Arrays

1) Merge Intervals (56)

Sample 1 Input: `intervals = [[1,3],[2,6],[8,10],[15,18]]` Output: `[[1,6],[8,10],[15,18]]`

Sample 2 Input: `intervals = [[1,4],[4,5]]` Output: `[[1,5]]`

Code:

```java
import java.util.*;
class Solution {
    public int[][] merge(int[][] intervals) {
        if (intervals == null || intervals.length == 0) return new int[0][];
        Arrays.sort(intervals, (a,b)->a[0]-b[0]);
        List<int[]> res = new ArrayList<>();
        int[] cur = intervals[0];
        for (int i=1;i<intervals.length;i++){
            if (intervals[i][0] <= cur[1]) {
                cur[1] = Math.max(cur[1], intervals[i][1]);
            } else {
                res.add(cur);
                cur = intervals[i];
            }
        }
        res.add(cur);
        return res.toArray(new int[res.size()][]);
    }
}
```

---

2) Subarray Sum Equals K (560)

Sample 1 Input: `nums = [1,1,1], k = 2` Output: `2`

Sample 2 Input: `nums = [1,2,3], k = 3` Output: `2`

Code:

```java
import java.util.*;
class Solution {
    public int subarraySum(int[] nums, int k) {
        Map<Integer,Integer> map = new HashMap<>();
        map.put(0,1);
        int sum=0, count=0;
        for (int n: nums){
            sum += n;
            count += map.getOrDefault(sum - k, 0);
```

```
            map.put(sum, map.getOrDefault(sum,0)+1);
        }
        return count;
    }
}
```

---

3) Meeting Rooms II (253)

Sample 1 Input: `intervals = [[0,30],[5,10],[15,20]]` Output: `2`

Sample 2 Input: `intervals = [[7,10],[2,4]]` Output: `1`

Code:

```
import java.util.*;
class Solution {
    public int minMeetingRooms(int[][] intervals) {
        if (intervals.length==0) return 0;
        Arrays.sort(intervals, (a,b)->a[0]-b[0]);
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(intervals[0][1]);
        for (int i=1;i<intervals.length;i++){
            if (intervals[i][0] >= pq.peek()) pq.poll();
            pq.add(intervals[i][1]);
        }
        return pq.size();
    }
}
```

---

4) Container With Most Water (11)

Sample 1 Input: `height = [1,8,6,2,5,4,8,3,7]` Output: `49`

Sample 2 Input: `height = [1,1]` Output: `1`

Code:

```
class Solution {
    public int maxArea(int[] height) {
        int l=0, r=height.length-1;
        int ans=0;
        while(l<r){
            ans = Math.max(ans, Math.min(height[l],height[r])*(r-l));
            if (height[l] < height[r]) l++; else r--;
        }
        return ans;
```

```
    }
}
```

---

5) Group Anagrams (49)

Sample 1 Input: `["eat","tea","tan","ate","nat","bat"]` Output: `[["eat","tea","ate"],["tan","nat"],["bat"]]` (order may vary)

Sample 2 Input: `[""]` Output: `[[""]]`

Code:

```java
import java.util.*;
class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> map = new HashMap<>();
        for (String s: strs){
            char[] ch = s.toCharArray();
            Arrays.sort(ch);
            String key = new String(ch);
            map.computeIfAbsent(key, k->new ArrayList<>()).add(s);
        }
        return new ArrayList<>(map.values());
    }
}
```

---

6) 3Sum (15)

Sample 1 Input: `nums = [-1,0,1,2,-1,-4]` Output: `[[-1,-1,2],[-1,0,1]]` (order may vary)

Sample 2 Input: `nums = []` Output: `[]`

Code:

```java
import java.util.*;
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);
        for (int i=0;i<nums.length-2;i++){
            if (i>0 && nums[i]==nums[i-1]) continue;
            int l=i+1, r=nums.length-1;
            while (l<r){
                int sum = nums[i]+nums[l]+nums[r];
                if (sum==0){
```

```
                    res.add(Arrays.asList(nums[i],nums[l],nums[r]));
                    while (l<r && nums[l]==nums[l+1]) l++;
                    while (l<r && nums[r]==nums[r-1]) r--;
                    l++; r--;
                } else if (sum < 0) l++; else r--;
            }
        }
        return res;
    }
}
```

---

7) Maximum Number of Events That Can Be Attended (1353)

Sample 1 Input: `events = [[1,2],[2,3],[3,4]]` Output: `3`

Sample 2 Input: `events = [[1,2],[2,2],[3,3]]` Output: `3`

Code:

```
import java.util.*;
class Solution {
    public int maxEvents(int[][] events) {
        Arrays.sort(events, (a,b)->a[0]-b[0]);
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        int i=0, day=0, res=0, n=events.length;
        while(i<n || !pq.isEmpty()){
            if (pq.isEmpty()) day = Math.max(day, events[i][0]);
            while(i<n && events[i][0] <= day){
                pq.add(events[i][1]);
                i++;
            }
            if (!pq.isEmpty()){
                pq.poll();
                res++;
                day++;
                while(!pq.isEmpty() && pq.peek() < day) pq.poll();
            }
        }
        return res;
    }
}
```

---

8) Next Permutation (31)

Sample 1 Input: `nums = [1,2,3]` Output: `[1,3,2]`

Sample 2 Input: `nums = [3,2,1]` Output: `[1,2,3]`

Code:

```java
import java.util.*;
class Solution {
    public void nextPermutation(int[] nums) {
        int i = nums.length-2;
        while(i>=0 && nums[i] >= nums[i+1]) i--;
        if (i>=0) {
            int j = nums.length-1;
            while(nums[j] <= nums[i]) j--;
            swap(nums, i, j);
        }
        reverse(nums, i+1, nums.length-1);
    }
    private void swap(int[] a,int i,int j){int t=a[i];a[i]=a[j];a[j]=t;}
    private void reverse(int[] a,int l,int r){while(l<r){swap(a,l++,r--);}}
}
```

---

9) Diagonal Traverse (498)

Sample 1 Input: `mat = [[1,2,3],[4,5,6],[7,8,9]]` Output: `[1,2,4,7,5,3,6,8,9]`

Sample 2 Input: `mat = [[1,2],[3,4]]` Output: `[1,2,3,4]`

Code:

```java
import java.util.*;
class Solution {
    public int[] findDiagonalOrder(int[][] mat) {
        if (mat == null || mat.length==0) return new int[0];
        int m=mat.length, n=mat[0].length;
        int[] res = new int[m*n];
        int idx=0;
        for (int s=0; s<=m+n-2; s++){
            List<Integer> temp = new ArrayList<>();
            for (int i=0;i<m;i++){
                int j = s - i;
                if (j>=0 && j<n) temp.add(mat[i][j]);
            }
            if (s%2==0){
                for (int k=temp.size()-1;k>=0;k--) res[idx++] = temp.get(k);
            } else {
                for (int v: temp) res[idx++] = v;
            }
        }
```

```
        return res;
    }
}
```

---

10) Find Peak Element (162)

Sample 1 Input: `nums = [1,2,3,1]` Output: `2` (index of peak)

Sample 2 Input: `nums = [1,2,1,3,5,6,4]` Output: `1` or `5` (either index works)

Code:

```
class Solution {
    public int findPeakElement(int[] nums) {
        int l=0, r=nums.length-1;
        while (l<r){
            int m = l + (r-l)/2;
            if (nums[m] > nums[m+1]) r = m;
            else l = m+1;
        }
        return l;
    }
}
```

---

11) Koko Eating Bananas (875)

Sample 1 Input: `piles = [3,6,7,11], h = 8` Output: `4`

Sample 2 Input: `piles = [30,11,23,4,20], h = 5` Output: `30`

Code:

```
class Solution {
    public int minEatingSpeed(int[] piles, int h) {
        int lo=1, hi=0;
        for (int p: piles) hi = Math.max(hi, p);
        while (lo<hi){
            int mid = lo + (hi-lo)/2;
            if (canFinish(piles, mid, h)) hi = mid;
            else lo = mid+1;
        }
        return lo;
    }
    private boolean canFinish(int[] piles, int k, int h){
        long hours=0;
        for (int p: piles) hours += (p + k -1)/k;
```

```
        return hours <= h;
    }
}
```

---

12) Kth Largest Element in an Array (215)

Sample 1 Input: `nums=[3,2,1,5,6,4], k=2` Output: `5`

Sample 2 Input: `nums=[3,2,3,1,2,4,5,5,6], k=4` Output: `4`

Code (Quickselect):

```
import java.util.*;
class Solution {
    public int findKthLargest(int[] nums, int k) {
        return quickselect(nums, 0, nums.length-1, nums.length - k);
    }
    private int quickselect(int[] a, int l, int r, int k){
        if (l==r) return a[l];
        int pivot = a[l + (r-l)/2];
        int i=l, j=r;
        while(i<=j){
            while(a[i]<pivot) i++;
            while(a[j]>pivot) j--;
            if (i<=j) swap(a,i++,j--);
        }
        if (k<=j) return quickselect(a,l,j,k);
        if (k>=i) return quickselect(a,i,r,k);
        return a[k];
    }
    private void swap(int[] a,int i,int j){int t=a[i];a[i]=a[j];a[j]=t;}
}
```

---

13) Maximum Subarray (53)

Sample 1 Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]` Output: `6` (subarray [4,-1,2,1])

Sample 2 Input: `nums = [1]` Output: `1`

Code:

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSoFar = nums[0], cur = nums[0];
        for (int i=1;i<nums.length;i++){
```

```
            cur = Math.max(nums[i], cur + nums[i]);
            maxSoFar = Math.max(maxSoFar, cur);
        }
        return maxSoFar;
    }
}
```

---

14) Spiral Matrix (54)

Sample 1 Input: `matrix = [[1,2,3],[4,5,6],[7,8,9]]` Output: `[1,2,3,6,9,8,7,4,5]`

Sample 2 Input: `matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]` Output: `[1,2,3,4,8,12,11,10,9,5,6,7]`

Code:

```
import java.util.*;
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> res = new ArrayList<>();
        if (matrix==null || matrix.length==0) return res;
        int top=0, bottom=matrix.length-1, left=0, right=matrix[0].length-1;
        while(top<=bottom && left<=right){
            for (int j=left;j<=right;j++) res.add(matrix[top][j]);
            top++;
            for (int i=top;i<=bottom;i++) res.add(matrix[i][right]);
            right--;
            if (top<=bottom){
                for (int j=right;j>=left;j--) res.add(matrix[bottom][j]);
                bottom--;
            }
            if (left<=right){
                for (int i=bottom;i>=top;i--) res.add(matrix[i][left]);
                left++;
            }
        }
        return res;
    }
}
```

---

15) Rotate Image (48)

Sample 1 Input: `matrix=[[1,2,3],[4,5,6],[7,8,9]]` Output: `[[7,4,1],[8,5,2],[9,6,3]]`

Sample 2 Input: `matrix=[[1]]` Output: `[[1]]`

Code:

```java
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        for (int i=0;i<n;i++){
            for (int j=i;j<n;j++){
                int t = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = t;
            }
        }
        for (int i=0;i<n;i++){
            for (int j=0;j<n/2;j++){
                int t = matrix[i][j];
                matrix[i][j] = matrix[i][n-1-j];
                matrix[i][n-1-j] = t;
            }
        }
    }
}
```

---

16) Coin Change (322)

Sample 1 Input: `coins=[1,2,5], amount=11` Output: `3` (5+5+1)

Sample 2 Input: `coins=[2], amount=3` Output: `-1`

Code:

```java
import java.util.*;
class Solution {
    public int coinChange(int[] coins, int amount) {
        int INF = amount+1;
        int[] dp = new int[amount+1];
        Arrays.fill(dp, INF);
        dp[0] = 0;
        for (int c: coins) {
            for (int i=c;i<=amount;i++){
                dp[i] = Math.min(dp[i], dp[i-c] + 1);
            }
        }
        return dp[amount] > amount ? -1 : dp[amount];
    }
}
```

---

17) Find First and Last Position of Element in Sorted Array (34)

Sample 1 Input: `nums=[5,7,7,8,8,10], target=8` Output: `[3,4]`

Sample 2 Input: `nums=[5,7,7,8,8,10], target=6` Output: `[-1,-1]`

Code:

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int left = findBound(nums, target, true);
        int right = findBound(nums, target, false);
        return new int[]{left, right};
    }
    private int findBound(int[] a, int t, boolean isLeft){
        int l=0, r=a.length-1, ans=-1;
        while (l<=r){
            int m = l + (r-l)/2;
            if (a[m] == t) { ans = m; if (isLeft) r = m-1; else l = m+1; }
            else if (a[m] < t) l = m+1;
            else r = m-1;
        }
        return ans;
    }
}
```

---

18) Gas Station (134)

Sample 1 Input: `gas=[1,2,3,4,5], cost=[3,4,5,1,2]` Output: `3`

Sample 2 Input: `gas=[2,3,4], cost=[3,4,3]` Output: `-1`

Code:

```
class Solution {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int total=0, sum=0, start=0;
        for (int i=0;i<gas.length;i++){
            int diff = gas[i]-cost[i];
            total += diff;
            sum += diff;
            if (sum < 0) { start = i+1; sum = 0; }
        }
        return total < 0 ? -1 : start;
    }
}
```

---

19) Word Search (79)

Sample 1 Input: `board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word="ABCCED"` Output: `true`

Sample 2 Input: `board = [["a"]], word="aa"` Output: `false`

Code:

```
class Solution {
    public boolean exist(char[][] board, String word) {
        int m=board.length, n=board[0].length;
        boolean[][] vis = new boolean[m][n];
        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (dfs(board, word, 0, i, j, vis)) return true;
            }
        }
        return false;
    }
    private boolean dfs(char[][] b, String w, int idx, int i, int j, boolean[][] vis){
        if (idx==w.length()) return true;
        if (i<0||j<0||i>=b.length||j>=b[0].length) return false;
        if (vis[i][j] || b[i][j] != w.charAt(idx)) return false;
        vis[i][j] = true;
        boolean found = dfs(b,w,idx+1,i+1,j,vis) || dfs(b,w,idx+1,i-1,j,vis)
            || dfs(b,w,idx+1,i,j+1,vis) || dfs(b,w,idx+1,i,j-1,vis);
        vis[i][j] = false;
        return found;
    }
}
```

---

20) Jump Game (55)

Sample 1 Input: `nums = [2,3,1,1,4]` Output: `true`

Sample 2 Input: `nums = [3,2,1,0,4]` Output: `false`

Code:

```
class Solution {
    public boolean canJump(int[] nums) {
        int reach = 0;
        for (int i=0;i<nums.length;i++){
            if (i > reach) return false;
            reach = Math.max(reach, i + nums[i]);
        }
        return true;
```

```
    }
}
```

---

21) Valid Sudoku (36)

Sample 1 Input: standard 9x9 valid sudoku board Output: `true`

Sample 2 Input: board with duplicate in row/col/box Output: `false`

Code:

```java
import java.util.*;
class Solution {
    public boolean isValidSudoku(char[][] board) {
        for (int i=0;i<9;i++){
            boolean[] row = new boolean[9];
            boolean[] col = new boolean[9];
            boolean[] box = new boolean[9];
            for (int j=0;j<9;j++){
                if (board[i][j] != '.') {
                    int idx = board[i][j]-'1';
                    if (row[idx]) return false;
                    row[idx] = true;
                }
                if (board[j][i] != '.') {
                    int idx = board[j][i]-'1';
                    if (col[idx]) return false;
                    col[idx] = true;
                }
                int br = (i/3)*3 + j/3;
                int bc = (i%3)*3 + j%3;
                if (board[br][bc] != '.') {
                    int idx = board[br][bc]-'1';
                    if (box[idx]) return false;
                    box[idx] = true;
                }
            }
        }
        return true;
    }
}
```

---

22) Best Time to Buy and Sell Stock II (122)

Sample 1 Input: `prices=[7,1,5,3,6,4]` Output: `7`

Sample 2 Input: `prices=[1,2,3,4,5]` Output: `4`

Code:

```
class Solution {
    public int maxProfit(int[] prices) {
        int profit=0;
        for (int i=1;i<prices.length;i++){
            if (prices[i] > prices[i-1]) profit += prices[i]-prices[i-1];
        }
        return profit;
    }
}
```

---

23) Longest Increasing Subsequence (300)

Sample 1 Input: `nums=[10,9,2,5,3,7,101,18]` Output: `4` (2,3,7,101)

Sample 2 Input: `nums=[0,1,0,3,2,3]` Output: `4`

Code:

```
import java.util.*;
class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums.length==0) return 0;
        ArrayList<Integer> tails = new ArrayList<>();
        for (int x: nums){
            int i = Collections.binarySearch(tails, x);
            if (i < 0) i = - (i+1);
            if (i == tails.size()) tails.add(x);
            else tails.set(i, x);
        }
        return tails.size();
    }
}
```

---

24) House Robber (198)

Sample 1 Input: `nums=[1,2,3,1]` Output: `4`

Sample 2 Input: `nums=[2,7,9,3,1]` Output: `12`

Code:

```
class Solution {
    public int rob(int[] nums) {
```

```
        int prev=0, curr=0;
        for (int n: nums){
            int temp = Math.max(curr, prev + n);
            prev = curr;
            curr = temp;
        }
        return curr;
    }
}
```

---

25) Subsets (78)

Sample 1 Input: `nums=[1,2,3]` Output: `[[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]` (order may vary)

Sample 2 Input: `nums=[]` Output: `[[]]`

Code:

```
import java.util.*;
class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        backtrack(nums, 0, new ArrayList<>(), res);
        return res;
    }
    private void backtrack(int[] nums, int idx, List<Integer> cur, List<List<Integer>> res){
        res.add(new ArrayList<>(cur));
        for (int i=idx;i<nums.length;i++){
            cur.add(nums[i]);
            backtrack(nums, i+1, cur, res);
            cur.remove(cur.size()-1);
        }
    }
}
```

Strings

---

1) Longest Substring Without Repeating Characters (3)

Sample 1 Input: `s = "abcabcbb"` Output: `3`

Sample 2 Input: `s = "bbbbb"` Output: `1`

Code:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int[] last = new int[256];
        Arrays.fill(last, -1);
        int res = 0, start = 0;
        for (int i = 0; i < s.length(); i++) {
            start = Math.max(start, last[s.charAt(i)] + 1);
            res = Math.max(res, i - start + 1);
            last[s.charAt(i)] = i;
        }
        return res;
    }
}
```

---

2) Longest Palindromic Substring (5)

Sample 1 Input: `s = "babad"` Output: `"bab"` (or `"aba"`)

Sample 2 Input: `s = "cbbd"` Output: `"bb"`

Code:

```
class Solution {
    public String longestPalindrome(String s) {
        if (s == null || s.length() < 1) return "";
        int start = 0, end = 0;
        for (int i = 0; i < s.length(); i++) {
            int len1 = expand(s, i, i);
            int len2 = expand(s, i, i+1);
            int len = Math.max(len1, len2);
            if (len > end - start + 1) {
                start = i - (len-1)/2;
                end = i + len/2;
            }
        }
        return s.substring(start, end+1);
    }
    private int expand(String s, int l, int r) {
        while (l>=0 && r<s.length() && s.charAt(l)==s.charAt(r)) { l--; r++; }
        return r - l - 1;
    }
}
```

---

3) Minimum Remove to Make Valid Parentheses (1249)

Sample 1 Input: `s = "a)b(c)d"` Output: `"ab(c)d"`

Sample 2 Input: `s = "))(("` Output: `""`

Code:

```
class Solution {
    public String minRemoveToMakeValid(String s) {
        StringBuilder sb = new StringBuilder(s);
        Stack<Integer> st = new Stack<>();
        for (int i=0;i<sb.length();i++){
            char c = sb.charAt(i);
            if (c=='(') st.push(i);
            else if (c==')') {
                if (st.isEmpty()) sb.setCharAt(i, '*');
                else st.pop();
            }
        }
        while(!st.isEmpty()) sb.setCharAt(st.pop(), '*');
        StringBuilder res = new StringBuilder();
        for (int i=0;i<sb.length();i++) if (sb.charAt(i)!='*') res.append(sb.charAt(i));
        return res.toString();
    }
}
```

---

4) Basic Calculator II (227)

Sample 1 Input: `s = "3+2*2"` Output: `7`

Sample 2 Input: `s = " 3/2 "` Output: `1`

Code:

```
class Solution {
    public int calculate(String s) {
        if (s==null) return 0;
        s = s.replace(" ", "");
        int n = s.length();
        Deque<Integer> st = new ArrayDeque<>();
        int num = 0;
        char sign = '+';
        for (int i=0;i<n;i++){
            char c = s.charAt(i);
            if (Character.isDigit(c)) num = num*10 + (c - '0');
            if (!Character.isDigit(c) || i == n-1) {
                if (sign=='+') st.push(num);
```

```
                else if (sign=='-') st.push(-num);
                else if (sign=='*') st.push(st.pop() * num);
                else if (sign=='/') st.push(st.pop() / num);
                sign = c;
                num = 0;
            }
        }
        int res = 0;
        for (int x: st) res += x;
        return res;
    }
}
```

---

5) Maximum Manhattan Distance After K Changes (3443)

Sample 1 Input: `s = "NWSE", k = 1` Output: `3`

Sample 2 Input: `s = "NSWWEW", k = 3` Output: `6`

Code:

```
class Solution {
    public int maxDistance(String s, int k) {
        return Math.max(Math.max(flip(s, k, "NE"), flip(s, k, "NW")),
                        Math.max(flip(s, k, "SE"), flip(s, k, "SW")));
    }
    private int flip(String s, int k, String dir) {
        int res = 0, pos = 0, opposite = 0;
        for (char c: s.toCharArray()){
            if (dir.indexOf(c) >= 0) pos++;
            else { pos--; opposite++; }
            res = Math.max(res, pos + 2 * Math.min(k, opposite));
        }
        return res;
    }
}
```

---

6) Generate Parentheses (22)

Sample 1 Input: `n = 3` Output: `["((()))","(()())","(())()","()(())","()()()"]`

Sample 2 Input: `n = 1` Output: `["()"]`

Code:

```java
import java.util.*;
class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> res = new ArrayList<>();
        backtrack(res, new StringBuilder(), 0, 0, n);
        return res;
    }
    private void backtrack(List<String> res, StringBuilder cur, int open, int close, int n)
        if (cur.length() == 2*n) { res.add(cur.toString()); return; }
        if (open < n) {
            cur.append('(');
            backtrack(res, cur, open+1, close, n);
            cur.deleteCharAt(cur.length()-1);
        }
        if (close < open) {
            cur.append(')');
            backtrack(res, cur, open, close+1, n);
            cur.deleteCharAt(cur.length()-1);
        }
    }
}
```

---

7) Decode String (394)

Sample 1 Input: `s = "3[a]2[bc]"` Output: `"aaabcbc"`

Sample 2 Input: `s = "3[a2[c]]"` Output: `"accaccacc"`

Code:

```java
import java.util.*;
class Solution {
    public String decodeString(String s) {
        Deque<Integer> counts = new ArrayDeque<>();
        Deque<StringBuilder> strs = new ArrayDeque<>();
        StringBuilder cur = new StringBuilder();
        int k = 0;
        for (char c: s.toCharArray()){
            if (Character.isDigit(c)) { k = k*10 + (c - '0'); }
            else if (c == '['){
                counts.push(k); strs.push(cur);
                k = 0; cur = new StringBuilder();
            } else if (c == ']'){
                int times = counts.pop();
                StringBuilder prev = strs.pop();
                for (int i=0;i<times;i++) prev.append(cur);
```

```
                cur = prev;
            } else cur.append(c);
        }
        return cur.toString();
    }
}
```

---

8) Longest Binary Subsequence Less Than or Equal to K (2311)

Sample 1 Input: `s = "1001010", k = 5` Output: `5`

Sample 2 Input: `s = "0010", k = 1` Output: `3`

Code:

```
class Solution {
    public int longestSubsequence(String s, int k) {
        int zeros = 0;
        for (char c: s.toCharArray()) if (c=='0') zeros++;
        int value = 0, len = 0;
        // Add ones from right if possible
        for (int i = s.length()-1; i>=0; i--) {
            if (s.charAt(i)=='0') { len++; continue; }
            if (value + (1 << len) <= k) {
                value += (1 << len);
                len++;
            }
        }
        return Math.min(s.length(), zeros + len);
    }
}
```

---

9) Reorganize String (767)

Sample 1 Input: `s = "aab"` Output: `"aba"`

Sample 2 Input: `s = "aaab"` Output: `""`

Code:

```
import java.util.*;
class Solution {
    public String reorganizeString(String s) {
        int[] cnt = new int[26];
        for (char c: s.toCharArray()) cnt[c-'a']++;
        PriorityQueue<int[]> pq = new PriorityQueue<>((a,b)->b[1]-a[1]);
```

```
        for (int i=0;i<26;i++) if (cnt[i]>0) pq.add(new int[]{i, cnt[i]});
        StringBuilder res = new StringBuilder();
        while (pq.size()>=2){
            int[] a = pq.poll(), b = pq.poll();
            res.append((char)('a'+a[0])).append((char)('a'+b[0]));
            if (--a[1]>0) pq.add(a);
            if (--b[1]>0) pq.add(b);
        }
        if (!pq.isEmpty()){
            int[] last = pq.poll();
            if (last[1] > 1) return "";
            res.append((char)('a'+last[0]));
        }
        return res.toString();
    }
}
```

---

10) Largest Number (179)

Sample 1 Input: `nums = [10,2]` Output: `"210"`

Sample 2 Input: `nums = [3,30,34,5,9]` Output: `"9534330"`

Code:

```
import java.util.*;
class Solution {
    public String largestNumber(int[] nums) {
        String[] arr = new String[nums.length];
        for (int i=0;i<nums.length;i++) arr[i] = String.valueOf(nums[i]);
        Arrays.sort(arr, (a,b) -> (b+a).compareTo(a+b));
        if (arr[0].equals("0")) return "0";
        StringBuilder sb = new StringBuilder();
        for (String s: arr) sb.append(s);
        return sb.toString();
    }
}
```

---

11) String Compression (443)

Sample 1 Input: `chars = ["a","a","b","b","c","c","c"]` Output: `6` (chars becomes [“a”,“2”,“b”,“2”,“c”,“3”])

Sample 2 Input: `chars = ["a"]` Output: `1`

Code:

```
class Solution {
    public int compress(char[] chars) {
        int write = 0, i = 0;
        while (i < chars.length) {
            char ch = chars[i];
            int j = i;
            while (j < chars.length && chars[j] == ch) j++;
            chars[write++] = ch;
            int count = j - i;
            if (count > 1) {
                for (char c: String.valueOf(count).toCharArray()) chars[write++] = c;
            }
            i = j;
        }
        return write;
    }
}
```

---

12) Longest Repeating Character Replacement (424)

Sample 1 Input: `s = "AABABBA", k = 1` Output: `4`

Sample 2 Input: `s = "ABAB", k = 2` Output: `4`

Code:

```
class Solution {
    public int characterReplacement(String s, int k) {
        int[] count = new int[26];
        int maxCount = 0, left = 0, res = 0;
        for (int right=0; right<s.length(); right++){
            maxCount = Math.max(maxCount, ++count[s.charAt(right)-'A']);
            while (right - left + 1 - maxCount > k) {
                count[s.charAt(left++)-'A']--;
            }
            res = Math.max(res, right - left + 1);
        }
        return res;
    }
}
```

---

13) Remove K Digits (402)

Sample 1 Input: `num = "1432219", k = 3` Output: `"1219"`

Sample 2 Input: `num = "10200", k = 1` Output: `"200"`

Code:

```
import java.util.*;
class Solution {
    public String removeKdigits(String num, int k) {
        Deque<Character> st = new ArrayDeque<>();
        for (char c: num.toCharArray()) {
            while (!st.isEmpty() && k > 0 && st.peekLast() > c) { st.pollLast(); k--; }
            st.addLast(c);
        }
        while (k-- > 0) st.pollLast();
        StringBuilder sb = new StringBuilder();
        while (!st.isEmpty()) sb.append(st.pollFirst());
        while (sb.length() > 1 && sb.charAt(0) == '0') sb.deleteCharAt(0);
        return sb.length() == 0 ? "0" : sb.toString();
    }
}
```

---

14) Edit Distance (72)

Sample 1 Input: `word1 = "horse", word2 = "ros"` Output: `3`

Sample 2 Input: `word1 = "intention", word2 = "execution"` Output: `5`

Code:

```
class Solution {
    public int minDistance(String word1, String word2) {
        int m = word1.length(), n = word2.length();
        int[][] dp = new int[m+1][n+1];
        for (int i=0;i<=m;i++) dp[i][0] = i;
        for (int j=0;j<=n;j++) dp[0][j] = j;
        for (int i=1;i<=m;i++){
            for (int j=1;j<=n;j++){
                if (word1.charAt(i-1) == word2.charAt(j-1)) dp[i][j] = dp[i-1][j-1];
                else dp[i][j] = 1 + Math.min(dp[i-1][j-1], Math.min(dp[i-1][j], dp[i][j-1]))
            }
        }
        return dp[m][n];
    }
}
```

---

15) Find All Anagrams in a String (438)

Sample 1 Input: `s = "cbaebabacd", p = "abc"` Output: `[0,6]`

Sample 2 Input: `s = "abab", p = "ab"` Output: `[0,1,2]`

Code:

```
import java.util.*;
class Solution {
    public List<Integer> findAnagrams(String s, String p) {
        List<Integer> res = new ArrayList<>();
        if (s.length() < p.length()) return res;
        int[] pd = new int[26], sd = new int[26];
        for (int i=0;i<p.length();i++){ pd[p.charAt(i)-'a']++; sd[s.charAt(i)-'a']++; }
        if (Arrays.equals(pd, sd)) res.add(0);
        for (int i=p.length(); i<s.length(); i++){
            sd[s.charAt(i)-'a']++;
            sd[s.charAt(i-p.length())-'a']--;
            if (Arrays.equals(pd, sd)) res.add(i-p.length()+1);
        }
        return res;
    }
}
```

---

16) Palindromic Substrings (647)

Sample 1 Input: `s = "aaa"` Output: `6`

Sample 2 Input: `s = "abc"` Output: `3`

Code:

```
class Solution {
    public int countSubstrings(String s) {
        int n = s.length(), res = 0;
        for (int center=0; center<n; center++){
            res += expand(s, center, center);
            res += expand(s, center, center+1);
        }
        return res;
    }
    private int expand(String s, int l, int r) {
        int cnt = 0;
        while (l>=0 && r<s.length() && s.charAt(l)==s.charAt(r)) { cnt++; l--; r++; }
        return cnt;
    }
}
```

---

17) Minimum Add to Make Parentheses Valid (921)

Sample 1 Input: `s = "())"` Output: `1`

Sample 2 Input: `s = "((("` Output: `3`

Code:

```java
class Solution {
    public int minAddToMakeValid(String s) {
        int balance = 0, res = 0;
        for (char c: s.toCharArray()){
            if (c == '(') balance++;
            else {
                if (balance==0) res++;
                else balance--;
            }
        }
        return res + balance;
    }
}
```

---

18) Integer to Roman (12)

Sample 1 Input: `num = 3` Output: `"III"`

Sample 2 Input: `num = 58` Output: `"LVIII"`

Code:

```java
class Solution {
    public String intToRoman(int num) {
        int[] vals = {1000,900,500,400,100,90,50,40,10,9,5,4,1};
        String[] romans = {"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};
        StringBuilder sb = new StringBuilder();
        for (int i=0;i<vals.length;i++){
            while (num >= vals[i]) { num -= vals[i]; sb.append(romans[i]); }
        }
        return sb.toString();
    }
}
```

---

19) Next Greater Element III (556)

Sample 1 Input: `n = 12` Output: `21`

Sample 2 Input: `n = 21` Output: `-1`

Code:

```
class Solution {
    public int nextGreaterElement(int n) {
        char[] a = String.valueOf(n).toCharArray();
        int i = a.length - 2;
        while (i>=0 && a[i] >= a[i+1]) i--;
        if (i < 0) return -1;
        int j = a.length - 1;
        while (a[j] <= a[i]) j--;
        swap(a, i, j);
        reverse(a, i+1, a.length-1);
        long val = Long.parseLong(new String(a));
        return (val > Integer.MAX_VALUE) ? -1 : (int)val;
    }
    private void swap(char[] a, int i, int j){ char t=a[i]; a[i]=a[j]; a[j]=t; }
    private void reverse(char[] a, int l, int r){ while(l<r) swap(a,l++,r--); }
}
```

---

20) Longest Common Subsequence (1143)

Sample 1 Input: `text1="abcde", text2="ace"` Output: `3`

Sample 2 Input: `text1="abc", text2="abc"` Output: `3`

Code:

```
class Solution {
    public int longestCommonSubsequence(String t1, String t2) {
        int m = t1.length(), n = t2.length();
        int[] dp = new int[n+1];
        for (int i=1;i<=m;i++){
            int prev = 0;
            for (int j=1;j<=n;j++){
                int temp = dp[j];
                if (t1.charAt(i-1) == t2.charAt(j-1)) dp[j] = prev + 1;
                else dp[j] = Math.max(dp[j], dp[j-1]);
                prev = temp;
            }
        }
        return dp[n];
    }
}
```

---

21) Group Shifted Strings (249)

Sample 1 Input: `["abc","bcd","acef","xyz","az","ba","a","z"]` Output: `[["abc","bcd","xyz"],["az","ba"],["acef"],["a","z"]]` (grouping)

Sample 2 Input: `["a"]` Output: `[["a"]]`

Code:

```
import java.util.*;
class Solution {
    public List<List<String>> groupStrings(String[] strings) {
        Map<String, List<String>> map = new HashMap<>();
        for (String s: strings) {
            String key = getKey(s);
            map.computeIfAbsent(key, k->new ArrayList<>()).add(s);
        }
        return new ArrayList<>(map.values());
    }
    private String getKey(String s) {
        if (s.length() == 0) return "";
        StringBuilder sb = new StringBuilder();
        for (int i=1;i<s.length();i++){
            int diff = (s.charAt(i) - s.charAt(i-1) + 26) % 26;
            sb.append(diff).append(',');
        }
        return sb.toString();
    }
}
```

---

22) Minimum Number of Steps to Make Two Strings Anagram (1347)

Sample 1 Input: `s = "bab", t = "aba"` Output: `1`

Sample 2 Input: `s = "leetcode", t = "practice"` Output: `5`

Code:

```
class Solution {
    public int minSteps(String s, String t) {
        int[] cnt = new int[26];
        for (char c: s.toCharArray()) cnt[c-'a']++;
        for (char c: t.toCharArray()) cnt[c-'a']--;
        int res = 0;
        for (int x: cnt) if (x>0) res += x;
        return res;
    }
}
```

---

23) Longest Substring with At Most K Distinct Characters (340)

Sample 1 Input: `s = "eceba", k = 2` Output: `3`

Sample 2 Input: `s = "aa", k = 1` Output: `2`

Code:

```java
class Solution {
    public int lengthOfLongestSubstringKDistinct(String s, int k) {
        if (k == 0) return 0;
        int[] cnt = new int[256];
        int left = 0, distinct = 0, res = 0;
        for (int right=0; right<s.length(); right++){
            if (cnt[s.charAt(right)]++ == 0) distinct++;
            while (distinct > k) {
                if (--cnt[s.charAt(left++)] == 0) distinct--;
            }
            res = Math.max(res, right - left + 1);
        }
        return res;
    }
}
```

---

24) Multiply Strings (43)

Sample 1 Input: `num1 = "2", num2 = "3"` Output: `"6"`

Sample 2 Input: `num1 = "123", num2 = "456"` Output: `"56088"`

Code:

```java
class Solution {
    public String multiply(String num1, String num2) {
        int n1 = num1.length(), n2 = num2.length();
        int[] res = new int[n1 + n2];
        for (int i = n1-1; i>=0; i--){
            for (int j = n2-1; j>=0; j--){
                int mul = (num1.charAt(i)-'0') * (num2.charAt(j)-'0');
                int sum = mul + res[i+j+1];
                res[i+j+1] = sum % 10;
                res[i+j] += sum / 10;
            }
        }
        StringBuilder sb = new StringBuilder();
        for (int num: res) if (!(sb.length()==0 && num==0)) sb.append(num);
```

```
        return sb.length()==0 ? "0" : sb.toString();
    }
}
```

---

25) Shifting Letters (848)

Sample 1 Input: `s = "abc", shifts = [3,5,9]` Output: `"rpl"`

Sample 2 Input: `s = "aaa", shifts = [1,2,3]` Output: `"g"`? *(example style; LeetCode typical output: "g" for combined shifts)*

Code:

```
class Solution {
    public String shiftingLetters(String s, int[] shifts) {
        int n = s.length();
        long sum = 0;
        char[] arr = s.toCharArray();
        for (int i = n-1; i>=0; i--){
            sum = (sum + shifts[i]) % 26;
            arr[i] = (char) ((arr[i]-'a' + sum) % 26 + 'a');
        }
        return new String(arr);
    }
}
```

BFS

---

1) Number of Islands (200)

Sample 1 Input:

```
grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
```

Output:

```
3
```

Sample 2 Input:

```
grid = [
  ["1","1","1"],
  ["0","1","0"],
  ["1","1","1"]
]
```

Output:

```
1
```

Code:

```
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') {
                    count++;
                    dfs(grid, i, j, m, n);
                }
            }
        }
        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);
    }
}
```

---

2) Binary Tree Vertical Order Traversal (314)

Sample 1 Input:

```
root = [3,9,20,null,null,15,7]
```

Output:

```
[[9],[3,15],[20],[7]]
```

(Columns from left to right; nodes in each column top-to-bottom)

Sample 2 Input:

root = [1,2,3,4,5,6,7]

Output:

[[4],[2],[1,5,6],[3],[7]]

Code:

```java
import java.util.*;
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val; this.left = left; this.right = right;
 *     }
 * }
 */
class Solution {
    public List<List<Integer>> verticalOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if (root == null) return res;
        Map<Integer, List<Integer>> map = new HashMap<>();
        Queue<TreeNode> qNode = new LinkedList<>();
        Queue<Integer> qCol = new LinkedList<>();
        qNode.add(root); qCol.add(0);
        int min = 0, max = 0;
        while (!qNode.isEmpty()) {
            TreeNode node = qNode.poll();
            int col = qCol.poll();
            map.computeIfAbsent(col, k->new ArrayList<>()).add(node.val);
            min = Math.min(min, col);
            max = Math.max(max, col);
            if (node.left != null) { qNode.add(node.left); qCol.add(col-1); }
            if (node.right != null) { qNode.add(node.right); qCol.add(col+1); }
        }
        for (int c = min; c <= max; c++) res.add(map.getOrDefault(c, new ArrayList<>()));
        return res;
    }
}
```

---

3) Binary Tree Right Side View (199)

Sample 1 Input:

`root = [1,2,3,null,5,null,4]`

Output:

`[1,3,4]`

Sample 2 Input:

`root = [1,2,3,4]`

Output:

`[1,3,4]`

Code (DFS approach, prefer right-first traversal):

```java
import java.util.*;
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val; this.left = left; this.right = right;
 *     }
 * }
 */
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        dfs(root, 0, res);
        return res;
    }
    private void dfs(TreeNode node, int depth, List<Integer> res) {
        if (node == null) return;
        if (depth == res.size()) res.add(node.val); // first visited at this depth is right
        dfs(node.right, depth+1, res);
        dfs(node.left, depth+1, res);
    }
}
```

---

4) Smallest String With Swaps (1202)

Sample 1 Input:

`s = "dcab", pairs = [[0,3],[1,2]]`

Output:

`"bacd"`

Sample 2 Input:

`s = "dcab", pairs = [[0,3],[1,2],[0,2]]`

Output:

`"abcd"`

Code (Union-Find + sort within components):

```java
import java.util.*;
class Solution {
    public String smallestStringWithSwaps(String s, List<List<Integer>> pairs) {
        int n = s.length();
        DSU dsu = new DSU(n);
        for (List<Integer> p : pairs) dsu.union(p.get(0), p.get(1));
        Map<Integer, List<Integer>> comp = new HashMap<>();
        for (int i = 0; i < n; i++) comp.computeIfAbsent(dsu.find(i), k->new ArrayList<>())
        char[] ans = new char[n];
        for (List<Integer> indices : comp.values()) {
            List<Character> chars = new ArrayList<>();
            for (int idx : indices) chars.add(s.charAt(idx));
            Collections.sort(chars);
            Collections.sort(indices);
            for (int i = 0; i < indices.size(); i++) ans[indices.get(i)] = chars.get(i);
        }
        return new String(ans);
    }
    static class DSU {
        int[] p;
        public DSU(int n) { p = new int[n]; for (int i=0;i<n;i++) p[i]=i; }
        int find(int x){ return p[x]==x?x:(p[x]=find(p[x])); }
        void union(int a,int b){ int ra=find(a), rb=find(b); if (ra!=rb) p[rb]=ra; }
    }
}
```

---

5) Shortest Bridge (934)

Sample 1 Input:

```
grid = [
 [0,1,0],
 [0,0,0],
 [0,0,1]
]
```

Output:

```
2
```

(Flip two 0s to connect islands)

Sample 2 Input:

```
grid = [
 [0,1,0,0,0],
 [0,0,0,0,1],
 [0,0,0,0,0],
 [0,0,1,0,0]
]
```

Output:

```
2
```

Code (DFS to mark first island, BFS from its border to reach second island):

```java
import java.util.*;
class Solution {
    public int shortestBridge(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        boolean found = false;
        Deque<int[]> q = new ArrayDeque<>();
        boolean[][] vis = new boolean[m][n];
        // find first island and mark with DFS, push border cells to queue
        for (int i=0;i<m && !found;i++){
            for (int j=0;j<n && !found;j++){
                if (grid[i][j] == 1) {
                    dfsMark(grid, i, j, vis, q);
                    found = true;
                }
            }
        }
        // BFS from all marked island cells
        int steps = 0;
        int[][] dirs = {{1,0},{-1,0},{0,1},{0,-1}};
        while (!q.isEmpty()) {
            int size = q.size();
            for (int k=0;k<size;k++){
                int[] cur = q.poll();
```

```
                int x = cur[0], y = cur[1];
                for (int[] d: dirs){
                    int nx = x + d[0], ny = y + d[1];
                    if (nx<0 || ny<0 || nx>=m || ny>=n || vis[nx][ny]) continue;
                    if (grid[nx][ny] == 1) return steps;
                    vis[nx][ny] = true;
                    q.add(new int[]{nx, ny});
                }
            }
            steps++;
        }
        return -1;
    }
    private void dfsMark(int[][] g, int i, int j, boolean[][] vis, Deque<int[]> q) {
        int m = g.length, n = g[0].length;
        if (i<0 || j<0 || i>=m || j>=n || vis[i][j] || g[i][j] == 0) return;
        vis[i][j] = true;
        q.add(new int[]{i, j}); // treat all island cells as BFS sources
        dfsMark(g, i+1, j, vis, q);
        dfsMark(g, i-1, j, vis, q);
        dfsMark(g, i, j+1, vis, q);
        dfsMark(g, i, j-1, vis, q);
    }
}
```

---

6) Number of Distinct Islands (694)

Sample 1 Input:

```
grid = [
 [1,1,0,0],
 [1,0,0,1],
 [0,0,0,1]
]
```

Output:

```
2
```

Sample 2 Input:

```
grid = [
 [1,1,1],
 [0,1,0],
 [1,1,1]
]
```

Output:

2

Code:

```
import java.util.*;
class Solution {
    public int numDistinctIslands(int[][] grid) {
        Set<String> shapes = new HashSet<>();
        int m = grid.length, n = grid[0].length;
        boolean[][] vis = new boolean[m][n];
        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (grid[i][j]==1 && !vis[i][j]) {
                    List<String> shape = new ArrayList<>();
                    dfs(grid, i, j, vis, i, j, shape);
                    shapes.add(String.join(",", shape));
                }
            }
        }
        return shapes.size();
    }
    private void dfs(int[][] g, int i, int j, boolean[][] vis, int bi, int bj, List<String>
        int m=g.length, n=g[0].length;
        if (i<0||j<0||i>=m||j>=n||g[i][j]==0||vis[i][j]) return;
        vis[i][j] = true;
        shape.add((i-bi)+":"+(j-bj));
        dfs(g, i+1, j, vis, bi, bj, shape);
        dfs(g, i-1, j, vis, bi, bj, shape);
        dfs(g, i, j+1, vis, bi, bj, shape);
        dfs(g, i, j-1, vis, bi, bj, shape);
    }
}
```

---

7) Water and Jug Problem (365)

Sample 1 Input:

`x = 3, y = 5, target = 4`

Output:

`true`

Sample 2 Input:

```
x = 2, y = 6, target = 5
```

Output:

```
false
```

Code (mathematical GCD + BFS optional):

```java
class Solution {
    public boolean canMeasureWater(int x, int y, int target) {
        if (target > x + y) return false;
        if (target == 0) return true;
        return target % gcd(x, y) == 0;
    }
    private int gcd(int a, int b){
        return b==0 ? a : gcd(b, a%b);
    }
}
```

---

8) Max Area of Island (695)

Sample 1 Input:

```
grid = [
 [0,0,1,0,0],
 [0,1,1,1,0],
 [0,0,1,0,0],
 [1,1,0,0,0]
]
```

Output:

```
5
```

Sample 2 Input:

```
grid = [
 [1,1],
 [1,1]
]
```

Output:

```
4
```

Code:

```java
class Solution {
    public int maxAreaOfIsland(int[][] grid) {
        int m = grid.length, n = grid[0].length, max = 0;
```

```
        boolean[][] vis = new boolean[m][n];
        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (grid[i][j]==1 && !vis[i][j]) {
                    max = Math.max(max, dfs(grid, i, j, vis));
                }
            }
        }
        return max;
    }
    private int dfs(int[][] g, int i, int j, boolean[][] vis){
        int m=g.length, n=g[0].length;
        if (i<0||j<0||i>=m||j>=n||g[i][j]==0||vis[i][j]) return 0;
        vis[i][j] = true;
        return 1 + dfs(g,i+1,j,vis) + dfs(g,i-1,j,vis)
                 + dfs(g,i,j+1,vis) + dfs(g,i,j-1,vis);
    }
}
```

---

9) Minimum Time to Collect All Apples in a Tree (1443)

Sample 1 Input:

```
n = 7
edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]]
hasApple = [false,false,true,false,true,true,false]
```

Output:

```
8
```

Sample 2 Input:

```
n = 4
edges = [[0,1],[1,2],[0,3]]
hasApple = [true,true,false,true]
```

Output:

```
4
```

Code:

```
import java.util.*;
class Solution {
    public int minTime(int n, int[][] edges, List<Boolean> hasApple) {
```

```
        List<List<Integer>> g = new ArrayList<>();
        for (int i=0;i<n;i++) g.add(new ArrayList<>());
        for (int[] e: edges){
            g.get(e[0]).add(e[1]);
            g.get(e[1]).add(e[0]);
        }
        return Math.max(0, dfs(0, -1, g, hasApple));
    }
    private int dfs(int node, int parent, List<List<Integer>> g, List<Boolean> has){
        int total = 0;
        for (int nei: g.get(node)){
            if (nei == parent) continue;
            int cost = dfs(nei, node, g, has);
            if (cost > 0 || has.get(nei)) total += cost + 2;
        }
        return total;
    }
}
```

---

10) Cheapest Flights Within K Stops (787)

Sample 1 Input:

```
n = 3
flights = [[0,1,100],[1,2,100],[0,2,500]]
src = 0, dst = 2, k = 1
```

Output:

```
200
```

Sample 2 Input:

```
n = 4
flights = [[0,1,1],[1,2,1],[2,3,1],[0,3,50]]
src = 0, dst = 3, k = 1
```

Output:

```
50
```

Code (BFS / Bellman-Ford style):

```
import java.util.*;
class Solution {
    public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k) {
        int[] cost = new int[n];
        Arrays.fill(cost, Integer.MAX_VALUE);
```

```
        cost[src] = 0;

        for (int steps=0; steps<=k; steps++){
            int[] temp = cost.clone();
            for (int[] f: flights){
                int u=f[0], v=f[1], w=f[2];
                if (cost[u] != Integer.MAX_VALUE && cost[u] + w < temp[v]) {
                    temp[v] = cost[u] + w;
                }
            }
            cost = temp;
        }
        return cost[dst] == Integer.MAX_VALUE ? -1 : cost[dst];
    }
}
```

---

11) Surrounded Regions (130)

Sample 1 Input:

```
board = [
 ["X","X","X","X"],
 ["X","O","O","X"],
 ["X","X","O","X"],
 ["X","O","X","X"]
]
```

Output:

```
[
 ["X","X","X","X"],
 ["X","X","X","X"],
 ["X","X","X","X"],
 ["X","O","X","X"]
]
```

Sample 2 Input:

```
board = [
 ["O","O"],
 ["O","O"]
]
```

Output:

```
[
 ["O","O"],
```

```
 ["O","O"]
]
```

Code:

```java
import java.util.*;
class Solution {
    public void solve(char[][] board) {
        int m = board.length, n = board[0].length;
        boolean[][] safe = new boolean[m][n];

        for (int i = 0; i < m; i++) {
            dfs(board, safe, i, 0);
            dfs(board, safe, i, n-1);
        }
        for (int j = 0; j < n; j++) {
            dfs(board, safe, 0, j);
            dfs(board, safe, m-1, j);
        }

        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (board[i][j]=='O' && !safe[i][j]) board[i][j]='X';
            }
        }
    }

    private void dfs(char[][] board, boolean[][] safe, int i, int j) {
        int m = board.length, n = board[0].length;
        if (i<0||j<0||i>=m||j>=n||board[i][j]!='O'||safe[i][j]) return;
        safe[i][j] = true;
        dfs(board, safe, i+1, j);
        dfs(board, safe, i-1, j);
        dfs(board, safe, i, j+1);
        dfs(board, safe, i, j-1);
    }
}
```

---

12) Number of Provinces (547)

Sample 1 Input:

```
isConnected = [
 [1,1,0],
 [1,1,0],
```

```
 [0,0,1]
]
```

Output:

```
2
```

Sample 2 Input:

```
isConnected = [
 [1,0,0],
 [0,1,0],
 [0,0,1]
]
```

Output:

```
3
```

Code:

```java
class Solution {
    public int findCircleNum(int[][] isConnected) {
        int n = isConnected.length, count=0;
        boolean[] vis = new boolean[n];
        for (int i=0;i<n;i++){
            if (!vis[i]) {
                dfs(isConnected, vis, i);
                count++;
            }
        }
        return count;
    }
    private void dfs(int[][] g, boolean[] vis, int i){
        vis[i] = true;
        for (int j=0;j<g.length;j++){
            if (g[i][j] == 1 && !vis[j]) dfs(g, vis, j);
        }
    }
}
```

---

13) Find Distance in a Binary Tree (1740)

Sample 1 Input:

```
root = [1,2,3]
p = 2, q = 3
```

Output:

2

Sample 2 Input:

```
root = [5,3,6,2,4,null,7]
p = 2, q = 7
```

Output:

4

Code:

```
import java.util.*;
class Solution {
    public int findDistance(TreeNode root, int p, int q) {
        TreeNode lca = lca(root, p, q);
        return dist(lca, p, 0) + dist(lca, q, 0);
    }
    private TreeNode lca(TreeNode node, int p, int q){
        if (node == null) return null;
        if (node.val == p || node.val == q) return node;
        TreeNode left = lca(node.left, p, q);
        TreeNode right = lca(node.right, p, q);
        if (left != null && right != null) return node;
        return left != null ? left : right;
    }
    private int dist(TreeNode node, int target, int d){
        if (node == null) return -1;
        if (node.val == target) return d;
        int left = dist(node.left, target, d+1);
        if (left != -1) return left;
        return dist(node.right, target, d+1);
    }
}
```

---

14) Number of Connected Components in an Undirected Graph (323)

Sample 1 Input:

```
n = 5
edges = [[0,1],[1,2],[3,4]]
```

Output:

2

Sample 2 Input:

```
n = 4
edges = [[0,1],[2,3],[1,2]]
```

Output:

1

Code:

```java
import java.util.*;
class Solution {
    public int countComponents(int n, int[][] edges) {
        List<List<Integer>> g = new ArrayList<>();
        for (int i=0;i<n;i++) g.add(new ArrayList<>());
        for (int[] e: edges){
            g.get(e[0]).add(e[1]);
            g.get(e[1]).add(e[0]);
        }
        boolean[] vis = new boolean[n];
        int count=0;
        for (int i=0;i<n;i++){
            if (!vis[i]) {
                dfs(g, vis, i);
                count++;
            }
        }
        return count;
    }
    private void dfs(List<List<Integer>> g, boolean[] vis, int u){
        vis[u] = true;
        for (int v: g.get(u)){
            if (!vis[v]) dfs(g, vis, v);
        }
    }
}
```

---

15) Possible Bipartition (886)

Sample 1 Input:

```
n = 4
dislikes = [[1,2],[1,3],[2,4]]
```

Output:

```
true
```

Sample 2 Input:

```
n = 3
dislikes = [[1,2],[1,3],[2,3]]
```

Output:

```
false
```

Code (DFS two-color graph):

```java
import java.util.*;
class Solution {
    public boolean possibleBipartition(int n, int[][] dislikes) {
        List<List<Integer>> g = new ArrayList<>();
        for (int i=0;i<=n;i++) g.add(new ArrayList<>());
        for (int[] d: dislikes){
            g.get(d[0]).add(d[1]);
            g.get(d[1]).add(d[0]);
        }
        int[] color = new int[n+1]; // 0=uncolored, 1, -1
        for (int i=1;i<=n;i++){
            if (color[i]==0 && !dfs(g, color, i, 1)) return false;
        }
        return true;
    }

    private boolean dfs(List<List<Integer>> g, int[] color, int node, int c){
        color[node] = c;
        for (int nei: g.get(node)) {
            if (color[nei] == c) return false;
            if (color[nei] == 0 && !dfs(g, color, nei, -c)) return false;
        }
        return true;
    }
}
```

---

16) Number of Operations to Make Network Connected (1319)

Sample 1 Input:

```
n = 4
```

```
connections = [[0,1],[0,2],[1,2]]
```

Output:

```
1
```

Sample 2 Input:

```
n = 6
connections = [[0,1],[0,2],[0,3],[1,2]]
```

Output:

```
2
```

Code (Union-Find):

```
class Solution {
    public int makeConnected(int n, int[][] connections) {
        if (connections.length < n-1) return -1;

        DSU dsu = new DSU(n);
        for (int[] c: connections) dsu.union(c[0], c[1]);

        int comp = 0;
        for (int i=0;i<n;i++) {
            if (dsu.find(i) == i) comp++;
        }
        return comp - 1;
    }

    static class DSU {
        int[] parent;
        DSU(int n){
            parent = new int[n];
            for (int i=0;i<n;i++) parent[i] = i;
        }
        int find(int x){
            return parent[x] == x ? x : (parent[x] = find(parent[x]));
        }
        void union(int a, int b){
            int ra = find(a), rb = find(b);
            if (ra != rb) parent[rb] = ra;
        }
    }
}
```

---

17) All Paths From Source to Target (797)

Sample 1 Input:

```
graph = [[1,2],[3],[3],[]]
```

Output:

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

Sample 2 Input:

```
graph = [[4,3,1],[3,2,4],[3],[4],[]]
```

Output:

```
[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]
```

Code:

```java
import java.util.*;
class Solution {
    List<List<Integer>> res;
    public List<List<Integer>> allPathsSourceTarget(int[][] graph) {
        res = new ArrayList<>();
        dfs(graph, 0, new ArrayList<>());
        return res;
    }

    private void dfs(int[][] g, int node, List<Integer> path){
        path.add(node);
        if (node == g.length-1) {
            res.add(new ArrayList<>(path));
            path.remove(path.size()-1);
            return;
        }
        for (int nei: g[node]){
            dfs(g, nei, path);
        }
        path.remove(path.size()-1);
    }
}
```

---

18) Minimum Height Trees (310)

Sample 1 Input:

```
n = 4
edges = [[1,0],[1,2],[1,3]]
```

Output:

```
[1]
```

Sample 2 Input:

```
n = 6
edges = [[0,3],[1,3],[2,3],[4,3],[5,4]]
```

Output:

```
[3,4]
```

Code (trim leaves BFS):

```java
import java.util.*;
class Solution {
    public List<Integer> findMinHeightTrees(int n, int[][] edges) {
        if (n == 1) return Arrays.asList(0);

        List<Set<Integer>> g = new ArrayList<>();
        for (int i=0;i<n;i++) g.add(new HashSet<>());
        for (int[] e: edges){
            g.get(e[0]).add(e[1]);
            g.get(e[1]).add(e[0]);
        }

        List<Integer> leaves = new ArrayList<>();
        for (int i=0;i<n;i++){
            if (g.get(i).size() == 1) leaves.add(i);
        }

        int remaining = n;
        while (remaining > 2){
            remaining -= leaves.size();
            List<Integer> newLeaves = new ArrayList<>();

            for (int leaf: leaves){
                int nei = g.get(leaf).iterator().next();
                g.get(nei).remove(leaf);

                if (g.get(nei).size() == 1) newLeaves.add(nei);
            }
            leaves = newLeaves;
        }

        return leaves;
    }
}
```

19) Number of Closed Islands (1254)

Sample 1 Input:

```
grid = [
 [1,1,1,1,1],
 [1,0,0,0,1],
 [1,0,1,0,1],
 [1,0,0,0,1],
 [1,1,1,1,1]
]
```

Output:

```
1
```

Sample 2 Input:

```
grid = [
 [0,0,1,0,0],
 [0,1,0,1,0],
 [0,1,1,1,0]
]
```

Output:

```
2
```

Code:

```java
class Solution {
    public int closedIsland(int[][] grid) {
        int m = grid.length, n = grid[0].length;

        // eliminate border-connected land
        for (int i=0;i<m;i++){
            dfs(grid, i, 0);
            dfs(grid, i, n-1);
        }
        for (int j=0;j<n;j++){
            dfs(grid, 0, j);
            dfs(grid, m-1, j);
        }

        int count = 0;
        for (int i=1;i<m-1;i++){
            for (int j=1;j<n-1;j++){
```

```
                if (grid[i][j] == 0) {
                    count++;
                    dfs(grid, i, j);
                }
            }
        }
        return count;
    }

    private void dfs(int[][] g, int i, int j){
        int m=g.length, n=g[0].length;
        if (i<0||j<0||i>=m||j>=n||g[i][j] == 1) return;
        g[i][j] = 1;
        dfs(g, i+1, j);
        dfs(g, i-1, j);
        dfs(g, i, j+1);
        dfs(g, i, j-1);
    }
}
```

---

20) Lexicographical Numbers (386)

Sample 1 Input:

```
n = 13
```

Output:

```
[1,10,11,12,13,2,3,4,5,6,7,8,9]
```

Sample 2 Input:

```
n = 2
```

Output:

```
[1,2]
```

Code:

```
import java.util.*;
class Solution {
    public List<Integer> lexicalOrder(int n) {
        List<Integer> res = new ArrayList<>();
        for (int i=1;i<=9;i++){
            dfs(i, n, res);
        }
        return res;
```

```
    }

    private void dfs(int cur, int n, List<Integer> res){
        if (cur > n) return;
        res.add(cur);
        for (int i=0;i<=9;i++){
            int next = cur*10 + i;
            if (next > n) break;
            dfs(next, n, res);
        }
    }
}
```

---

21) Smallest Common Region (1257)

Sample 1 Input:

```
regions = [
 ["Earth","North America","South America"],
 ["North America","USA","Canada"],
 ["USA","New York","Boston"],
 ["South America","Brazil"]
]
region1 = "Boston"
region2 = "Canada"
```

Output:

```
"North America"
```

Sample 2 Input:

```
regions = [
 ["Animal","Mammal","Reptile"],
 ["Mammal","Dog","Cat"],
 ["Dog","Bulldog","Beagle"]
]
region1 = "Beagle"
region2 = "Cat"
```

Output:

```
"Mammal"
```

Code:

```
import java.util.*;
class Solution {
```

```
    public String findSmallestRegion(List<List<String>> regions, String r1, String r2) {
        Map<String,String> parent = new HashMap<>();
        for (List<String> r : regions){
            String root = r.get(0);
            for (int i=1;i<r.size();i++) {
                parent.put(r.get(i), root);
            }
        }
        Set<String> ancestors = new HashSet<>();
        while (r1 != null) {
            ancestors.add(r1);
            r1 = parent.get(r1);
        }
        while (!ancestors.contains(r2)) {
            r2 = parent.get(r2);
        }
        return r2;
    }
}
```

---

22) Redundant Connection (684)

Sample 1 Input:

```
edges = [[1,2],[1,3],[2,3]]
```

Output:

```
[2,3]
```

Sample 2 Input:

```
edges = [[1,4],[3,4],[1,3],[1,2],[4,5]]
```

Output:

```
[1,3]
```

Code:

```
class Solution {
    public int[] findRedundantConnection(int[][] edges) {
        DSU dsu = new DSU(edges.length + 1);
        for (int[] e: edges){
            if (!dsu.union(e[0], e[1])) return e;
        }
        return new int[0];
    }
```

```
    static class DSU {
        int[] p;
        DSU(int n){
            p = new int[n];
            for (int i=0;i<n;i++) p[i]=i;
        }
        int find(int x){
            return p[x]==x ? x : (p[x]=find(p[x]));
        }
        boolean union(int a, int b){
            int ra=find(a), rb=find(b);
            if (ra==rb) return false;
            p[rb]=ra;
            return true;
        }
    }
}
```

---

23) Path With Maximum Minimum Value (1102)

Sample 1 Input:

```
grid = [
 [5,4,5],
 [1,2,6],
 [7,4,6]
]
```

Output:

```
4
```

Sample 2 Input:

```
grid = [
 [2,2,1,2,2,2],
 [1,2,2,2,1,2]
]
```

Output:

```
2
```

Code (max-heap Dijkstra-style):

```
import java.util.*;
class Solution {
```

```
    public int maximumMinimumPath(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        PriorityQueue<int[]> pq = new PriorityQueue<>((a,b)->b[2]-a[2]);
        boolean[][] vis = new boolean[m][n];
        pq.add(new int[]{0,0,grid[0][0]});

        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};

        while (!pq.isEmpty()) {
            int[] cur = pq.poll();
            int r=cur[0], c=cur[1], score=cur[2];

            if (r==m-1 && c==n-1) return score;

            if (vis[r][c]) continue;
            vis[r][c] = true;

            for (int[] d: dirs){
                int nr=r+d[0], nc=c+d[1];
                if (nr<0||nc<0||nr>=m||nc>=n||vis[nr][nc]) continue;
                pq.add(new int[]{nr, nc, Math.min(score, grid[nr][nc])});
            }
        }
        return -1;
    }
}
```

---

24) Lowest Common Ancestor of a BST (235)

Sample 1 Input:

```
root = [6,2,8,0,4,7,9,null,null,3,5]
p = 2
q = 8
```

Output:

```
6
```

Sample 2 Input:

```
root = [6,2,8,0,4,7,9,null,null,3,5]
p = 2
q = 4
```

Output:

2

Code:

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        int pv = p.val, qv = q.val;
        while (root != null) {
            if (pv < root.val && qv < root.val) root = root.left;
            else if (pv > root.val && qv > root.val) root = root.right;
            else return root;
        }
        return null;
    }
}
```

---

25) Find Leaves of Binary Tree (366)

Sample 1 Input:

```
root = [1,2,3,4,5]
```

Output:

```
[[4,5,3],[2],[1]]
```

Sample 2 Input:

```
root = [1]
```

Output:

```
[[1]]
```

Code:

```
import java.util.*;
class Solution {
    public List<List<Integer>> findLeaves(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        dfs(root, res);
        return res;
    }

    private int dfs(TreeNode node, List<List<Integer>> res){
        if (node == null) return -1;
        int left = dfs(node.left, res);
        int right = dfs(node.right, res);
        int height = Math.max(left, right) + 1;
```

```
        if (res.size() == height) res.add(new ArrayList<>());
        res.get(height).add(node.val);
        return height;
    }
}
```

DFS

---

1) Rotting Oranges (994)

Sample 1 Input:

```
grid = [
 [2,1,1],
 [1,1,0],
 [0,1,1]
]
```

Output:

```
4
```

Sample 2 Input:

```
grid = [
 [0,2]
]
```

Output:

```
0
```

Code:

```
import java.util.*;
class Solution {
    public int orangesRotting(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        Queue<int[]> q = new LinkedList<>();
        int fresh = 0;

        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (grid[i][j] == 2) q.add(new int[]{i,j});
                if (grid[i][j] == 1) fresh++;
```

```
            }
        }

        if (fresh == 0) return 0;

        int minutes = -1;
        int[][] dirs = {{1,0},{-1,0},{0,1},{0,-1}};

        while (!q.isEmpty()) {
            int size = q.size();
            minutes++;

            for (int k=0;k<size;k++){
                int[] cur = q.poll();
                for (int[] d: dirs){
                    int x = cur[0] + d[0], y = cur[1] + d[1];
                    if (x<0||y<0||x>=m||y>=n||grid[x][y] != 1) continue;
                    grid[x][y] = 2;
                    fresh--;
                    q.add(new int[]{x,y});
                }
            }
        }

        return fresh == 0 ? minutes : -1;
    }
}
```

---

2) Shortest Path in Binary Matrix (1091)

Sample 1 Input:

```
grid = [
 [0,1],
 [1,0]
]
```

Output:

```
2
```

Sample 2 Input:

```
grid = [
 [0,0,0],
 [1,1,0],
```

```
 [1,1,0]
]
```

Output:

4

Code:

```java
import java.util.*;
class Solution {
    public int shortestPathBinaryMatrix(int[][] grid) {
        int n = grid.length;
        if (grid[0][0] == 1 || grid[n-1][n-1] == 1) return -1;

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{0,0});
        grid[0][0] = 1;

        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1},{1,1},{1,-1},{-1,1},{-1,-1}};
        while (!q.isEmpty()) {
            int[] cur = q.poll();
            int r = cur[0], c = cur[1];
            int dist = grid[r][c];

            if (r == n-1 && c == n-1) return dist;

            for (int[] d: dirs){
                int nr=r+d[0], nc=c+d[1];
                if (nr>=0 && nc>=0 && nr<n && nc<n && grid[nr][nc] == 0){
                    grid[nr][nc] = dist + 1;
                    q.add(new int[]{nr,nc});
                }
            }
        }
        return -1;
    }
}
```

---

3) Shortest Path to Get Food (1730)

Sample 1 Input:

```
grid = [
 ["X","X","X","X","X"],
 ["X","*","O","O","X"],
```

```
 ["X","O","O","#","X"],
 ["X","X","X","X","X"]
]
```

Output:

```
3
```

Sample 2 Input:

```
grid = [
 ["X","X","X"],
 ["X","*","X"],
 ["X","X","X"]
]
```

Output:

```
-1
```

Code:

```
import java.util.*;
class Solution {
    public int getFood(char[][] grid) {
        int m = grid.length, n = grid[0].length;
        Queue<int[]> q = new LinkedList<>();
        boolean[][] vis = new boolean[m][n];

        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (grid[i][j] == '*') {
                    q.add(new int[]{i,j,0});
                    vis[i][j] = true;
                }
            }
        }

        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
        while (!q.isEmpty()){
            int[] cur = q.poll();
            int r=cur[0], c=cur[1], d=cur[2];

            if (grid[r][c] == '#') return d;

            for (int[] dir: dirs){
                int nr=r+dir[0], nc=c+dir[1];
                if (nr>=0&&nc>=0&&nr<m&&nc<n&&!vis[nr][nc] && grid[nr][nc] != 'X'){
                    vis[nr][nc] = true;
                    q.add(new int[]{nr,nc,d+1});
```

```
                }
            }
        }
        return -1;
    }
}
```

---

4) Smallest String With Swaps (1202)

(BFS/Union-Find version — alternative to DFS version)

Sample 1 Input:

```
s = "dcab"
pairs = [[0,3],[1,2]]
```

Output:

```
"bacd"
```

Sample 2 Input:

```
s = "dcab"
pairs = [[0,3],[1,2],[0,2]]
```

Output:

```
"abcd"
```

Code:

```
import java.util.*;
class Solution {
    public String smallestStringWithSwaps(String s, List<List<Integer>> pairs) {
        int n = s.length();
        List<List<Integer>> g = new ArrayList<>();
        for (int i=0;i<n;i++) g.add(new ArrayList<>());

        for (List<Integer> p: pairs){
            int a=p.get(0), b=p.get(1);
            g.get(a).add(b);
            g.get(b).add(a);
        }

        boolean[] vis = new boolean[n];
        char[] res = new char[n];

        for (int i=0;i<n;i++){
```

```
            if (!vis[i]) {
                List<Integer> comp = new ArrayList<>();
                Queue<Integer> q = new LinkedList<>();
                q.add(i); vis[i] = true;

                // BFS collecting connected component
                while (!q.isEmpty()){
                    int node = q.poll();
                    comp.add(node);
                    for (int nei: g.get(node)){
                        if (!vis[nei]){
                            vis[nei] = true;
                            q.add(nei);
                        }
                    }
                }

                // sort indices and chars
                List<Character> chars = new ArrayList<>();
                for (int idx: comp) chars.add(s.charAt(idx));

                Collections.sort(comp);
                Collections.sort(chars);

                for (int k=0;k<comp.size();k++){
                    res[comp.get(k)] = chars.get(k);
                }
            }
        }

        return new String(res);
    }
}
```

---

5) Shortest Bridge (934)

(Using BFS from first island after DFS marking)

Sample 1 Input:

```
grid = [
 [0,1],
 [1,0]
]
```

Output:

1

Sample 2 Input:

```
grid = [
 [0,1,0],
 [0,0,0],
 [0,0,1]
]
```

Output:

2

Code:

```
import java.util.*;
class Solution {
    public int shortestBridge(int[][] grid) {
        int m=grid.length, n=grid[0].length;
        boolean found=false;
        Queue<int[]> q = new LinkedList<>();
        boolean[][] vis = new boolean[m][n];

        // Step 1: Find and mark first island using DFS
        for (int i=0;i<m && !found;i++){
            for (int j=0;j<n && !found;j++){
                if (grid[i][j] == 1) {
                    markIsland(grid, vis, q, i, j);
                    found = true;
                }
            }
        }

        // Step 2: BFS from island border to reach second island
        int steps = 0;
        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
        while (!q.isEmpty()){
            int size = q.size();
            for (int k=0;k<size;k++){
                int[] cur = q.poll();
                for (int[] d: dirs){
                    int x = cur[0]+d[0], y = cur[1]+d[1];
                    if (x<0||y<0||x>=m||y>=n||vis[x][y]) continue;

                    if (grid[x][y] == 1) return steps; // reached second island
```

```
                    vis[x][y] = true;
                    q.add(new int[]{x,y});
                }
            }
            steps++;
        }
        return -1;
    }

    private void markIsland(int[][] g, boolean[][] vis, Queue<int[]> q, int i, int j){
        int m=g.length, n=g[0].length;
        if (i<0||j<0||i>=m||j>=n||g[i][j] == 0 || vis[i][j]) return;
        vis[i][j] = true;
        q.add(new int[]{i,j});
        markIsland(g, vis, q, i+1, j);
        markIsland(g, vis, q, i-1, j);
        markIsland(g, vis, q, i, j+1);
        markIsland(g, vis, q, i, j-1);
    }
}
```

---

6) Number of Distinct Islands (694)

(Usually solved by DFS; BFS version for shape collection is also possible.)

Sample 1 Input:

```
grid = [
 [1,1,0,0],
 [1,0,0,1],
 [0,0,0,1]
]
```

Output:

```
2
```

Sample 2 Input:

```
grid = [
 [1,1,1],
 [0,1,0],
 [1,1,1]
]
```

Output:

2

Code (BFS shape collection):

```
import java.util.*;
class Solution {
    public int numDistinctIslands(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        boolean[][] vis = new boolean[m][n];
        Set<String> shapes = new HashSet<>();

        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (grid[i][j] == 1 && !vis[i][j]) {
                    List<int[]> shape = new ArrayList<>();
                    bfs(grid, vis, i, j, shape);
                    shapes.add(encode(shape));
                }
            }
        }
        return shapes.size();
    }

    private void bfs(int[][] g, boolean[][] vis, int r0, int c0, List<int[]> shape){
        int m=g.length, n=g[0].length;
        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{r0,c0});
        vis[r0][c0] = true;

        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};

        while (!q.isEmpty()){
            int[] cur = q.poll();
            int r=cur[0], c=cur[1];
            shape.add(new int[]{r-r0, c-c0}); // relative position

            for (int[] d: dirs){
                int nr=r+d[0], nc=c+d[1];
                if (nr>=0 && nc>=0 && nr<m && nc<n && g[nr][nc]==1 && !vis[nr][nc]) {
                    vis[nr][nc] = true;
                    q.add(new int[]{nr,nc});
                }
            }
        }
    }

    private String encode(List<int[]> shape){
```

```
        shape.sort((a,b)-> a[0]==b[0] ? a[1]-b[1] : a[0]-b[0]);
        StringBuilder sb = new StringBuilder();
        for (int[] p: shape) sb.append(p[0]).append(":").append(p[1]).append(",");
        return sb.toString();
    }
}
```

---

7) Water and Jug Problem (365)

(BFS state search approach — alternative to GCD method.)

Sample 1 Input:

```
x = 3, y = 5, target = 4
```

Output:

```
true
```

Sample 2 Input:

```
x = 2, y = 6, target = 5
```

Output:

```
false
```

Code (BFS simulation of states):

```
import java.util.*;
class Solution {
    public boolean canMeasureWater(int x, int y, int target) {
        if (target > x + y) return false;
        if (target == 0) return true;

        Queue<int[]> q = new LinkedList<>();
        Set<String> seen = new HashSet<>();

        q.add(new int[]{0,0});
        seen.add("0,0");

        while (!q.isEmpty()){
            int[] cur = q.poll();
            int a = cur[0], b = cur[1];

            if (a == target || b == target || a + b == target) return true;

            List<int[]> next = new ArrayList<>();
```

```
                // fill jug a
                next.add(new int[]{x, b});
                // fill jug b
                next.add(new int[]{a, y});
                // empty jug a
                next.add(new int[]{0, b});
                // empty jug b
                next.add(new int[]{a, 0});
                // pour a -> b
                int pour = Math.min(a, y - b);
                next.add(new int[]{a - pour, b + pour});
                // pour b -> a
                pour = Math.min(b, x - a);
                next.add(new int[]{a + pour, b - pour});

                for (int[] n: next){
                    String key = n[0] + "," + n[1];
                    if (!seen.contains(key)){
                        seen.add(key);
                        q.add(n);
                    }
                }
            }
            return false;
        }
}
```

---

8) Shortest Path with Alternating Colors (1129)

Sample 1 Input:

```
n = 3
redEdges = [[0,1],[1,2]]
blueEdges = []
```

Output:

```
[0,1,2]
```

Sample 2 Input:

```
n = 3
redEdges = [[0,1]]
blueEdges = [[1,2]]
```

Output:

[0,1,2]

Code:

```java
import java.util.*;
class Solution {
    public int[] shortestAlternatingPaths(int n, int[][] redEdges, int[][] blueEdges) {
        List<Integer>[] red = new ArrayList[n];
        List<Integer>[] blue = new ArrayList[n];
        for (int i=0;i<n;i++){
            red[i] = new ArrayList<>();
            blue[i] = new ArrayList<>();
        }
        for (int[] e: redEdges) red[e[0]].add(e[1]);
        for (int[] e: blueEdges) blue[e[0]].add(e[1]);

        int[][] dist = new int[2][n];
        Arrays.fill(dist[0], Integer.MAX_VALUE);
        Arrays.fill(dist[1], Integer.MAX_VALUE);
        dist[0][0] = dist[1][0] = 0;

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{0,0}); // node 0 with red previous
        q.add(new int[]{0,1}); // node 0 with blue previous

        while (!q.isEmpty()){
            int[] cur = q.poll();
            int node = cur[0], color = cur[1]; // 0=redPrev, 1=bluePrev

            List<Integer>[] nextEdges = (color == 0) ? blue : red;
            int nextColor = 1 - color;

            for (int nei : nextEdges[node]){
                if (dist[nextColor][nei] == Integer.MAX_VALUE){
                    dist[nextColor][nei] = dist[color][node] + 1;
                    q.add(new int[]{nei, nextColor});
                }
            }
        }

        int[] ans = new int[n];
        for (int i=0;i<n;i++){
            int d = Math.min(dist[0][i], dist[1][i]);
            ans[i] = (d == Integer.MAX_VALUE) ? -1 : d;
        }
        return ans;
    }
```

```
}
```

---

9) Perfect Squares (279)

Sample 1 Input:

`n = 12`

Output:

`3`

$(12 = 4 + 4 + 4)$

Sample 2 Input:

`n = 13`

Output:

`2`

$(13 = 4 + 9)$

Code (BFS over residual values):

```
import java.util.*;
class Solution {
    public int numSquares(int n) {
        Queue<Integer> q = new LinkedList<>();
        boolean[] vis = new boolean[n+1];
        q.add(n);
        vis[n] = true;

        int level = 0;
        while (!q.isEmpty()){
            int size = q.size();
            level++;

            for (int i=0;i<size;i++){
                int cur = q.poll();

                for (int x=1; x*x <= cur; x++){
                    int next = cur - x*x;
                    if (next == 0) return level;
                    if (!vis[next]){
                        vis[next] = true;
                        q.add(next);
                    }
```

```
                }
            }
        }
        return -1;
    }
}
```

---

10) Open the Lock (752)

Sample 1 Input:

```
deadends = ["0201","0101","0102","1212","2002"]
target = "0202"
```

Output:

```
6
```

Sample 2 Input:

```
deadends = ["8888"]
target = "0009"
```

Output:

```
1
```

Code:

```
import java.util.*;
class Solution {
    public int openLock(String[] deadends, String target) {
        Set<String> dead = new HashSet<>(Arrays.asList(deadends));
        if (dead.contains("0000")) return -1;

        Queue<String> q = new LinkedList<>();
        q.add("0000");

        Set<String> vis = new HashSet<>();
        vis.add("0000");

        int steps = 0;

        while (!q.isEmpty()){
            int size = q.size();
            for (int i=0;i<size;i++){
                String cur = q.poll();
                if (cur.equals(target)) return steps;
```

```
                if (dead.contains(cur)) continue;

                for (String nei: neighbors(cur)){
                    if (!vis.contains(nei)){
                        vis.add(nei);
                        q.add(nei);
                    }
                }
            }
            steps++;
        }
        return -1;
    }

    private List<String> neighbors(String s){
        List<String> res = new ArrayList<>();
        char[] arr = s.toCharArray();
        for (int i=0;i<4;i++){
            char c = arr[i];
            char up = c == '9' ? '0' : (char)(c+1);
            char down = c == '0' ? '9' : (char)(c-1);

            arr[i] = up; res.add(new String(arr));
            arr[i] = down; res.add(new String(arr));
            arr[i] = c;
        }
        return res;
    }
}
```

---

11) Minimum Time to Collect All Apples in a Tree (1443)

(Tree BFS/DFS possible — BFS solution shown)

Sample 1 Input:

```
n = 7
edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]]
hasApple = [false,false,true,false,true,true,false]
```

Output:

```
8
```

Sample 2 Input:

```
n = 4
edges = [[0,1],[1,2],[0,3]]
hasApple = [true,true,false,true]
```

Output:

```
4
```

Code (tree build + BFS to compute required edges):

```java
import java.util.*;
class Solution {
    public int minTime(int n, int[][] edges, List<Boolean> hasApple) {
        List<List<Integer>> g = new ArrayList<>();
        for (int i=0;i<n;i++) g.add(new ArrayList<>());
        for (int[] e: edges){
            g.get(e[0]).add(e[1]);
            g.get(e[1]).add(e[0]);
        }

        // BFS parent discovery
        int[] parent = new int[n];
        Arrays.fill(parent, -1);

        Queue<Integer> q = new LinkedList<>();
        q.add(0);

        while (!q.isEmpty()){
            int node = q.poll();
            for (int nei: g.get(node)){
                if (nei == parent[node]) continue;
                parent[nei] = node;
                q.add(nei);
            }
        }

        // Track which nodes must be visited
        boolean[] needed = new boolean[n];
        for (int i=0;i<n;i++){
            if (hasApple.get(i)){
                int cur = i;
                needed[cur] = true;
                while (cur != 0 && !needed[parent[cur]]){
                    needed[parent[cur]] = true;
                    cur = parent[cur];
                }
```

```
            }
        }

        int cost = 0;
        for (int i=1;i<n;i++){
            if (needed[i]) cost += 2; // go down and return
        }
        return cost;
    }
}
```

---

12) Cheapest Flights Within K Stops (787)

(BFS-like Bellman Ford)

Sample 1 Input:

```
n = 3
flights = [[0,1,100],[1,2,100],[0,2,500]]
src = 0, dst = 2, k = 1
```

Output:

```
200
```

Sample 2 Input:

```
n = 4
flights = [[0,1,1],[1,2,1],[2,3,1],[0,3,50]]
src = 0, dst = 3, k = 1
```

Output:

```
50
```

Code:

```java
import java.util.*;
class Solution {
    public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k) {
        int[] dist = new int[n];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;

        for (int i = 0; i <= k; i++) {
            int[] temp = dist.clone();
            for (int[] f : flights) {
                int u = f[0], v = f[1], w = f[2];
```

```
                if (dist[u] != Integer.MAX_VALUE && dist[u] + w < temp[v]) {
                    temp[v] = dist[u] + w;
                }
            }
            dist = temp;
        }
        return dist[dst] == Integer.MAX_VALUE ? -1 : dist[dst];
    }
}
```

---

13) Surrounded Regions (130)

(BFS flood-fill from borders)

Sample 1 Input:

```
board = [
 ["X","X","X","X"],
 ["X","O","O","X"],
 ["X","X","O","X"],
 ["X","O","X","X"]
]
```

Output:

```
[
 ["X","X","X","X"],
 ["X","X","X","X"],
 ["X","X","X","X"],
 ["X","O","X","X"]
]
```

Sample 2 Input:

```
board = [
 ["O","O"],
 ["O","O"]
]
```

Output:

```
[
 ["O","O"],
 ["O","O"]
]
```

Code:

```java
import java.util.*;
class Solution {
    public void solve(char[][] board) {
        int m = board.length, n = board[0].length;
        Queue<int[]> q = new LinkedList<>();
        boolean[][] safe = new boolean[m][n];

        // add all border O's to queue
        for (int i=0;i<m;i++){
            if (board[i][0] == 'O') q.add(new int[]{i,0});
            if (board[i][n-1] == 'O') q.add(new int[]{i,n-1});
        }
        for (int j=0;j<n;j++){
            if (board[0][j] == 'O') q.add(new int[]{0,j});
            if (board[m-1][j] == 'O') q.add(new int[]{m-1,j});
        }

        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
        while (!q.isEmpty()){
            int[] cur = q.poll();
            int r=cur[0], c=cur[1];
            if (safe[r][c]) continue;
            safe[r][c] = true;

            for (int[] d: dirs){
                int nr=r+d[0], nc=c+d[1];
                if (nr>=0&&nc>=0&&nr<m&&nc<n && board[nr][nc]=='O' && !safe[nr][nc]) {
                    q.add(new int[]{nr,nc});
                }
            }
        }

        // flip all non-safe 'O' to 'X'
        for (int i=0;i<m;i++){
            for (int j=0;j<n;j++){
                if (board[i][j]=='O' && !safe[i][j]) board[i][j]='X';
            }
        }
    }
}
```

---

14) Binary Tree Level Order Traversal (102)

Sample 1 Input:

`root = [3,9,20,null,null,15,7]`

Output:

`[[9],[3,15],[20],[7]]`

(Correct BFS output: [[3],[9,20],[15,7]])

Sample 2 Input:

`root = [1]`

Output:

`[[1]]`

Code:

```java
import java.util.*;
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if (root == null) return res;

        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);

        while (!q.isEmpty()){
            int size = q.size();
            List<Integer> level = new ArrayList<>();

            for (int i=0;i<size;i++){
                TreeNode node = q.poll();
                level.add(node.val);
                if (node.left != null) q.add(node.left);
                if (node.right != null) q.add(node.right);
            }

            res.add(level);
        }
        return res;
    }
}
```

---

15) Number of Provinces (547)

(BFS graph traversal)

Sample 1 Input:

```
isConnected = [
 [1,1,0],
 [1,1,0],
 [0,0,1]
]
```

Output:

```
2
```

Sample 2 Input:

```
isConnected = [
 [1,0,0],
 [0,1,0],
 [0,0,1]
]
```

Output:

```
3
```

Code:

```java
import java.util.*;
class Solution {
    public int findCircleNum(int[][] isConnected) {
        int n = isConnected.length;
        boolean[] vis = new boolean[n];
        int count = 0;

        for (int i=0;i<n;i++){
            if (!vis[i]){
                count++;
                bfs(isConnected, vis, i);
            }
        }
        return count;
    }

    private void bfs(int[][] g, boolean[] vis, int start){
        Queue<Integer> q = new LinkedList<>();
        q.add(start);
        vis[start] = true;
```

```
        while (!q.isEmpty()){
            int node = q.poll();
            for (int j=0;j<g.length;j++){
                if (g[node][j] == 1 && !vis[j]){
                    vis[j] = true;
                    q.add(j);
                }
            }
        }
    }
}
```

---

16) Find Distance in a Binary Tree (1740)

(Use BFS from one node after finding LCA)

Sample 1 Input:

```
root = [1,2,3]
p = 2
q = 3
```

Output:

```
2
```

Sample 2 Input:

```
root = [5,3,6,2,4,null,7]
p = 2
q = 7
```

Output:

```
4
```

Code:

```
class Solution {
    public int findDistance(TreeNode root, int p, int q) {
        TreeNode lca = lca(root, p, q);
        return dist(lca, p) + dist(lca, q);
    }

    private TreeNode lca(TreeNode root, int p, int q) {
        if (root == null) return null;
        if (root.val == p || root.val == q) return root;
```

```
        TreeNode left = lca(root.left, p, q);
        TreeNode right = lca(root.right, p, q);

        if (left != null && right != null) return root;
        return (left != null) ? left : right;
    }

    private int dist(TreeNode root, int target) {
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);

        int steps = 0;
        while (!q.isEmpty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = q.poll();
                if (node.val == target) return steps;

                if (node.left != null) q.add(node.left);
                if (node.right != null) q.add(node.right);
            }
            steps++;
        }
        return -1;
    }
}
```

---

17) Number of Connected Components in an Undirected Graph (323)

(BFS version)

Sample 1 Input:

```
n = 5
edges = [[0,1],[1,2],[3,4]]
```

Output:

```
2
```

Sample 2 Input:

```
n = 4
edges = [[0,1],[2,3],[1,2]]
```

Output:

1

Code:

```
import java.util.*;
class Solution {
    public int countComponents(int n, int[][] edges) {
        List<List<Integer>> g = new ArrayList<>();
        for (int i = 0; i < n; i++) g.add(new ArrayList<>());
        for (int[] e : edges){
            g.get(e[0]).add(e[1]);
            g.get(e[1]).add(e[0]);
        }

        boolean[] vis = new boolean[n];
        int count = 0;

        for (int i=0;i<n;i++){
            if (!vis[i]) {
                count++;
                bfs(g, vis, i);
            }
        }
        return count;
    }

    private void bfs(List<List<Integer>> g, boolean[] vis, int start){
        Queue<Integer> q = new LinkedList<>();
        q.add(start);
        vis[start] = true;

        while (!q.isEmpty()){
            int u = q.poll();
            for (int v : g.get(u)){
                if (!vis[v]){
                    vis[v] = true;
                    q.add(v);
                }
            }
        }
    }
}
```

---

18) Possible Bipartition (886)

(BFS graph coloring solution)

Sample 1 Input:

```
n = 4
dislikes = [[1,2],[1,3],[2,4]]
```

Output:

```
true
```

Sample 2 Input:

```
n = 3
dislikes = [[1,2],[1,3],[2,3]]
```

Output:

```
false
```

Code:

```
import java.util.*;
class Solution {
    public boolean possibleBipartition(int n, int[][] dislikes) {
        List<List<Integer>> g = new ArrayList<>();
        for (int i=0;i<=n;i++) g.add(new ArrayList<>());

        for (int[] d : dislikes){
            g.get(d[0]).add(d[1]);
            g.get(d[1]).add(d[0]);
        }

        int[] color = new int[n+1];

        for (int i=1;i<=n;i++){
            if (color[i] == 0){
                if (!bfs(g, color, i)) return false;
            }
        }
        return true;
    }

    private boolean bfs(List<List<Integer>> g, int[] color, int start){
        Queue<Integer> q = new LinkedList<>();
        q.add(start);
        color[start] = 1;

        while (!q.isEmpty()){
```

```
            int node = q.poll();
            for (int nei : g.get(node)){
                if (color[nei] == color[node]) return false;
                if (color[nei] == 0){
                    color[nei] = -color[node];
                    q.add(nei);
                }
            }
        }
        return true;
    }
}
```

---

19) Minimum Knight Moves (1197)

Sample 1 Input:

```
x = 2, y = 1
```

Output:

```
1
```

Sample 2 Input:

```
x = 5, y = 5
```

Output:

```
4
```

Code:

```
import java.util.*;
class Solution {
    public int minKnightMoves(int x, int y) {
        x = Math.abs(x);
        y = Math.abs(y);  // symmetry

        int[][] dirs = {
            {1,2},{2,1},{-1,2},{-2,1},
            {1,-2},{2,-1},{-1,-2},{-2,-1}
        };

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{0,0});
        Set<String> vis = new HashSet<>();
        vis.add("0,0");
```

```
        int steps = 0;

        while (!q.isEmpty()){
            int size = q.size();
            for (int i=0;i<size;i++){
                int[] cur = q.poll();
                int cx = cur[0], cy = cur[1];

                if (cx == x && cy == y) return steps;

                for (int[] d: dirs){
                    int nx = cx + d[0], ny = cy + d[1];
                    if (nx >= -2 && ny >= -2 && nx <= x+2 && ny <= y+2){
                        String key = nx + "," + ny;
                        if (!vis.contains(key)){
                            vis.add(key);
                            q.add(new int[]{nx,ny});
                        }
                    }
                }
            }
            steps++;
        }
        return -1;
    }
}
```

---

20) Number of Operations to Make Network Connected (1319)

(BFS alternative approach)

Sample 1 Input:

```
n = 4
connections = [[0,1],[0,2],[1,2]]
```

Output:

```
1
```

Sample 2 Input:

```
n = 6
connections = [[0,1],[0,2],[0,3],[1,2]]
```

Output:

2

Code (BFS count connected components):

```
import java.util.*;
class Solution {
    public int makeConnected(int n, int[][] connections) {
        if (connections.length < n - 1) return -1;

        List<List<Integer>> g = new ArrayList<>();
        for (int i=0;i<n;i++) g.add(new ArrayList<>());
        for (int[] c : connections){
            g.get(c[0]).add(c[1]);
            g.get(c[1]).add(c[0]);
        }

        boolean[] vis = new boolean[n];
        int comp = 0;

        for (int i=0;i<n;i++){
            if (!vis[i]){
                comp++;
                bfs(g, vis, i);
            }
        }

        return comp - 1;
    }

    private void bfs(List<List<Integer>> g, boolean[] vis, int start){
        Queue<Integer> q = new LinkedList<>();
        q.add(start);
        vis[start] = true;

        while (!q.isEmpty()){
            int node = q.poll();
            for (int nei : g.get(node)){
                if (!vis[nei]){
                    vis[nei] = true;
                    q.add(nei);
                }
            }
        }
    }
}
```

21) All Paths From Source to Target (797)

(Traditionally DFS — but BFS can also generate all paths.)

Sample 1 Input:

`graph = [[1,2],[3],[3],[]]`

Output:

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

Sample 2 Input:

`graph = [[4,3,1],[3,2,4],[3],[4],[]]`

Output:

`[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]`

Code (BFS storing entire paths):

```java
import java.util.*;
class Solution {
    public List<List<Integer>> allPathsSourceTarget(int[][] graph) {
        List<List<Integer>> res = new ArrayList<>();
        int n = graph.length;

        Queue<List<Integer>> q = new LinkedList<>();
        q.add(Arrays.asList(0));

        while (!q.isEmpty()) {
            List<Integer> path = q.poll();
            int last = path.get(path.size() - 1);

            if (last == n - 1) {
                res.add(new ArrayList<>(path));
                continue;
            }

            for (int nei : graph[last]) {
                List<Integer> newPath = new ArrayList<>(path);
                newPath.add(nei);
                q.add(newPath);
            }
        }
        return res;
```

```
    }
}
```

---

22) Brace Expansion (1087)

(BFS used to build all combinations)

Sample 1 Input:

`s = "{a,b}c{d,e}f"`

Output:

`["acdf","acef","bcdf","bcef"]`

Sample 2 Input:

`s = "abcd"`

Output:

`["abcd"]`

Code:

```java
import java.util.*;
class Solution {
    public String[] expand(String s) {
        Queue<String> q = new LinkedList<>();
        q.add("");

        int i = 0;
        while (i < s.length()) {
            Queue<String> next = new LinkedList<>();

            if (s.charAt(i) == '{') {
                int end = i;
                while (s.charAt(end) != '}') end++;

                String[] options = s.substring(i+1, end).split(",");
                while (!q.isEmpty()) {
                    String base = q.poll();
                    for (String opt : options) next.add(base + opt);
                }
                q = next;
                i = end + 1;

            } else {
```

```
                while (!q.isEmpty()) {
                    next.add(q.poll() + s.charAt(i));
                }
                q = next;
                i++;
            }
        }

        List<String> result = new ArrayList<>(q);
        Collections.sort(result);
        return result.toArray(new String[0]);
    }
}
```

---

23) Minimum Height Trees (310)

(Center-finding using BFS leaf trimming)

Sample 1 Input:

```
n = 4
edges = [[1,0],[1,2],[1,3]]
```

Output:

```
[1]
```

Sample 2 Input:

```
n = 6
edges = [[0,3],[1,3],[2,3],[4,3],[5,4]]
```

Output:

```
[3,4]
```

Code (BFS leaf trimming):

```
import java.util.*;
class Solution {
    public List<Integer> findMinHeightTrees(int n, int[][] edges) {
        if (n == 1) return Arrays.asList(0);

        List<Set<Integer>> g = new ArrayList<>();
        for (int i = 0; i < n; i++) g.add(new HashSet<>());
        for (int[] e : edges) {
            g.get(e[0]).add(e[1]);
            g.get(e[1]).add(e[0]);
```

```
        }

        List<Integer> leaves = new ArrayList<>();
        for (int i = 0; i < n; i++)
            if (g.get(i).size() == 1) leaves.add(i);

        int remaining = n;
        while (remaining > 2) {
            remaining -= leaves.size();
            List<Integer> newLeaves = new ArrayList<>();

            for (int leaf : leaves) {
                int nei = g.get(leaf).iterator().next();
                g.get(nei).remove(leaf);
                if (g.get(nei).size() == 1) newLeaves.add(nei);
            }
            leaves = newLeaves;
        }
        return leaves;
    }
}
```

---

24) As Far from Land as Possible (1162)

(Multi-source BFS from all land cells)

Sample 1 Input:

```
grid = [
 [1,0,1],
 [0,0,0],
 [1,0,1]
]
```

Output:

```
2
```

Sample 2 Input:

```
grid = [
 [1,1],
 [1,1]
]
```

Output:

`-1`

Code:

```
import java.util.*;
class Solution {
    public int maxDistance(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        Queue<int[]> q = new LinkedList<>();

        for (int i=0;i<m;i++)
            for (int j=0;j<n;j++)
                if (grid[i][j] == 1)
                    q.add(new int[]{i,j});

        if (q.isEmpty() || q.size() == m*n) return -1;

        int dist = -1;
        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};

        while (!q.isEmpty()) {
            int size = q.size();
            dist++;

            for (int k = 0; k < size; k++) {
                int[] cur = q.poll();
                int r = cur[0], c = cur[1];

                for (int[] d: dirs) {
                    int nr=r+d[0], nc=c+d[1];
                    if (nr>=0&&nc>=0&&nr<m&&nc<n && grid[nr][nc] == 0) {
                        grid[nr][nc] = 1;
                        q.add(new int[]{nr,nc});
                    }
                }
            }
        }
        return dist;
    }
}
```

---

25) Number of Closed Islands (1254)

(BFS version — flood-fill water boundary first)

Sample 1 Input:

```
grid = [
 [1,1,1,1,1],
 [1,0,0,0,1],
 [1,0,1,0,1],
 [1,0,0,0,1],
 [1,1,1,1,1]
]
```

Output:

```
1
```

Sample 2 Input:

```
grid = [
 [0,0,1,0,0],
 [0,1,0,1,0],
 [0,1,1,1,0]
]
```

Output:

```
2
```

Code:

```java
import java.util.*;
class Solution {
    public int closedIsland(int[][] grid) {
        int m = grid.length, n = grid[0].length;

        // Remove open islands (touch boundary)
        for (int i=0;i<m;i++){
            bfs(grid, i, 0);
            bfs(grid, i, n-1);
        }
        for (int j=0;j<n;j++){
            bfs(grid, 0, j);
            bfs(grid, m-1, j);
        }

        int count = 0;
        for (int i=1;i<m-1;i++){
            for (int j=1;j<n-1;j++){
                if (grid[i][j] == 0) {
                    count++;
                    bfs(grid, i, j);
                }
```

```
            }
        }
        return count;
    }

    private void bfs(int[][] g, int r, int c){
        int m=g.length, n=g[0].length;
        if (g[r][c] == 1) return;

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{r,c});
        g[r][c] = 1;

        int[][] dirs={{1,0},{-1,0},{0,1},{0,-1}};
        while (!q.isEmpty()){
            int[] cur = q.poll();
            for (int[] d: dirs){
                int nr=cur[0]+d[0], nc=cur[1]+d[1];
                if (nr>=0&&nc>=0&&nr<m&&nc<n && g[nr][nc] == 0){
                    g[nr][nc] = 1;
                    q.add(new int[]{nr,nc});
                }
            }
        }
    }
}
```