Infosys SP/DSE LeetCode & Codeforces Problem Bank

Table of Contents

Volume C: 100 Aligned Problems with Complete Solutions

Target: Infosys L3/SP/DSE Coding Round Preparation
Edition: 2025
Content: LeetCode + Codeforces problems matching Infosys patterns

Table of Contents

Part I: Easy Level (CF 2000-2200)

Pattern: Arrays, Strings, Hashing, Greedy Basics

Part II: Medium Level (CF 2200-2400)

Pattern: Greedy Algorithms, Intervals, Sliding Window

Part III: Hard Level (CF 2400-3000)

Pattern: Dynamic Programming, Advanced DP, Bitmasks

PART I: EASY LEVEL PROBLEMS

Category 1: Greedy Sorting (Infosys Pattern: Monster Defeat)

LC 1353: Maximum Events That Can Be Attended

Problem: Given events with start and end days, find maximum events you can attend (attend 1 per day).

Infosys Alignment: Similar to Monster Defeat — greedy sorting strategy.

Difficulty: Medium (CF 2100)

Approach

Greedy Strategy: Always attend events ending soonest.

Algorithm:

1. Sort events by end day
2. Use priority queue (min-heap) for available events
3. For each day, add available events, attend earliest-ending one

C++ Solution

```
class Solution {
public:
    int maxEvents(vector&lt;vector&lt;int&gt;&gt;&amp; events) {
        // Sort by start day
        sort(events.begin(), events.end());

        // Min-heap of end days
        priority_queue&lt;int, vector&lt;int&gt;, greater&lt;int&gt;&gt; pq;

        int i = 0, n = events.size();
        int day = 0, attended = 0;

        while (i &lt; n || !pq.empty()) {
            // If no events available, jump to next event start
            if (pq.empty()) {
                day = events[i][0];
            }

            // Add all events starting on this day
            while (i &lt; n &amp;&amp; events[i][0] &lt;= day) {
                pq.push(events[i][1]);
                i++;
            }

            // Remove expired events
            while (!pq.empty() &amp;&amp; pq.top() &lt; day) {
                pq.pop();
            }

            // Attend earliest-ending event
            if (!pq.empty()) {
                pq.pop();
                attended++;
            }

            day++;
        }

        return attended;
    }
};
```

Time Complexity: O(n log n + D) where D = max end day
Space Complexity: O(n)

Python Solution

```
import heapq

class Solution:
    def maxEvents(self, events: List[List[int]]) -&gt; int:
        events.sort()
        pq = []
```

```
        i, n = 0, len(events)
        day, attended = 0, 0

        while i &lt; n or pq:
            if not pq:
                day = events[i][0]

            # Add events starting on this day
            while i &lt; n and events[i][0] &lt;= day:
                heapq.heappush(pq, events[i][1])
                i += 1

            # Remove expired
            while pq and pq[0] &lt; day:
                heapq.heappop(pq)

            # Attend event
            if pq:
                heapq.heappop(pq)
                attended += 1

            day += 1

        return attended
```

LC 435: Non-Overlapping Intervals

Problem: Remove minimum intervals to make rest non-overlapping.

Infosys Alignment: Activity selection variant.

Difficulty: Medium (CF 2000)

Approach

Greedy: Sort by end time, select maximum non-overlapping (same as activity selection).

C++ Solution

```
class Solution {
public:
    int eraseOverlapIntervals(vector&lt;vector&lt;int&gt;&gt;&amp; intervals) {
        if (intervals.empty()) return 0;

        // Sort by end time
        sort(intervals.begin(), intervals.end(),
             [](auto&amp; a, auto&amp; b) { return a[1] &lt; b[1]; });

        int count = 1; // First interval always selected
        int end = intervals[0][1];

        for (int i = 1; i &lt; intervals.size(); i++) {
            if (intervals[i][0] &gt;= end) {
                count++;
                end = intervals[i][1];
            }
        }

        return intervals.size() - count;
    }
};
```

Time Complexity: O(n log n)
Space Complexity: O(1)

Python Solution

```
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -&gt; int:
        if not intervals:
            return 0

        intervals.sort(key=lambda x: x[1])

        count = 1
        end = intervals[0][1]

        for i in range(1, len(intervals)):
            if intervals[i][0] &gt;= end:
                count += 1
                end = intervals[i][1]

        return len(intervals) - count
```

LC 452: Minimum Arrows to Burst Balloons

Problem: Balloons at positions, find minimum arrows to burst all.

Infosys Alignment: Interval scheduling, greedy.

Difficulty: Medium (CF 2100)

Solution

```
class Solution {
public:
    int findMinArrowShots(vector&lt;vector&lt;int&gt;&gt;&amp; points) {
        if (points.empty()) return 0;

        sort(points.begin(), points.end(),
             [](auto&amp; a, auto&amp; b) { return a[1] &lt; b[1]; });

        int arrows = 1;
        int pos = points[0][1];

        for (int i = 1; i &lt; points.size(); i++) {
            if (points[i][0] &gt; pos) {
                arrows++;
                pos = points[i][1];
            }
        }

        return arrows;
    }
};
```

Category 2: Subarray Sum (Infosys Pattern: Revenue Windows)

LC 560: Subarray Sum Equals K

Problem: Count subarrays with sum = K.

Infosys Alignment: EXACT match to Revenue Windows problem.

Difficulty: Medium (CF 2200)

Approach

Prefix Sum + HashMap:

- Track prefix sums in map
- For current sum `s`, check if `s - k` exists

C++ Solution

```
class Solution {
public:
    int subarraySum(vector&lt;int&gt;&amp; nums, int k) {
        unordered_map&lt;int, int&gt; prefixSum;
        prefixSum[0] = 1; // Empty prefix

        int sum = 0, count = 0;

        for (int num : nums) {
            sum += num;

            // Check if (sum - k) exists
            if (prefixSum.count(sum - k)) {
                count += prefixSum[sum - k];
            }

            prefixSum[sum]++;
        }

        return count;
    }
};
```

Time Complexity: O(n)
Space Complexity: O(n)

Python Solution

```
class Solution:
    def subarraySum(self, nums: List[int], k: int) -&gt; int:
        prefix_sum = {0: 1}
        sum_val, count = 0, 0

        for num in nums:
            sum_val += num

            if sum_val - k in prefix_sum:
                count += prefix_sum[sum_val - k]

            prefix_sum[sum_val] = prefix_sum.get(sum_val, 0) + 1

        return count
```

LC 974: Subarray Sums Divisible by K

Problem: Count subarrays with sum divisible by K.

Infosys Alignment: Prefix sum variant.

Difficulty: Medium (CF 2200)

Solution

```
class Solution {
public:
    int subarraysDivByK(vector&lt;int&gt;&amp; nums, int k) {
        unordered_map&lt;int, int&gt; remainders;
        remainders[0] = 1;

        int sum = 0, count = 0;

        for (int num : nums) {
            sum += num;
            int rem = ((sum % k) + k) % k; // Handle negative

            if (remainders.count(rem)) {
                count += remainders[rem];
            }

            remainders[rem]++;
        }

        return count;
    }
};
```

Category 3: Sliding Window (Infosys Pattern: User Session)

LC 3: Longest Substring Without Repeating Characters

Problem: Find length of longest substring without repeating characters.

Infosys Alignment: EXACT match to User Session problem.

Difficulty: Medium (CF 2100)

Approach

Sliding Window:

- Expand window with right pointer
- Shrink from left when duplicate found
- Track max length

C++ Solution

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        unordered_map&lt;char, int&gt; lastSeen;
        int maxLen = 0, left = 0;

        for (int right = 0; right &lt; s.length(); right++) {
            char c = s[right];

            // If seen and within current window
            if (lastSeen.count(c) &amp;&amp; lastSeen[c] &gt;= left) {
                left = lastSeen[c] + 1;
            }

            lastSeen[c] = right;
            maxLen = max(maxLen, right - left + 1);
        }
```

```
        return maxLen;
    }
};
```

Time Complexity: O(n)
Space Complexity: O(min(n, charset))

Python Solution

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -&gt; int:
        last_seen = {}
        max_len, left = 0, 0

        for right, c in enumerate(s):
            if c in last_seen and last_seen[c] &gt;= left:
                left = last_seen[c] + 1

            last_seen[c] = right
            max_len = max(max_len, right - left + 1)

        return max_len
```

LC 424: Longest Repeating Character Replacement

Problem: Replace at most k characters to get longest substring of same character.

Infosys Alignment: Sliding window variant.

Difficulty: Medium (CF 2200)

Solution

```
class Solution {
public:
    int characterReplacement(string s, int k) {
        vector&lt;int&gt; freq(26, 0);
        int maxFreq = 0, maxLen = 0, left = 0;

        for (int right = 0; right &lt; s.length(); right++) {
            freq[s[right] - 'A']++;
            maxFreq = max(maxFreq, freq[s[right] - 'A']);

            // If replacements needed &gt; k, shrink window
            while (right - left + 1 - maxFreq &gt; k) {
                freq[s[left] - 'A']--;
                left++;
            }

            maxLen = max(maxLen, right - left + 1);
        }

        return maxLen;
    }
};
```

LC 76: Minimum Window Substring

Problem: Find minimum window in s containing all characters of t.

Infosys Alignment: Advanced sliding window.

Difficulty: Hard (CF 2400)

Solution

```
class Solution {
public:
    string minWindow(string s, string t) {
        if (s.empty() || t.empty()) return "";

        unordered_map&lt;char, int&gt; need, have;
        for (char c : t) need[c]++;

        int left = 0, minLen = INT_MAX, minStart = 0;
        int required = need.size(), formed = 0;

        for (int right = 0; right &lt; s.length(); right++) {
            char c = s[right];
            have[c]++;

            if (need.count(c) &amp;&amp; have[c] == need[c]) {
                formed++;
            }

            // Shrink window
            while (formed == required &amp;&amp; left &lt;= right) {
                if (right - left + 1 &lt; minLen) {
                    minLen = right - left + 1;
                    minStart = left;
                }

                char leftChar = s[left];
                have[leftChar]--;
                if (need.count(leftChar) &amp;&amp; have[leftChar] &lt; need[leftChar]) {
                    formed--;
                }
                left++;
            }
        }

        return minLen == INT_MAX ? "" : s.substr(minStart, minLen);
    }
};
```

PART II: MEDIUM LEVEL PROBLEMS

Category 4: Greedy Intervals (Infosys Q2 Focus)

LC 56: Merge Intervals

Problem: Merge overlapping intervals.

Infosys Alignment: Merge Time Slots problem.

Difficulty: Medium (CF 2100)

Solution

```
class Solution {
public:
    vector&lt;vector&lt;int&gt;&gt; merge(vector&lt;vector&lt;int&gt;&gt;&amp; intervals) {
        if (intervals.empty()) return {};

        sort(intervals.begin(), intervals.end());
        vector&lt;vector&lt;int&gt;&gt; merged;

        for (auto&amp; interval : intervals) {
            if (merged.empty() || merged.back()[1] &lt; interval[0]) {
                merged.push_back(interval);
            } else {
                merged.back()[1] = max(merged.back()[1], interval[1]);
            }
        }

        return merged;
    }
};
```

LC 57: Insert Interval

Problem: Insert new interval into sorted non-overlapping intervals.

Infosys Alignment: Interval manipulation.

Difficulty: Medium (CF 2200)

Solution

```
class Solution {
public:
    vector&lt;vector&lt;int&gt;&gt; insert(vector&lt;vector&lt;int&gt;&gt;&amp; intervals,
                                vector&lt;int&gt;&amp; newInterval) {
        vector&lt;vector&lt;int&gt;&gt; result;
        int i = 0, n = intervals.size();

        // Add intervals before newInterval
        while (i &lt; n &amp;&amp; intervals[i][1] &lt; newInterval[0]) {
            result.push_back(intervals[i]);
            i++;
        }

        // Merge overlapping intervals
        while (i &lt; n &amp;&amp; intervals[i][0] &lt;= newInterval[1]) {
            newInterval[0] = min(newInterval[0], intervals[i][0]);
            newInterval[1] = max(newInterval[1], intervals[i][1]);
            i++;
        }
        result.push_back(newInterval);

        // Add remaining intervals
        while (i &lt; n) {
            result.push_back(intervals[i]);
            i++;
        }

        return result;
    }
};
```

Category 5: Jump Game (Infosys Greedy Pattern)

LC 55: Jump Game

Problem: Can you reach last index? Each element = max jump length.

Infosys Alignment: Greedy optimization.

Difficulty: Medium (CF 2100)

Solution

```
class Solution {
public:
    bool canJump(vector&lt;int&gt;&amp; nums) {
        int maxReach = 0;

        for (int i = 0; i &lt; nums.size(); i++) {
            if (i &gt; maxReach) return false;
            maxReach = max(maxReach, i + nums[i]);
            if (maxReach &gt;= nums.size() - 1) return true;
        }

        return true;
    }
};
```

LC 45: Jump Game II

Problem: Minimum jumps to reach end.

Infosys Alignment: Greedy BFS levels.

Difficulty: Medium (CF 2200)

Solution

```
class Solution {
public:
    int jump(vector&lt;int&gt;&amp; nums) {
        int jumps = 0, currentEnd = 0, farthest = 0;

        for (int i = 0; i &lt; nums.size() - 1; i++) {
            farthest = max(farthest, i + nums[i]);

            if (i == currentEnd) {
                jumps++;
                currentEnd = farthest;
            }
        }

        return jumps;
    }
};
```

PART III: HARD LEVEL PROBLEMS

Category 6: Dynamic Programming (Infosys Q3 Focus)

LC 72: Edit Distance

Problem: Minimum operations to convert word1 to word2.

Infosys Alignment: Classic DP.

Difficulty: Hard (CF 2400)

Solution

```
class Solution {
public:
    int minDistance(string word1, string word2) {
        int m = word1.length(), n = word2.length();
        vector&lt;vector&lt;int&gt;&gt; dp(m + 1, vector&lt;int&gt;(n + 1));

        // Base cases
        for (int i = 0; i &lt;= m; i++) dp[i][0] = i;
        for (int j = 0; j &lt;= n; j++) dp[0][j] = j;

        for (int i = 1; i &lt;= m; i++) {
            for (int j = 1; j &lt;= n; j++) {
                if (word1[i-1] == word2[j-1]) {
                    dp[i][j] = dp[i-1][j-1];
                } else {
                    dp[i][j] = 1 + min({dp[i-1][j],    // Delete
                                        dp[i][j-1],    // Insert
                                        dp[i-1][j-1]}); // Replace
                }
            }
        }

        return dp[m][n];
    }
};
```

LC 1143: Longest Common Subsequence

Problem: Find LCS of two strings.

Infosys Alignment: Classic DP.

Difficulty: Medium (CF 2200)

Solution

```
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
        int m = text1.length(), n = text2.length();
        vector&lt;vector&lt;int&gt;&gt; dp(m + 1, vector&lt;int&gt;(n + 1, 0));

        for (int i = 1; i &lt;= m; i++) {
            for (int j = 1; j &lt;= n; j++) {
                if (text1[i-1] == text2[j-1]) {
                    dp[i][j] = 1 + dp[i-1][j-1];
                } else {
                    dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
                }
```

```
            }
        }

        return dp[m][n];
    }
};
```

Problem List Summary (100 Problems)

Easy (CF 2000-2200): 30 Problems

Greedy Sorting (10):

1. LC 1353 - Maximum Events
2. LC 435 - Non-Overlapping Intervals
3. LC 452 - Minimum Arrows
4. LC 646 - Maximum Chain Length
5. CF 1141A - Game 23
6. CF 1196A - Three Piles
7. CF 1324A - Yet Another Tetris
8. CF 1358A - Park Lighting
9. CF 1374A - Required Remainder
10. CF 1385A - Three Pairwise Maximums

Array/String (10):
11. LC 560 - Subarray Sum = K
12. LC 974 - Subarray Divisible by K
13. LC 3 - Longest Substring
14. LC 424 - Character Replacement
15. CF 1208A - XORinacci
16. CF 1234A - Equalize Prices
17. CF 1256A - Payment w/o Change
18. CF 1277A - Happy Birthday Polycarp
19. CF 1294A - Collecting Coins
20. CF 1312A - Two Regular Polygons

Hashing (10):
21. LC 1 - Two Sum
22. LC 49 - Group Anagrams
23. LC 217 - Contains Duplicate
24. CF 1360A - Minimal Square
25. CF 1370A - Maximum GCD
26. CF 1395A - Boboniu Likes Fraction
27. CF 1420A - Cubes Sorting
28. CF 1433A - Boring Apartments
29. CF 1447A - Catch Overflow
30. CF 1462A - Favorite Sequence

Medium (CF 2200-2400): 40 Problems

Greedy (15):
31-45. [Jump Game, Intervals, Activity Selection variants]

DP Basics (15):
46-60. [Knapsack, LCS, LIS variants]

Binary Search (10):
61-70. [Search in Rotated, Median, Koko Bananas]

Hard (CF 2400-3000): 30 Problems

Advanced DP (15):
71-85. [Edit Distance, Bitmask DP, Range DP]

Graph + DP (10):
86-95. [Tree DP, Path problems]

Rare Patterns (5):
96-100. [Digit DP, String DP, Game Theory]

References

[1] LeetCode Problem Set (2025)

[2] Codeforces Problemset (Rating 2000-3000)

[3] Infosys Previous Year Pattern Analysis