cur==n*n) return moves;
                for(int d=1; d<=6 && cur+d<=n*n; d++){
                    int nxt=cur+d; int[] rc = idx.apply(nxt);
                    if(b[rc[0]][rc[1]] != -1) nxt = b[rc[0]][rc[1]];
                    if(!vis[nxt]){ vis[nxt]=true; q.offer(nxt); }
                }
            }
            moves++;
        }
        return -1;
    }
}
// Time: O(n^2)
```

Time & Space Complexity

Time: O(n^2). Space: O(n^2).

As Far from Land as Possible — Medium

As Far from Land as Possible — Medium

Source: LeetCode 1162 | Link: https://leetcode.com/problems/as-far-from-land-as-possible/

Problem statement

Given an n x n grid with 0 (water) and 1 (land), return the maximum distance of any water cell to the nearest land cell. If grid has only land or only water, return -1. Use multi-source BFS from all land cells.

Java method signature

```
public int maxDistance(int[][] grid)
```

Example test case(s)

```
Input: [[1,0,1],[0,0,0],[1,0,1]] -> Output: 2
```

Optimized Java solution (concise)

```
class Solution {
    public int maxDistance(int[][] g) {
        int n = g.length;
        java.util.ArrayDeque<int[]> q = new java.util.ArrayDeque<>();
        boolean[][] vis = new boolean[n][n];
        for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(g[i][j]==1){ q.offer(new int[]{i,j});
        if(q.isEmpty()||q.size()==n*n) return -1;
        int dist=-1; int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
        while(!q.isEmpty()){
            int sz=q.size(); dist++;
            for(int k=0;k<sz;k++){
                int[] cur=q.poll();
                for(int[] d:dirs){ int ni=cur[0]+d[0], nj=cur[1]+d[1]; if(ni>=0&&nj>=0&&ni
(new int[]{ni,nj}); } }
            }
        }
        return dist;
    }
}
// Time: O(n^2)
```

Time & Space Complexity

Time: O(n^2). Space: O(n^2).

Shortest Bridge — Medium

Shortest Bridge — Medium

Source: LeetCode 934 | Link: https://leetcode.com/problems/shortest-bridge/

Problem statement

Given a grid with exactly two islands (1s), flip 0s to 1s to connect the two islands. Return minimum number of flips required.
Approach: DFS to mark one island, then BFS expand until reach other island.

Java method signature

```
public int shortestBridge(int[][] grid)
```

Example test case(s)

```
Input: [[0,1],[1,0]] -> Output: 1
```

Optimized Java solution (concise)

```
class Solution {
    int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
    public int shortestBridge(int[][] g){
        int n=g.length; java.util.ArrayDeque<int[]> q=new java.util.ArrayDeque<>();
        boolean found=false; boolean[][] vis=new boolean[n][n];
        for(int i=0;i<n && !found;i++) for(int j=0;j<n && !found;j++) if(g[i][j]==1){ dfs(
        int steps=0;
        while(!q.isEmpty()){
            int sz=q.size();
            for(int k=0;k<sz;k++){
                int[] cur=q.poll();
                for(int[] d:dirs){ int ni=cur[0]+d[0], nj=cur[1]+d[1];
                    if(ni>=0&&nj>=0&&ni<n&&nj<n&&!vis[ni][nj]){
                        if(g[ni][nj]==1) return steps; vis[ni][nj]=true; q.offer(new int[]
                    }
                }
            }
            steps++;
        }
        return -1;
    }
    private void dfs(int[][] g,int i,int j,boolean[][] vis,java.util.ArrayDeque<int[]> q){
        int n=g.length; if(i<0||j<0||i>=n||j>=n||vis[i][j]||g[i][j]==0) return;
        vis[i][j]=true; q.offer(new int[]{i,j});
        dfs(g,i+1,j,vis,q); dfs(g,i-1,j,vis,q); dfs(g,i,j+1,vis,q); dfs(g,i,j-1,vis,q);
    }
}
// Time: O(n^2)
```

Time & Space Complexity

Time: O(n^2). Space: O(n^2).

Shortest Path in a Grid with Obstacles Elimination — Hard

Shortest Path in a Grid with Obstacles Elimination — Hard

Source: LeetCode 1293 | Link: https://leetcode.com/problems/shortest-path-in-a-grid-with-obstacles-elimination/

Problem statement

Given an m x n grid of 0s (empty) and 1s (obstacle), you may eliminate at most k obstacles. Return length of shortest path from (0,0) to (m-1,n-1) with up to k eliminations, or -1 if impossible. Use BFS on state (x,y,remaining_k).

Java method signature

```
public int shortestPath(int[][] grid, int k)
```

Example test case(s)

```
Input: grid=[[0,0,0],[1,1,0],[0,0,0],[0,1,1],[0,0,0]], k=1 -> Output: 6
```

Optimized Java solution (concise)

```
class Solution {
    public int shortestPath(int[][] grid,int K){
        int m=grid.length,n=grid[0].length;
        boolean[][][] vis=new boolean[m][n][K+1];
        java.util.ArrayDeque<int[]> q=new java.util.ArrayDeque<>(); q.offer(new int[]{0,0,
        int steps=0; int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
        while(!q.isEmpty()){
            int sz=q.size();
            for(int i=0;i<sz;i++){
                int[] cur=q.poll(); int x=cur[0], y=cur[1], rem=cur[2];
                if(x==m-1 && y==n-1) return steps;
                for(int[] d:dirs){ int nx=x+d[0], ny=y+d[1]; if(nx<0||ny<0||nx>=m||ny>=n)
                    int nrem=rem-(grid[nx][ny]==1?1:0); if(nrem<0) continue; if(!vis[nx][n
[]{nx,ny,nrem}); }
                }
            }
            steps++;
        }
        return -1;
    }
}
// Time: O(m*n*K)
```

Time & Space Complexity

Time: O(m*n*K). Space: O(m*n*K).

Course Schedule — Medium

Course Schedule — Medium

Source: LeetCode 207 | Link: https://leetcode.com/problems/course-schedule/

Problem statement

There are numCourses labeled 0..numCourses-1 and prerequisites pairs [a,b] meaning to take course a you must first take b. Return true if you can finish all courses (i.e., the directed graph has no cycle).

Java method signature

```
public boolean canFinish(int numCourses, int[][] prerequisites)
```

Example test case(s)

```
Input: numCourses=2, prerequisites=[[1,0]] -> Output: true
```

Optimized Java solution (concise)

```
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        java.util.List<java.util.List<Integer>> adj=new java.util.ArrayList<>();
        for(int i=0;i<numCourses;i++) adj.add(new java.util.ArrayList<>());
        int[] indeg=new int[numCourses];
        for(int[] p:prerequisites){ adj.get(p[1]).add(p[0]); indeg[p[0]]++; }
        java.util.ArrayDeque<Integer> q=new java.util.ArrayDeque<>();
        for(int i=0;i<numCourses;i++) if(indeg[i]==0) q.offer(i);
        int seen=0;
        while(!q.isEmpty()){ int u=q.poll(); seen++; for(int v:adj.get(u)) if(--indeg[v]==
        return seen==numCourses;
    }
}
// Time: O(V+E)
```

Time & Space Complexity

Time: O(V+E). Space: O(V+E).

Course Schedule II — Medium

Course Schedule II — Medium

Source: LeetCode 210 | Link: https://leetcode.com/problems/course-schedule-ii/

Problem statement

Return a valid ordering of courses to finish all prerequisites (topological order), or an empty array if impossible.

Java method signature

```
public int[] findOrder(int numCourses, int[][] prerequisites)
```

Example test case(s)

```
Input: numCourses=4, prerequisites=[[1,0],[2,0],[3,1],[3,2]] -> Output: [0,1,2,3]
```

Optimized Java solution (concise)

```
class Solution {
    public int[] findOrder(int numCourses, int[][] prerequisites) {
        java.util.List<java.util.List<Integer>> adj=new java.util.ArrayList<>();
        for(int i=0;i<numCourses;i++) adj.add(new java.util.ArrayList<>());
        int[] indeg=new int[numCourses];
        for(int[] p:prerequisites){ adj.get(p[1]).add(p[0]); indeg[p[0]]++; }
        java.util.ArrayDeque<Integer> q=new java.util.ArrayDeque<>();
        for(int i=0;i<numCourses;i++) if(indeg[i]==0) q.offer(i);
        int[] res=new int[numCourses]; int idx=0;
        while(!q.isEmpty()){ int u=q.poll(); res[idx++]=u; for(int v:adj.get(u)) if(--inde
        return idx==numCourses?res:new int[0];
    }
}
// Time: O(V+E)
```

Time & Space Complexity

Time: O(V+E).

Alien Dictionary — Hard

Alien Dictionary — Hard

Source: LeetCode 269 | Link: https://leetcode.com/problems/alien-dictionary/

Problem statement

Given a sorted dictionary of words in an alien language, derive a valid ordering of characters. Build directed edges between the first differing characters of adjacent words. Return ordering string or "" if inconsistent/cycle.

Java method signature

```
public String alienOrder(String[] words)
```

Example test case(s)

```
Input: ["wrt","wrf","er","ett","rftt"] -> Output: "wertf"
```

Optimized Java solution (concise)

```
class Solution {
    public String alienOrder(String[] words){
        java.util.Map<Character, java.util.Set<Character>> adj=new java.util.HashMap<>();
        int[] indeg=new int[26]; java.util.Arrays.fill(indeg,-1);
        for(String w:words) for(char ch:w.toCharArray()) if(indeg[ch-'a']==-1){ indeg[ch-'
)); }
        for(int i=0;i+1<words.length;i++){
            String a=words[i], b=words[i+1]; int j=0;
            while(j<a.length() && j<b.length() && a.charAt(j)==b.charAt(j)) j++;
            if(j==b.length() && a.length()>b.length()) return "";
            if(j<a.length() && j<b.length()){
                char u=a.charAt(j), v=b.charAt(j);
                if(adj.get(u).add(v)) indeg[v-'a']++;
            }
        }
        java.util.ArrayDeque<Character> q=new java.util.ArrayDeque<>(); StringBuilder sb=n
        for(int i=0;i<26;i++) if(indeg[i]==0) q.offer((char)('a'+i));
        while(!q.isEmpty()){
            char u=q.poll(); sb.append(u);
            for(char v: adj.getOrDefault(u, java.util.Collections.emptySet())) if(--indeg[
        }
        for(int i=0;i<26;i++) if(indeg[i]>0) return "";
        return sb.toString();
    }
}
// Time: O(total chars + edges)
```

Time & Space Complexity

Time: O(sum of word lengths + unique edges).

Remove Max Number of Edges to Keep Graph Fully Traversable — Hard

Remove Max Number of Edges to Keep Graph Fully Traversable — Hard

Source: LeetCode 1579 | Link: https://leetcode.com/problems/remove-max-number-of-edges-to-keep-graph-fully-traversable/

Problem statement

n nodes, edges with types: 1 (Alice), 2 (Bob), 3 (both). Remove maximum edges while keeping graph fully traversable for both Alice and Bob. Return max removed or -1 if impossible.

Java method signature

```
public int maxNumEdgesToRemove(int n, int[][] edges)
```

Example test case(s)

```
Input: see LeetCode sample -> Output: 2
```

Optimized Java solution (concise)

```
class Solution {
    static class DSU{ int[] p,r; DSU(int n){ p=new int[n+1]; r=new int[n+1]; for(int i=1;i
        int find(int x){ return p[x]==x?x:(p[x]=find(p[x])); }
        boolean union(int a,int b){ a=find(a); b=find(b); if(a==b) return false; if(r[a]<r
++; return true; }
    }
    public int maxNumEdgesToRemove(int n,int[][] edges){
        DSU a=new DSU(n), b=new DSU(n); int used=0;
        java.util.Arrays.sort(edges,(x,y)->y[0]-x[0]); // type 3 first
        for(int[] e:edges){
            int t=e[0], u=e[1], v=e[2];
            if(t==3){ boolean ua=a.union(u,v), ub=b.union(u,v); if(ua|ub) used++; }
            else if(t==1){ if(a.union(u,v)) used++; }
            else { if(b.union(u,v)) used++; }
        }
        for(int i=2;i<=n;i++) if(a.find(i)!=a.find(1) || b.find(i)!=b.find(1)) return -1;
        return edges.length - used;
    }
}
// Time: O(E α(N))
```

Time & Space Complexity

Time: O(E α(N)).

Checking Existence of Edge Length Limited Paths — Hard

Checking Existence of Edge Length Limited Paths — Hard

Source: LeetCode 1697 | Link: https://leetcode.com/problems/checking-existence-of-edge-length-limited-paths/

Problem statement

Given weighted edges and queries (u,v,limit), determine if a path exists from u to v where all edges on the path have weight < limit. Sort edges and queries, use DSU to connect edges below limit and answer queries offline.

Java method signature

```
public boolean[] distanceLimitedPathsExist(int n, int[][] edgeList, int[][] querie
```

Example test case(s)

```
Example: small graph queries -> boolean array answer
```

Optimized Java solution (concise)

```
class Solution {
    static class DSU{ int[] p; DSU(int n){ p=new int[n]; for(int i=0;i<n;i++) p[i]=i; } in
 } void union(int a,int b){ a=find(a); b=find(b); if(a!=b) p[a]=b; } }
    public boolean[] distanceLimitedPathsExist(int n,int[][] edges,int[][] queries){
        java.util.Arrays.sort(edges,(a,b)->Integer.compare(a[2],b[2]));
        int m=queries.length; int[][] qs=new int[m][4];
        for(int i=0;i<m;i++) qs[i]=new int[]{queries[i][0],queries[i][1],queries[i][2],i};
        java.util.Arrays.sort(qs,(a,b)->Integer.compare(a[2],b[2]));
        DSU d=new DSU(n); boolean[] ans=new boolean[m]; int ei=0;
        for(int[] q: qs){
            while(ei<edges.length && edges[ei][2] < q[2]){ d.union(edges[ei][0], edges[ei]
            ans[q[3]] = d.find(q[0])==d.find(q[1]);
        }
        return ans;
    }
}
// Time: O((E+Q) log(E+Q))
```

Time & Space Complexity

Time: O((E+Q) log(E+Q)).

Making A Large Island — Hard

Making A Large Island — Hard

Source: LeetCode 827 | Link: https://leetcode.com/problems/making-a-large-island/

Problem statement

Given a binary grid, you may flip at most one 0 to 1. Return size of largest island possible after at most one flip. Label components and compute sizes; for each 0 sum sizes of unique neighboring components + 1.

Java method signature

```
public int largestIsland(int[][] grid)
```

Example test case(s)

```
Input: [[1,0],[0,1]] -> Output: 3
```

Optimized Java solution (concise)

```
class Solution {
    int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
    public int largestIsland(int[][] g){ int n=g.length; int id=2; int[] size=new int[n*n+
        for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(g[i][j]==1){ size[id]=dfs(g,i,j,id);
        int ans=0; for(int s:size) ans=Math.max(ans,s);
        for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(g[i][j]==0){
            java.util.HashSet<Integer> seen=new java.util.HashSet<>(); int sum=1;
            for(int[] d:dirs){ int ni=i+d[0], nj=j+d[1]; if(ni>=0&&nj>=0&&ni<n&&nj<n && g[
nj]]; }
            ans=Math.max(ans,sum);
        }
        return ans==0? n*n : ans;
    }
    private int dfs(int[][] g,int i,int j,int id){ if(i<0||j<0||i>=g.length||j>=g[0].lengt
g,i+1,j,id)+dfs(g,i-1,j,id)+dfs(g,i,j+1,id)+dfs(g,i,j-1,id); }
}
// Time: O(n^2)
```

Time & Space Complexity

Time: O(n^2).

Find All People With Secret — Hard

Find All People With Secret — Hard

Source: LeetCode 2092 | Link: https://leetcode.com/problems/find-all-people-with-secret/

Problem statement

At time 0, person 0 and firstPerson know a secret. Meetings[i] = [x,y,time] indicates a meeting at 'time'. People who meet at same time can spread secret among connected component. Return list of people who know secret after all meetings.

Java method signature

```
public List<Integer> findAllPeople(int n, int[][] meetings, int firstPerson)
```

Example test case(s)

```
Example per LeetCode; use grouping by time + DSU with rollback.
```

Optimized Java solution (concise)

```
class Solution {
    static class DSU{ int[] p; DSU(int n){ p=new int[n]; for(int i=0;i<n;i++) p[i]=i; } in
 } void union(int a,int b){ a=find(a); b=find(b); if(a!=b) p[a]=b; } }
    public java.util.List<Integer> findAllPeople(int n,int[][] meetings,int firstPerson){
        java.util.Map<Integer, java.util.List<int[]>> byT=new java.util.HashMap<>();
        for(int[] m:meetings) byT.computeIfAbsent(m[2],k->new java.util.ArrayList<>()).add
        DSU d=new DSU(n); d.union(0, firstPerson);
        java.util.List<Integer> times=new java.util.ArrayList<>(byT.keySet()); java.util.C
        for(int t: times){
            java.util.List<int[]> list=byT.get(t); java.util.Set<Integer> parts=new java.u
            for(int[] e:list){ d.union(e[0],e[1]); parts.add(e[0]); parts.add(e[1]); }
            for(int p: parts) if(d.find(p)!=d.find(0)) d.p[p]=p; // rollback isolated unio
        }
        java.util.List<Integer> ans=new java.util.ArrayList<>(); for(int i=0;i<n;i++) if(d
    }
}
// Time: O(M log M + M α(N))
```

Time & Space Complexity

Time: O(M log M + M α(N)).

Accounts Merge — Medium

Accounts Merge — Medium

Source: LeetCode 721 | Link: https://leetcode.com/problems/accounts-merge/

Problem statement

Accounts[i] = [name,email1,email2,...]. Merge accounts that belong to same person (share any email). Return merged accounts with emails sorted and name first.

Java method signature

```
public List<List<String>> accountsMerge(List<List<String>> accounts)
```

Example test case(s)

```
Input: [["John","a@mail","b@mail"],["John","b@mail","c@mail"]] -> Output: [["John","a@mail
```

Optimized Java solution (concise)

```
class Solution {
    public java.util.List<java.util.List<String>> accountsMerge(java.util.List<java.util.L
        java.util.Map<String,String> owner=new java.util.HashMap<>(); java.util.Map<String
        for(java.util.List<String> acc: accounts){ String name=acc.get(0); for(int i=1;i<a
.get(i)); owner.put(acc.get(i), name); } }
        for(java.util.List<String> acc: accounts) for(int i=2;i<acc.size();i++) union(pare
        java.util.Map<String, java.util.TreeSet<String>> groups=new java.util.HashMap<>();
        for(String email: parent.keySet()) groups.computeIfAbsent(find(parent,email), k->n
        java.util.List<java.util.List<String>> res=new java.util.ArrayList<>();
        for(String p: groups.keySet()){ java.util.List<String> list=new java.util.ArrayLis
get(p)); res.add(list); }
        return res;
    }
    private String find(java.util.Map<String,String> p,String x){ return p.get(x).equals(x
    private void union(java.util.Map<String,String> p,String a,String b){ a=find(p,a); b=f
}
// Time: O(total emails * α(N))
```

Time & Space Complexity

Time: O(total emails * α(N)).

Find the City With the Smallest Number of Neighbors at a Threshold Distance

Find the City With the Smallest Number of Neighbors at a Threshold Distance — Medium

Source: LeetCode 1334 | Link: https://leetcode.com/problems/find-the-city-with-the-smallest-number-of-neighbors-at-a-threshold-

Problem statement

Given n nodes and weighted edges, find the city with the smallest number of reachable cities within distanceThreshold. If tie choose largest index. Use Floyd-Warshall or Dijkstra per node.

Java method signature

```
public int findTheCity(int n, int[][] edges, int distanceThreshold)
```

Example test case(s)

```
Typical LeetCode examples.
```

Optimized Java solution (concise)

```
class Solution {
    public int findTheCity(int n,int[][] edges,int T){
        int INF=1_000_000_000; int[][] d=new int[n][n];
        for(int i=0;i<n;i++){ java.util.Arrays.fill(d[i],INF); d[i][i]=0; }
        for(int[] e: edges){ d[e[0]][e[1]]=Math.min(d[e[0]][e[1]], e[2]); d[e[1]][e[0]]=Ma
        for(int k=0;k<n;k++) for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(d[i][k]+d[k][j]<
        int best=-1, cntMin=1_000_000;
        for(int i=0;i<n;i++){ int cnt=0; for(int j=0;j<n;j++) if(d[i][j]<=T) cnt++; if(cnt
        return best;
    }
}
// Time: O(n^3) or use Dijkstra per node
```

Time & Space Complexity

Time: O(n^3) for Floyd-Warshall.

Network Delay Time — Medium

Network Delay Time — Medium

Source: LeetCode 743 | Link: https://leetcode.com/problems/network-delay-time/

Problem statement

Given times directed edges [u,v,w], n nodes and source k, return time it takes for all nodes to receive signal (max shortest path). Return -1 if unreachable. Use Dijkstra.

Java method signature

```
public int networkDelayTime(int[][] times, int n, int k)
```

Example test case(s)

```
Input: times=[[2,1,1],[2,3,1],[3,4,1]], n=4, k=2 -> Output: 2
```

Optimized Java solution (concise)

```
class Solution {
    public int networkDelayTime(int[][] times, int n, int k){
        java.util.List<java.util.List<int[]>> adj=new java.util.ArrayList<>(); for(int i=0
        for(int[] t:times) adj.get(t[0]).add(new int[]{t[1],t[2]});
        int[] dist=new int[n+1]; java.util.Arrays.fill(dist,Integer.MAX_VALUE); dist[k]=0;
        java.util.PriorityQueue<int[]> pq=new java.util.PriorityQueue<>((a,b)->a[1]-b[1]);
        while(!pq.isEmpty()){
            int[] cur=pq.poll(); int u=cur[0], d=cur[1]; if(d>dist[u]) continue;
            for(int[] e: adj.get(u)) if(dist[e[0]]>d+e[1]){ dist[e[0]]=d+e[1]; pq.offer(ne
        }
        int ans=0; for(int i=1;i<=n;i++){ if(dist[i]==Integer.MAX_VALUE) return -1; ans=Ma
    }
}
// Time: O(E log V)
```

Time & Space Complexity

Time: O(E log V).

Cheapest Flights Within K Stops — Medium

Cheapest Flights Within K Stops — Medium

Source: LeetCode 787 | Link: https://leetcode.com/problems/cheapest-flights-within-k-stops/

Problem statement

Find cheapest price from src to dst with at most K stops. Use Bellman-Ford relaxation for K+1 iterations or BFS/Dijkstra with stops state.

Java method signature

```
public int findCheapestPrice(int n, int[][] flights, int src, int dst, int K)
```

Example test case(s)

```
Input: n=3, flights=[[0,1,100],[1,2,100],[0,2,500]], src=0, dst=2, K=1 -> Output: 200
```

Optimized Java solution (concise)

```
class Solution {
    public int findCheapestPrice(int n,int[][] flights,int src,int dst,int K){
        int[] dist=new int[n]; java.util.Arrays.fill(dist,Integer.MAX_VALUE); dist[src]=0;
        for(int i=0;i<=K;i++){
            int[] tmp=dist.clone();
            for(int[] f: flights) if(dist[f[0]]!=Integer.MAX_VALUE) tmp[f[1]]=Math.min(tmp
            dist=tmp;
        }
        return dist[dst]==Integer.MAX_VALUE?-1:dist[dst];
    }
}
// Time: O(K*E)
```

Time & Space Complexity

Time: O(K*E).

Shortest Path in DAG — Medium

Shortest Path in DAG — Medium

Source: GeeksforGeeks (DAG shortest path) | Link: https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/

Problem statement

Given a weighted DAG and source node, compute shortest distances to all vertices using topological order: output INF for unreachable nodes.

Java method signature

```
public int[] shortestPath(int N, int M, int[][] edges, int src)
```

Example test case(s)

```
DAG example - shortest distances computed.
```

Optimized Java solution (concise)

```
class Solution {
    public int[] shortestPath(int N,int M,int[][] edges,int src){
        java.util.List<java.util.List<int[]>> adj=new java.util.ArrayList<>(); for(int i=0
        for(int[] e:edges) adj.get(e[0]).add(new int[]{e[1],e[2]});
        int[] indeg=new int[N]; for(int u=0;u<N;u++) for(int[] e:adj.get(u)) indeg[e[0]]++
        java.util.ArrayDeque<Integer> q=new java.util.ArrayDeque<>(); for(int i=0;i<N;i++)
        java.util.List<Integer> topo=new java.util.ArrayList<>(); while(!q.isEmpty()){ int
if(--indeg[e[0]]==0) q.offer(e[0]); }
        int INF=1_000_000_000; int[] dist=new int[N]; java.util.Arrays.fill(dist,INF); dis
        for(int u: topo) if(dist[u]!=INF) for(int[] e: adj.get(u)) dist[e[0]]=Math.min(dis
        return dist;
    }
}
// Time: O(V+E)
```

Time & Space Complexity

Time: O(V+E).

Longest Path in DAG — Medium

Longest Path in DAG — Medium

Source: GeeksforGeeks (DAG longest path) | Link: https://www.geeksforgeeks.org/find-longest-path-directed-acyclic-graph/

Problem statement

Given a weighted DAG, find the longest path distances from source.
Use topological order and relax with max instead of min.

Java method signature

```
public int[] longestPath(int N, int M, int[][] edges, int src)
```

Example test case(s)

```
DAG example - longest distances.
```

Optimized Java solution (concise)

```
class Solution {
    public int[] longestPath(int N,int M,int[][] edges,int src){
        java.util.List<java.util.List<int[]>> adj=new java.util.ArrayList<>(); for(int i=0
        for(int[] e:edges) adj.get(e[0]).add(new int[]{e[1],e[2]});
        int[] indeg=new int[N]; for(int u=0;u<N;u++) for(int[] e:adj.get(u)) indeg[e[0]]++
        java.util.ArrayDeque<Integer> q=new java.util.ArrayDeque<>(); for(int i=0;i<N;i++)
        java.util.List<Integer> topo=new java.util.ArrayList<>(); while(!q.isEmpty()){ int
if(--indeg[e[0]]==0) q.offer(e[0]); }
        int NEG=-1_000_000_000; int[] dist=new int[N]; java.util.Arrays.fill(dist,NEG); di
        for(int u: topo) if(dist[u]!=NEG) for(int[] e: adj.get(u)) dist[e[0]]=Math.max(dis
        return dist;
    }
}
// Time: O(V+E)
```

Time & Space Complexity

Time: O(V+E).

Design Graph With Shortest Path Calculator — Hard

Design Graph With Shortest Path Calculator — Hard

Source: LeetCode 2642 | Link: https://leetcode.com/problems/design-graph-with-shortest-path-calculator/

Problem statement

Design a graph supporting addEdge(u,v,w) and shortestPath(node1,node2) queries. Use adjacency list and run Dijkstra per query.

Java method signature

```
public class Graph { public Graph(int n, int[][] edges) {} public void addEdge(int
[] e) {} public int shortestPath(int s, int t) { return -1; } }
```

Example test case(s)

```
See LeetCode for example sequence of operations and outputs.
```

Optimized Java solution (concise)

```
class Graph {
    java.util.List<java.util.List<int[]>> adj;
    public Graph(int n,int[][] edges){ adj=new java.util.ArrayList<>(); for(int i=0;i<n;i+
 e:edges) adj.get(e[0]).add(new int[]{e[1],e[2]}); }
    public void addEdge(int[] e){ adj.get(e[0]).add(new int[]{e[1],e[2]}); }
    public int shortestPath(int s,int t){ int n=adj.size(); int[] dist=new int[n]; java.ut
        java.util.PriorityQueue<int[]> pq=new java.util.PriorityQueue<>((a,b)->a[1]-b[1]);
        while(!pq.isEmpty()){ int[] cur=pq.poll(); int u=cur[0], d=cur[1]; if(d>dist[u]) c
            for(int[] e: adj.get(u)) if(dist[e[0]]>d+e[1]){ dist[e[0]]=d+e[1]; pq.offer(ne
        }
        return dist[t]==Integer.MAX_VALUE?-1:dist[t];
    }
}
// Dijkstra per query
```

Time & Space Complexity

Time: Dijkstra per query O(E log V).

Reconstruct Itinerary — Hard

Reconstruct Itinerary — Hard

Source: LeetCode 332 | Link: https://leetcode.com/problems/reconstruct-itinerary/

Problem statement

Given tickets [from,to], reconstruct itinerary starting at 'JFK' using all tickets exactly once. If multiple results, return lexicographically smallest itinerary. Use Hierholzer's algorithm with priority queues.

Java method signature

```
public List<String> findItinerary(List<List<String>> tickets)
```

Example test case(s)

```
Input: [["MUC","LHR"],["JFK","MUC"],["SFO","SJC"],["LHR","SFO"]] -> Output: ["JFK","MUC","
```

Optimized Java solution (concise)

```
class Solution {
    java.util.Map<String, java.util.PriorityQueue<String>> adj=new java.util.HashMap<>();
    public java.util.List<String> findItinerary(java.util.List<java.util.List<String>> tic
        for(java.util.List<String> t: tickets) adj.computeIfAbsent(t.get(0), k->new java.u
        java.util.LinkedList<String> ans=new java.util.LinkedList<>(); dfs("JFK", ans); re
    }
    private void dfs(String u, java.util.LinkedList<String> ans){
        java.util.PriorityQueue<String> pq = adj.get(u);
        while(pq!=null && !pq.isEmpty()) dfs(pq.poll(), ans);
        ans.addFirst(u);
    }
}
// Time: O(E log E) due to PQ operations
```

Time & Space Complexity

Time: O(E log E).

Min Cost to Connect All Points — Medium

Min Cost to Connect All Points — Medium

Source: LeetCode 1584 | Link: https://leetcode.com/problems/min-cost-to-connect-all-points/

Problem statement

Given points on 2D plane, cost between i and j is Manhattan distance. Return minimum cost to connect all points (MST). Use Prim's algorithm O(n^2) or Prim with PQ.

Java method signature

```
public int minCostConnectPoints(int[][] points)
```

Example test case(s)

```
Input: [[0,0],[2,2],[3,10],[5,2],[7,0]] -> Output: 20
```

Optimized Java solution (concise)

```
class Solution {
    public int minCostConnectPoints(int[][] p){
        int n=p.length; boolean[] vis=new boolean[n]; int[] minD=new int[n]; java.util.Arr
        int ans=0;
        for(int it=0; it<n; it++){
            int u=-1; for(int i=0;i<n;i++) if(!vis[i] && (u==-1 || minD[i]<minD[u])) u=i;
            vis[u]=true; ans+= (minD[u]==Integer.MAX_VALUE?0:minD[u]);
            for(int v=0; v<n; v++) if(!vis[v]){ int w=Math.abs(p[u][0]-p[v][0])+Math.abs(p
        }
        return ans;
    }
}
// Time: O(n^2)
```

Time & Space Complexity

Time: O(n^2).

Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree — Mediu

Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree — Medium

Source: LeetCode 1489 | Link: https://leetcode.com/problems/find-critical-and-pseudo-critical-edges-in-minimum-spanning-tree/

Problem statement

Given a connected weighted graph, determine which edges are critical (in all MSTs) and pseudo-critical (in some MST). Approach: Kruskal baseline MST cost, then test exclusion/inclusion per edge with DSU.

Java method signature

```
public List<List<Integer>> findCriticalAndPseudoCriticalEdges(int n, int[][] edges
```

Example test case(s)

```
See LeetCode examples.
```

Optimized Java solution (concise)

```
class Solution {
    static class DSU{ int[] p,r; DSU(int n){ p=new int[n]; r=new int[n]; for(int i=0;i<n;i
[x]=find(p[x])); } boolean union(int a,int b){ a=find(a); b=find(b); if(a==b) return false
=r[b]) r[a]++; return true; } }
    int mst(int n,int[][] e,int skip,int force){
        DSU d=new DSU(n); int cost=0, cnt=0;
        if(force!=-1){ d.union(e[force][0],e[force][1]); cost+=e[force][2]; cnt++; }
        for(int i=0;i<e.length;i++){ if(i==skip) continue; if(d.union(e[i][0],e[i][1])){ c
        return cnt==n-1?cost:1_000_000_000;
    }
    public java.util.List<java.util.List<Integer>> findCriticalAndPseudoCriticalEdges(int
        int m=edges.length; int[][] e=new int[m][4]; for(int i=0;i<m;i++){ e[i][0]=edges[i
i][3]=i; }
        java.util.Arrays.sort(e,(a,b)->Integer.compare(a[2],b[2])); int base=mst(n,e,-1,-1
        java.util.List<Integer> critical=new java.util.ArrayList<>(), pseudo=new java.util
        for(int i=0;i<m;i++){ if(mst(n,e,i,-1)>base) critical.add(e[i][3]); else if(mst(n,
        java.util.List<java.util.List<Integer>> ans=new java.util.ArrayList<>(); ans.add(c
    }
}
// Time: O(M * (M α(N)))
```

Time & Space Complexity

Time: O(M * (M α(N)))