Leetcode Pattern Recognition Guide

Guia aprimorado e traduzido para português. *Os nomes dos tópicos (títulos das seções) foram mantidos em inglês, conforme solicitado.*

Step 1: Check The Constraints

Small n (≤ 20)

- Abordagens de força bruta são viáveis.
- Backtracking e recursão são apropriados.
- Complexidade de tempo exponencial (2^n, n!) pode ser aceitável.
- Tente todas as combinações e permutações, quando fizer sentido.

Medium n (10^3 to 10^6)

- Evite soluções de força bruta.
- Procure soluções em tempo linear `O(n)` ou `O(n log n)`.
- Algoritmos gulosos (greedy), two pointers, heaps e dynamic programming são candidatos comuns.
- Técnicas como divisão e conquista e estruturas de dados eficientes são úteis.

Large n (≥ 10^7)

- Evite algoritmos lineares sempre que possível; prefira soluções logarítmicas ou constantes.
- Busque soluções `O(log n)` ou `O(1)`:
  - Binary search quando aplicável.
  - Fórmulas matemáticas e simplificações analíticas.
  - Abordagens que utilizem propriedades matemáticas do problema.

Step 2: Analyze Input Format

Tree/Binary Tree/BST

- Use `DFS` para percorrer todos os caminhos, exploração recursiva e travessias `preorder/inorder/postorder`.
- Use `BFS` para travessias por nível (level-order) e para achar o caminho mais curto em árvores não ponderadas.
- Considere propriedades específicas da árvore (por exemplo, BST tem ordenação inerente).
- Verifique relações pai-filho e se a solução exige processamento bottom-up ou top-down.

Graph (nodes + edges)

- `BFS` é indicado para o menor caminho em grafos não ponderados.
- `DFS` funciona bem para componentes conectados e busca profunda.
- `Union-Find` (Disjoint Set) é ideal para problemas de componentes conectados e para contar grupos.
- `Topological sort` resolve dependências e ordens válidas em grafos acíclicos dirigidos.

2D Grid/Matrix

- `DFS` / `BFS` são usados frequentemente em problemas de "ilhas" (islands) e preenchimento (flood fill).
- `Union-Find` pode ser usado para regiões conectadas quando se deseja otimização.
- `Dynamic Programming` é útil para problemas de caminhos com custos ou contagem de caminhos.
- Verifique se o movimento é 4-direcional ou 8-direcional e trate limites e visitas corretamente.

Sorted Array

- `Two pointers` para soma-alvo, remoção de duplicados in-place e operações em pares.
- `Binary search` para buscar limites, primeira/última ocorrência ou elementos próximos.
- Abordagem `greedy` quando decisões locais levam a solução ótima global.

String

- `Two pointers` para checagens de palíndromos e operações in-place.
- `Sliding window` para substrings com restrições (tamanho fixo ou variável, sem repetições, anagramas).
- `Trie` para problemas envolvendo dicionários de palavras e buscas por prefixo.
- `Stack` para validação de parênteses, expressões aninhadas e problemas de última entrada/primeira saída.

Linked List

- `Two pointers` (fast/slow) para detectar ciclos, encontrar o meio da lista e remover elementos.
- Uso de `dummy node` para simplificar operações que alteram o head.
- Técnicas de manipulação de ponteiros e cuidado com referências nulas.

Step 3: Analyze Output Format

List of Lists (combinations, subsets, paths)

- Backtracking é a abordagem mais comum para gerar todas as possibilidades.
- Use recursão com padrão escolha / não-escolha e controle de estado (path / visited).
- Preste atenção ao tamanho do output: pode ser exponencial e exigir memória proporcional.

Single Number (max/min profit, cost, ways, jumps)

- `Dynamic Programming` para otimização de máximos/mínimos e contagem de maneiras.
- `Greedy` quando escolhas locais são seguras (provar correção).
- Abordagens matemáticas podem simplificar contagens ou limites.

Modified Array/String (in-place operations)

- `Two Pointers` para modificações in-place, compressão e remoção de

elementos.
• Evite estruturas auxiliares grandes para manter `O(1)` de espaço extra.

Ordered List (sorted sequence, valid task order)

• Ordenação com comparadores personalizados quando necessário.
• `Topological Sort` para ordem válida em tarefas com dependências.
• `Heap` para manter uma ordem dinâmica e obter top-k de forma eficiente.

Step 4: Keyword Pattern Recognition

Dynamic Programming Keywords

Procure por termos que sugiram sobreposição de subproblemas e otimização:

- "number of ways"
- "maximum"/"minimum" combinado com "sum", "profit", "cost"
- "can you reach"
- "longest"/"shortest subsequence"
- "optimal" ou "best"

Two Pointers Keywords

Frases que normalmente indicam técnicas de dois ponteiros:

- "palindrome"
- "sorted array"
- "target sum"
- "remove duplicates"

Heap Keywords

Indicações para usar heaps / priority queues:

- "k largest" ou "k smallest"
- "top k elements"
- "median"
- "priority"

Stack Keywords

Procure por expressões que elogiem estruturas LIFO:

- "parentheses" ou "brackets"
- "valid expression"
- "nested structure"
- "undo operations"

Monotonic Stack Keywords

Quando procurar monotonic stack:

- "next greater element"
- "next smaller element"

HashMap Keywords

Situações para usar mapas/hash:

- "count frequency"
- "find duplicates"
- "anagram"

Trie Keywords

Indica uso de Trie:

- "word search"
- "word prefixes"

Greedy Keywords

Pistas para soluções gulosas:

- "minimum operations"
- Problemas onde uma escolha local leva a solução global

Union Find Keywords

Quando usar Union-Find / DSU:

- "connected components"
- "number of groups"

Binary Search Keywords

Sinais de busca binária ou busca em espaço de resposta:

- "kth element"
- "search in sorted"
- "minimize maximum"
- "first/last occurrence"

Bit Manipulation

Casos para manipulação de bits:

- operações com `XOR`
- problemas "single number"
- verificar "power of 2"

Math/Geometry

Assuntos matemáticos/geometria:

- "greatest/least common denominator"
- "prime numbers"
- "angle calculations"
- "coordinate problems"

Game Theory

Problemas de teoria dos jogos:

- "optimal strategy"
- "win/lose scenarios"
- "minimax"

Sliding Window

Casos típicos:

- "substring" com condições
- "subarray" com tamanho fixo ou variável
- "maximum/minimum window"
- "contains all" (ex.: containing all characters)

Referências rápidas (resumo)

- **Small n (≤ 20):** força bruta, backtracking,

permutações/combinatórias.
• **Medium n (10^3 a 10^6):** prefira `O(n)` ou `O(n log n)`; two pointers, heaps, DP, greedy.
• **Large n (≥ 10^7):** busque `O(log n)` ou `O(1)`; binary search e fórmulas matemáticas.

Observações finais

• Antes de codificar, identifique o *input format*, o *output format* e as *keywords* no enunciado.
• Escolha a técnica com base nas restrições e no formato dos dados.
• Quando em dúvida, escreva uma solução ingênua para entender o problema e depois otimize com o padrão adequado.

Versão adaptada e traduzida a partir do "Leetcode Pattern Recognition Guide" (Bitflip).