LeetCode Complete Study Guide with Static Mermaid Flows

2026-02-22

LeetCode Complete Study Guide with Static Mermaid Flows

This PDF contains a concise problem statement, core idea, variable focus, and Mermaid flow for each problem.

1) Two Sum

Problem statement: Given nums and target, return indices i and j where nums[i] + nums[j] == target, with i != j.

Core idea: Hash map complement lookup.

Key variables: seen map value->index.

Answer target: return [seen[target-x], i].

```
flowchart TD
    A[Input parsed for problem 1] --> B[Initialize seen map value->index]
    B --> C[Apply Hash map complement lookup]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return [seen[target-x], i]]
```

49) Group Anagrams

Problem statement: Group strings that are anagrams and return grouped lists.

Core idea: Hash by character signature.

Key variables: groups map signature->list.

Answer target: return all group lists.

```
flowchart TD
    A[Input parsed for problem 49] --> B[Initialize groups map signature->list]
    B --> C[Apply Hash by character signature]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return all group lists]
```

238) Product of Array Except Self

Problem statement: Return output where output[i] is product of all nums except nums[i], without division.

Core idea: Prefix and suffix products.

Key variables: out prefix, suffix accumulator.

Answer target: return out.

```
flowchart TD
    A[Input parsed for problem 238] --> B[Initialize out prefix, suffix accumulator]
    B --> C[Apply Prefix and suffix products]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return out]
```

560) Subarray Sum Equals K

Problem statement: Count contiguous subarrays with sum exactly k.

Core idea: Prefix sum count map.

Key variables: cur sum, cnt map, ans.

Answer target: return ans.

```
flowchart TD
    A[Input parsed for problem 560] --> B[Initialize cur sum, cnt map, ans]
    B --> C[Apply Prefix sum count map]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return ans]
```

347) Top K Frequent Elements

Problem statement: Return k most frequent elements.

Core idea: Frequency + buckets.

Key variables: freq map, buckets, out.

Answer target: return first k.

```
flowchart TD
    A[Input parsed for problem 347] --> B[Initialize freq map, buckets, out]
    B --> C[Apply Frequency + buckets]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return first k]
```

3) Longest Substring Without Repeating Characters

Problem statement: Return length of longest substring without repeating characters.

Core idea: Sliding window without repeats.

Key variables: l,r,last,best.

Answer target: return best.

```
flowchart TD
    A[Input parsed for problem 3] --> B[Initialize l,r,last,best]
    B --> C[Apply Sliding window without repeats]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return best]
```

76) Minimum Window Substring

Problem statement: Return minimum window in s containing all chars of t with multiplicity.

Core idea: Expand and contract window.

Key variables: need,have,formed,best.

Answer target: return best window.

```
flowchart TD
    A[Input parsed for problem 76] --> B[Initialize need,have,formed,best]
    B --> C[Apply Expand and contract window]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return best window]
```

424) Longest Repeating Character Replacement

Problem statement: Return longest window transformable to single repeated char with at most k replacements.

Core idea: Window with max freq.

Key variables: l,r,cnt,maxf,best.

Answer target: return best.

```
flowchart TD
    A[Input parsed for problem 424] --> B[Initialize l,r,cnt,maxf,best]
    B --> C[Apply Window with max freq]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return best]
```

567) Permutation in String

Problem statement: Return true if s2 contains any permutation of s1.

Core idea: Fixed-size frequency window.

Key variables: need,win,m.

Answer target: return any match.

```
flowchart TD
    A[Input parsed for problem 567] --> B[Initialize need,win,m]
    B --> C[Apply Fixed-size frequency window]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return any match]
```

125) Valid Palindrome

Problem statement: Return true if string is palindrome after removing non-alphanumeric chars and lowercasing.

Core idea: Two pointers and skip non-alnum.

Key variables: l,r.

Answer target: return bool.

```
flowchart TD
    A[Input parsed for problem 125] --> B[Initialize l,r]
    B --> C[Apply Two pointers and skip non-alnum]
    C --> D{Invariant valid and work remains}
```

```
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return bool]
```

33) Search in Rotated Sorted Array

Problem statement: Find target index in rotated sorted array, else return -1.

Core idea: Binary search on rotated halves.

Key variables: l,r,mid.

Answer target: return index or -1.

```
flowchart TD
    A[Input parsed for problem 33] --> B[Initialize l,r,mid]
    B --> C[Apply Binary search on rotated halves]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return index or -1]
```

153) Find Minimum in Rotated Sorted Array

Problem statement: Find minimum in rotated sorted array.

Core idea: Binary search minimum boundary.

Key variables: l,r,mid.

Answer target: return nums[l].

```
flowchart TD
    A[Input parsed for problem 153] --> B[Initialize l,r,mid]
    B --> C[Apply Binary search minimum boundary]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return nums[l]]
```

875) Koko Eating Bananas

Problem statement: Find minimal integer speed to eat all banana piles within h hours.

Core idea: Binary search answer speed.

Key variables: l,r,mid,hours.

Answer target: return min feasible speed.

```
flowchart TD
    A[Input parsed for problem 875] --> B[Initialize l,r,mid,hours]
    B --> C[Apply Binary search answer speed]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return min feasible speed]
```

981) Time Based Key-Value Store

Problem statement: Implement time-based key-value store with set(key,val,ts) and get(key,ts).

Core idea: Per-key sorted timeline + binary search.

Key variables: store[key]=[(ts,val)].

Answer target: return latest <= ts.

```
flowchart TD
    A[Input parsed for problem 981] --> B[Initialize store[key]=[(ts,val)]]
    B --> C[Apply Per-key sorted timeline + binary search]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return latest <= ts]
```

102) Binary Tree Level Order Traversal

Problem statement: Return binary tree level order traversal.

Core idea: BFS level traversal.

Key variables: queue,level.

Answer target: return levels.

```
flowchart TD
    A[Input parsed for problem 102] --> B[Initialize queue,level]
    B --> C[Apply BFS level traversal]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return levels]
```

199) Binary Tree Right Side View

Problem statement: Return right side view of binary tree.

Core idea: BFS keep last node per level.

Key variables: queue,last.

Answer target: return right view.

```
flowchart TD
    A[Input parsed for problem 199] --> B[Initialize queue,last]
    B --> C[Apply BFS keep last node per level]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return right view]
```

236) Lowest Common Ancestor of a Binary Tree

Problem statement: Return lowest common ancestor of two nodes in a binary tree.

Core idea: Postorder DFS returns matches.

Key variables: left,right.

Answer target: return LCA.

```
flowchart TD
    A[Input parsed for problem 236] --> B[Initialize left,right]
    B --> C[Apply Postorder DFS returns matches]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return LCA]
```

200) Number of Islands

Problem statement: Count number of islands in a 2D grid.

Core idea: Grid flood fill.

Key variables: visited or mutate grid.

Answer target: return island count.

```
flowchart TD
    A[Input parsed for problem 200] --> B[Initialize visited or mutate grid]
    B --> C[Apply Grid flood fill]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return island count]
```

133) Clone Graph

Problem statement: Clone and return deep copy of an undirected connected graph.

Core idea: Graph clone with map.

Key variables: old->new map.

Answer target: return clone.

```
flowchart TD
    A[Input parsed for problem 133] --> B[Initialize old->new map]
    B --> C[Apply Graph clone with map]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return clone]
```

206) Reverse Linked List

Problem statement: Reverse a singly linked list.

Core idea: Pointer reversal.

Key variables: prev,cur,next.

Answer target: return prev.

```
flowchart TD
    A[Input parsed for problem 206] --> B[Initialize prev,cur,next]
    B --> C[Apply Pointer reversal]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return prev]
```

21) Merge Two Sorted Lists

Problem statement: Merge two sorted linked lists.

Core idea: Two-pointer merge.

Key variables: l1,l2,tail.

Answer target: return merged head.

```
flowchart TD
    A[Input parsed for problem 21] --> B[Initialize l1,l2,tail]
    B --> C[Apply Two-pointer merge]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
```

```
    E --> C
    D -- No --> F[return merged head]
```

143) Reorder List

Problem statement: Reorder list as L0 Ln L1 Ln-1 ...

Core idea: Find mid + reverse + weave.

Key variables: slow,fast,second.

Answer target: return reordered list.

```
flowchart TD
    A[Input parsed for problem 143] --> B[Initialize slow,fast,second]
    B --> C[Apply Find mid + reverse + weave]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return reordered list]
```

141) Linked List Cycle

Problem statement: Detect if linked list has a cycle.

Core idea: Floyd cycle detection.

Key variables: slow,fast.

Answer target: return bool.

```
flowchart TD
    A[Input parsed for problem 141] --> B[Initialize slow,fast]
    B --> C[Apply Floyd cycle detection]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return bool]
```

2) Add Two Numbers

Problem statement: Add two numbers represented by reversed linked lists.

Core idea: Digit addition with carry.

Key variables: l1,l2,carry.

Answer target: return summed list.

```
flowchart TD
    A[Input parsed for problem 2] --> B[Initialize l1,l2,carry]
    B --> C[Apply Digit addition with carry]
```

```
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return summed list]
```

56) Merge Intervals

Problem statement: Merge overlapping intervals.

Core idea: Sort then merge.

Key variables: out,last.

Answer target: return merged intervals.

```
flowchart TD
    A[Input parsed for problem 56] --> B[Initialize out,last]
    B --> C[Apply Sort then merge]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return merged intervals]
```

57) Insert Interval

Problem statement: Insert new interval into sorted non-overlapping intervals and merge where needed.

Core idea: Left + merge + right.

Key variables: newInterval,out.

Answer target: return result.

```
flowchart TD
    A[Input parsed for problem 57] --> B[Initialize newInterval,out]
    B --> C[Apply Left + merge + right]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return result]
```

435) Non-overlapping Intervals

Problem statement: Return minimum intervals to remove to make remaining intervals non-overlapping.

Core idea: Greedy by earliest end.

Key variables: sorted intervals,lastEnd,removed.

Answer target: return removed.

```
flowchart TD
    A[Input parsed for problem 435] --> B[Initialize sorted intervals,lastEnd,removed]
    B --> C[Apply Greedy by earliest end]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return removed]
```

452) Minimum Number of Arrows to Burst Balloons

Problem statement: Return minimum arrows to burst all interval balloons.

Core idea: Greedy arrow at current end.

Key variables: sorted points,arrows,end.

Answer target: return arrows.

```
flowchart TD
    A[Input parsed for problem 452] --> B[Initialize sorted points,arrows,end]
    B --> C[Apply Greedy arrow at current end]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return arrows]
```

70) Climbing Stairs

Problem statement: Return number of ways to climb n steps with 1 or 2 step moves.

Core idea: DP Fibonacci recurrence.

Key variables: a,b.

Answer target: return b.

```
flowchart TD
    A[Input parsed for problem 70] --> B[Initialize a,b]
    B --> C[Apply DP Fibonacci recurrence]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return b]
```

198) House Robber

Problem statement: Return max amount robbable from houses without robbing adjacent houses.

Core idea: House robber DP.

Key variables: rob,skip.

Answer target: return max.

```
flowchart TD
    A[Input parsed for problem 198] --> B[Initialize rob,skip]
    B --> C[Apply House robber DP]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return max]
```

322) Coin Change

Problem statement: Return minimum coins to make amount, else -1.

Core idea: Bottom-up DP.

Key variables: dp array.

Answer target: return dp[amount] or -1.

```
flowchart TD
    A[Input parsed for problem 322] --> B[Initialize dp array]
    B --> C[Apply Bottom-up DP]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return dp[amount] or -1]
```

139) Word Break

Problem statement: Return true if s can be segmented into dictionary words.

Core idea: Word break DP.

Key variables: dp positions.

Answer target: return dp[n].

```
flowchart TD
    A[Input parsed for problem 139] --> B[Initialize dp positions]
    B --> C[Apply Word break DP]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return dp[n]]
```

300) Longest Increasing Subsequence

Problem statement: Return length of longest increasing subsequence.

Core idea: Tails + binary search.

Key variables: tails.

Answer target: return len tails.

```
flowchart TD
    A[Input parsed for problem 300] --> B[Initialize tails]
    B --> C[Apply Tails + binary search]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return len tails]
```

39) Combination Sum

Problem statement: Return all unique combinations summing to target with unlimited use of candidates.

Core idea: Backtracking with reuse.

Key variables: path,remain,index.

Answer target: record combinations.

```
flowchart TD
    A[Input parsed for problem 39] --> B[Initialize path,remain,index]
    B --> C[Apply Backtracking with reuse]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[record combinations]
```

46) Permutations

Problem statement: Return all permutations of distinct numbers.

Core idea: Backtracking permutations.

Key variables: path,used.

Answer target: record permutations.

```
flowchart TD
    A[Input parsed for problem 46] --> B[Initialize path,used]
    B --> C[Apply Backtracking permutations]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[record permutations]
```

78) Subsets

Problem statement: Return all subsets of unique numbers.

Core idea: Include/exclude recursion.

Key variables: index,path.

Answer target: record subsets.

```
flowchart TD
    A[Input parsed for problem 78] --> B[Initialize index,path]
    B --> C[Apply Include/exclude recursion]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[record subsets]
```

79) Word Search

Problem statement: Return true if word exists in board by adjacent traversal without reusing cell.

Core idea: DFS with backtracking.

Key variables: idx,r,c,visited.

Answer target: return bool.

```
flowchart TD
    A[Input parsed for problem 79] --> B[Initialize idx,r,c,visited]
    B --> C[Apply DFS with backtracking]
    C --> D{Invariant valid and work remains}
    D -- Yes --> E[Update variables and continue]
    E --> C
    D -- No --> F[return bool]
```