Шпаргалка по частым задачам на leetcode / Паттерны и примеры Cheatsheet (XeLaTeX)

Краткий справочник

Содержание

1 HashTable / Словари / Множества (Hash Map / Set)

1.1. Суть Структуры Данных

Академическое Описание

Структура данных, отображающая ключи (keys) на значения (values) с использованием хеш-функции. Обеспечивает в среднем $O(1)$ время для операций вставки, удаления и поиска.

Множества (Sets)

Хранят только уникальные ключи, эффективно проверяя наличие элемента ($O(1)$ в среднем).

1.2. Простое Объяснение

Аналогия

Представьте себе картотеку. У каждой карточки есть уникальный номер (ключ). Вы можете почти мгновенно найти карточку по номеру, добавить новую или убрать старую. Множество (Set) похоже на список уникальных посетителей: можно быст-

ро проверить, есть ли человек в списке.

1.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Ищите необходимость быстро проверить *наличие* элемента, *подсчитать частоту*, *сгруппировать* элементы, найти *пары* с определенным свойством, проверить на *дубликаты* или *анаграммы*.
Ключевые слова: "пара", "сумма", "частота", "группа", "уникальный", "дубликат", "анаграмма", `contains`, `exists`.

1.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С массивами/строками:** Подсчет частот (`Group Anagrams`, `First Unique Character`), поиск пар (`Two Sum`), проверка условий (`Longest Substring...`, `Continuous Subarray Sum`).
- **С Linked Lists:** В реализации `LRU Cache`.
- **С Design:** В `Insert Delete GetRandom O(1)`.
- **С Геометрией:** Хранение координат (`Line Reflection`).

1.5. Пример Python Сниппета

Подсчет частоты / Проверка уникальности

```python
def count_frequency(items):
    counts = {} # или collections.defaultdict(int)
    for item in items:
        counts[item] = counts.get(item, 0) + 1
    return counts
# Пример использования set для проверки уникальности
seen = set()
for item in items:
    if item in seen:
        print(f"Duplicate found: {item}")
    seen.add(item)
```

1.6. Анализ Сложности

Время (среднее)

$O(1)$ для `add`, `remove`, `get`, `in`. Достигается за счет хеш-функции.

Время (худшее)

$O(N)$ при коллизиях (редко с хорошими реализациями).

Память

$O(K)$, где K - количество хранимых уникальных ключей (или пар ключ-значение).

1.7. Задачи из Списка

Примеры Задач LeetCode

Easy: `Two Sum` (*), `Jewels and Stones`, `Intersection of Two Arrays II`, `First Unique Character in a String`, `Missing Number` ● **Medium:** `Group Anagrams` (*), `Subarray Sum Equals K` (*), `Insert Delete GetRandom O(1)` (*), `Longest Substring Without Repeating Characters` (*), `Continuous Subarray Sum`, `Line Reflection` (*)

2 Two Pointers / Скользящее Окно (Sliding Window)

2.1. Суть Техники

Академическое Описание

Техника использования двух (иногда больше) указателей, которые итерируются по структуре данных (массив, строка). Указатели могут двигаться навстречу, в одном направлении, или определять границы "окна".

2.2. Простое Объяснение

Аналогия

Два "пальца", скользящие по данным. Идут навстречу (часто в отсортированных данных) или в одном направлении, определяя "окно".

2.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Работа с *отсортированными* массивами. Поиск *подмассивов/подстрок* с свойствами (макс/мин, уникальность). *Последовательные* элементы, *диапазоны*. Проверка *палиндромов*. *In-place* модификации.
Ключевые слова: "sorted array", "subarray", "substring", "consecutive", "palindrome", "window", "longest", "shortest", "at most K distinct", "in-place".

2.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С HashTable/Set:** В Sliding Window для отслеживания элементов окна (`Longest Substring...`, `Permutation in String`).
- **С Сортировкой:** Предварительный шаг для Two Pointers (`Two Sum II`).
- **С Linked Lists:** В задачах типа `Remove Nth Node From End of List`.

2.5. Пример Python Сниппета

Two Pointers / Sliding Window

```python
# Two Pointers (с двух концов отсорт. массива)
def find_pair_sum(arr, target):
    left, right = 0, len(arr) - 1
    while left < right:
        current_sum = arr[left] + arr[right]
        if current_sum == target: return (left, right)
        elif current_sum < target: left += 1
        else: right -= 1
    return None
# Sliding Window (макс. сумма окна размера k)
def max_subarray_sum(arr, k):
    max_sum = float('-inf')
    current_sum = 0
    window_start = 0
    for window_end in range(len(arr)):
        current_sum += arr[window_end]
        if window_end >= k - 1:
            max_sum = max(max_sum, current_sum)
            current_sum -= arr[window_start]
            window_start += 1
    return max_sum
```

2.6. Анализ Сложности

Время

Обычно $O(N)$. Каждый указатель проходит массив не более раза.

Память

Часто $O(1)$ (in-place). Может быть $O(K)$ для Sliding Window с хранением состояния окна (K - размер алфавита/уникальных элементов).

2.7. Задачи из Списка

Примеры Задач LeetCode

Easy: `Valid Palindrome` (*), `Move Zeroes` (*), `Merge Sorted Array`, `Is Subsequence`, `Squares of a Sorted Array`, `Remove Duplicates from Sorted Array`, `Two Sum II – Input Array Is Sorted`, `Reverse Words in a String III` ● **Medium:** `Longest Subarray of 1's After Deleting One Element` (*), `One Edit Distance` (*), `Permutation in String` (*), `Max Consecutive Ones II` (*), `Maximize Distance to Closest Person` (*), `Find All Anagrams in a String` (*), `Interval List Intersections`, `Max Consecutive Ones III`, `Longest Palindromic Substring` (Expand Around Center), `Longest Substring with At Most Two Distinct Characters` (*), `Partition Labels` (*), `Product of Array Except Self` (*), `Remove Nth Node From End of List` (*) ● **Hard:** `Trapping Rain Water` (*) (Оптимальное O(N))

3 Linked Lists / Связанные Списки

3.1. Суть Структуры Данных

Академическое Описание

Линейная структура: узлы хранят данные и ссылку `next`. В двусвязных - и ссылку `prev`. Доступ по индексу $O(N)$, вставка/удаление при известном узле $O(1)$.

3.2. Простое Объяснение

Аналогия

Цепочка вагонов, каждый знает следующий. Легко вставить-/расцепить, если стоишь у нужного вагона, но долго идти до 5-го вагона.

3.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Явное упоминание "Linked List", "Node", "next". Задачи на *реверс*, *циклы*, *слияние*, *удаление* узлов.

3.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С Two Pointers:** Поиск середины, N-го с конца (`Remove Nth Node...`), палиндром (`Palindrome Linked List`), цикл.
- **С HashTable:** В `LRU Cache`.
- **С Рекурсией:** Реверс, слияние, обход.

3.5. Пример Python Сниппета

Обход / Реверс Списка

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def traverse_list(head):
    current = head
    while current:
        print(current.val)
        current = current.next
# Идея реверса (итеративно)
def reverse_list(head):
    prev = None
    current = head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev
```

3.6. Анализ Сложности

Время

$O(N)$ для поиска/доступа по индексу/обхода. $O(1)$ для вставки/удаления в начале или при известном узле.

Память

$O(N)$ для хранения узлов. Рекурсия может добавить $O(N)$ на стек. Итерация часто $O(1)$ доп. памяти.

3.7. Задачи из Списка

Примеры Задач LeetCode

Easy: `Reverse Linked List` (*), `Merge Two Sorted Lists`, `Palindrome Linked List` ● **Medium:** `Add Two Numbers`, `Remove Nth Node From End of List` (*), `LRU Cache` (*)

4 Stack / Стек

4.1. Суть Структуры Данных

Академическое Описание

Линейная структура данных, работающая по принципу LIFO (Last-In, First-Out). Операции: `push` (добавить), `pop` (удалить сверху).

4.2. Простое Объяснение

Аналогия

Стопка книг: кладешь и берешь сверху. Последняя положенная берется первой.

4.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Обработка в *обратном порядке*. Проверка *сбалансированности* (`()[]{}`). Вычисление *RPN*. Итеративный *DFS*. *Отмена действия* (*backtracking*). Поиск *след. большего/меньшего*.
Ключевые слова: "parentheses", "balanced", "RPN", "backtrack", "next greater element", "simplify path".

4.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С Массивами/Строками:** Обработка символов/чисел (`Valid Parentheses`, `Eval RPN`).
- **С Design:** Реализация очередей/стеков (`Implement Queue...`, `Max Stack`).
- **С Итераторами:** В `Flatten Nested List Iterator`.
- **С DP/Greedy:** Оптимизация (гистограммы в `Maximal Rectangle`).

4.5. Пример Python Сниппета

Проверка Сбалансированности Скобок

```python
def is_valid_parentheses(s):
    stack = []
    mapping = {")": "(", "}": "{", "]": "["}
    for char in s:
        if char in mapping.values():
            stack.append(char)
        elif char in mapping.keys():
            if not stack or mapping[char] !=
            stack.pop():
                return False
    return not stack
```

4.6. Анализ Сложности

Время

$O(1)$ для `push`, `pop`, `peek`. Общее время алгоритма часто $O(N)$.

Память

$O(N)$ в худшем случае (все элементы в стеке).

4.7. Задачи из Списка

Примеры Задач LeetCode

Easy: `Valid Parentheses` (*), `Implement Queue using Stacks`, `Max Stack` (*) ● **Medium:** `Evaluate Reverse Polish Notation`, `Simplify Path`, `Flatten Nested List Iterator` (*) ● **Hard:** `Maximal Rectangle` (*)

5 Trees / Деревья (BST, Binary Tree)

5.1. Суть Структуры Данных

Академическое Описание

Иерархическая структура (узлы, ребра). Бинарное дерево: <= 2 детей. BST: левое < узел < правое. Обходы: BFS (по уровням), DFS (вглубь).

5.2. Простое Объяснение

Аналогия

Семейное древо / структура папок. BST - отсортированный справочник (налево - меньше, направо - больше).

5.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Иерархия. Упоминание "Tree", "Node", "BST". *Обход* (BFS, DFS). *Поиск пути*. *LCA*. *Валидация* (BST, симметрия). *Макс. глубина/-сумма пути*.

5.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С Рекурсией:** DFS и многие задачи (`Validate BST`, `LCA`, `Max Path Sum`).
- **С Очередью (Queue):** Для BFS.
- **С Стеком (Stack):** Для итеративного DFS.

5.5. Пример Python Сниппета

DFS (In-order) / BFS (Level-order)

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val; self.left = left; self.right =
        right
# DFS (In-order - рекурсивный)
def inorder_traversal(root):
    res = []; dfs(root, res); return res
def dfs(node, res):
    if not node: return
```

```
    dfs(node.left, res); res.append(node.val);
    dfs(node.right, res)
# BFS (Level-order - итеративный)
from collections import deque
def level_order(root):
    if not root: return []
    res, q = [], deque([root])
    while q:
        level = []
        # Используем \_ для экранирования в LaTeX
        for \_ in range(len(q)):
            node = q.popleft()
            level.append(node.val)
            if node.left: q.append(node.left)
            if node.right: q.append(node.right)
        res.append(level)
    return res
```

5.6. Анализ Сложности

Время (N узлов, H высота)

$O(N)$ для обходов. В сбалансированном BST поиск/вставка/удаление $O(H) = O(\log N)$. В несбалансированном $O(N)$.

Память (N узлов, H высота, W макс. ширина)

$O(N)$ для хранения дерева. BFS $O(W)$ (до $O(N)$). DFS (рекурсия) $O(H)$ (до $O(N)$). DFS (итерация) $O(H)$.

5.7. Задачи из Списка

Примеры Задач LeetCode

Easy: `Symmetric Tree` (*), `Range Sum of BST` ● **Medium:** `Validate Binary Search Tree` (*), `Lowest Common Ancestor of a Binary Tree` (*), `Lowest Common Ancestor of a Binary Tree III` (*) ● **Hard:** `Binary Tree Maximum Path Sum` (*)

6 Heap / Priority Queue / Куча

6.1. Суть Структуры Данных

Академическое Описание

Древовидная структура (бинарная куча) со свойством кучи: min-heap (узел <= потомки), max-heap (узел >= потомки).

Операции

Эффективное добавление ($O(\log N)$) и извлечение min/max ($O(\log N)$). Просмотр min/max $O(1)$. Priority Queue часто реализуется кучей.

6.2. Простое Объяснение

Аналогия

"Умная" очередь, всегда держит самый "важный" (min/max) элемент сверху. Быстро добавить или забрать самый важный.

6.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Постоянный доступ к *наименьшему/наибольшему*. "Найти *K-ый*" или "топ *K*". *Слияние K* списков. Задачи с *приоритетами*. Ключевые слова: "top K", "smallest/largest", "K-th element", "median" (иногда), "priority", "merge K sorted lists", "meeting rooms II".

6.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С Сортировкой:** В задачах на интервалы (`Meeting Rooms II`).
- **С Массивами/Списками:** Загрузка данных, слияние (`Merge k Sorted Lists`).

6.5. Пример Python Сниппета

Min-Heap / Поиск K наибольших

```
import heapq
# Min-Heap операции
min_heap = []
heapq.heappush(min_heap, 5)
heapq.heappush(min_heap, 1)
smallest = heapq.heappop(min_heap) # 1
# Найти K наибольших (min-heap размера K)
def find_k_largest(nums, k):
    min_heap = []
    for num in nums:
        if len(min_heap) < k:
            heapq.heappush(min_heap, num)
        elif num > min_heap[0]:
            heapq.heapreplace(min_heap, num)
    return list(min_heap)
```

6.6. Анализ Сложности

Время (N элементов)

Вставка (`push`) $O(\log N)$. Извлечение (`pop`) $O(\log N)$. Просмотр (`heap[0]`) $O(1)$. Построение (`heapify`) $O(N)$.

Память

$O(N)$ для хранения всех элементов. $O(K)$ если храним только K элементов.

6.7. Задачи из Списка

Примеры Задач LeetCode

Medium: `Meeting Rooms II` (*) ● **Hard:** `Merge k Sorted Lists` (*)

7 Binary Search / Бинарный Поиск

7.1. Суть Алгоритма

Академическое Описание

Алгоритм эффективного поиска элемента в *отсортированном* массиве (или другой упорядоченной структуре). Работает путем многократного деления интервала поиска пополам.

7.2. Простое Объяснение

Аналогия

Игра "угадай число" в отсортированном списке. На каждой попытке отбрасываешь половину оставшихся вариантов.

7.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Дан *отсортированный* массив. Поиск элемента, *границы*, первой/последней позиции. Поиск в *повернутом (rotated)* отсортированном массиве. Задача сводится к поиску ответа x в монотонном пространстве решений (существует `check(x)`, монотонно меняющая T/F) - можно бинарно искать границу.
Ключевые слова: "sorted array", "find element", "search", "rotated array", "minimum/maximum in rotated", "median of sorted arrays".

7.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С Массивами:** Поиск в отсортированных/повернутых массивах (`Search in Rotated...`, `Find Minimum...`, `Median...`).
- **С Сортировкой:** Часто требует предварительной сортировки.

7.5. Пример Python Сниппета

Классический Бинарный Поиск

```python
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

7.6. Анализ Сложности

Время

$O(\log N)$. На каждом шаге пространство поиска уменьшается вдвое.

Память

$O(1)$ для итеративной реализации. $O(\log N)$ для рекурсивной (стек вызовов).

7.7. Задачи из Списка

Примеры Задач LeetCode

Medium: `Search in Rotated Sorted Array` (*), `Find Minimum in Rotated Sorted Array` (*) ● **Hard:** `Median of Two Sorted Arrays` (*)

8 Graph / Графы (BFS/DFS)

8.1. Суть Структуры Данных и Обходов

Академическое Описание

Структура из вершин (nodes) и ребер (edges). Обход в ширину (BFS) исследует граф по уровням. Обход в глубину (DFS) идет вглубь по ветке до упора, затем возвращается.

8.2. Простое Объяснение

Аналогия

Карта дорог (граф): города - вершины, дороги - ребра. BFS - волны на воде: исследуем сначала ближайших. DFS - блуждание по лабиринту: идем до тупика, возвращаемся.

8.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Задачи на *связи*. Поиск *пути*. *Кратчайший путь* в *невзвешенном* графе (BFS). Обход *матрицы/сетки*. *Циклы*. *Связанные компоненты*. Топологическая сортировка (DFS).
Ключевые слова: "graph", "grid", "matrix", "connected", "path", "shortest path (unweighted)", "cycle", "neighbors", "dependencies", "islands".

8.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С Матрицами:** Граф часто представлен сеткой (`Number of Islands`).
- **С Очередью (Queue):** Для BFS.
- **С Стеком (Stack) / Рекурсией:** Для DFS.
- **С Множествами (Set):** Для отслеживания посещенных вер-

шин.

8.5. Пример Python Сниппета

BFS / DFS на Графе (представлен dict)

```python
from collections import deque
# BFS (поиск пути)
def bfs(graph, start, target):
    q = deque([(start, [start])]); visited = {start}
    while q:
        (v, path) = q.popleft()
        for neighbor in graph.get(v, []):
            if neighbor == target: return path +
            [neighbor]
            if neighbor not in visited:
                visited.add(neighbor)
                q.append((neighbor, path + [neighbor]))
    return None
# DFS (рекурсивный обход)
def dfs(graph, node, visited):
    visited.add(node); print(node) # Обработка
    for neighbor in graph.get(node, []):
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
# visited_set = set(); dfs(my_graph, start_node,
visited_set)
```

8.6. Анализ Сложности

Время (V вершин, E ребер)

$O(V + E)$. Посещаем каждую вершину и ребро константное число раз.

Память (V вершин)

$O(V)$ в худшем случае. Для BFS/DFS нужна очередь/стек и `visited`, хранящие до $O(V)$ элементов.

8.7. Задачи из Списка

Примеры Задач LeetCode

Medium: `Number of Islands` (*) (BFS/DFS на сетке), `Perfect Squares` (BFS на графе состояний)

9 Design / Проектирование Систем

9.1. Суть Задач

Академическое Описание

Задачи, требующие спроектировать и реализовать класс/структуру данных с определенным API и ограничениями по сложности операций. Часто - комбинация стандартных структур.

9.2. Простое Объяснение

Аналогия

Собрать механизм (кэш, итератор) из стандартных "деталей" (словари, списки) так, чтобы он работал быстро ($O(1)$ или $O(\log N)$).

9.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Прямая постановка: "Design...", "Implement...". Требования к *сложности операций* (особенно $O(1)$). Реализация *итераторов*, *кэшей*, *счетчиков*.

9.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

Почти всегда комбинация других паттернов:

- **С HashTable:** Для $O(1)$ доступа/поиска (LRU `Cache`, `InsertDeleteGetRandom`).
- **С Linked Lists:** Для $O(1)$ вставки/удаления/порядка (LRU `Cache`).
- **С Массивами:** Хранение, $O(1)$ доступ по индексу (`InsertDeleteGetRandom`).
- **С Стеком/Очередью:** Итераторы, счетчики, спец. стеки.

9.5. Пример Python Сниппета

Пример: LRU Cache (Идея с OrderedDict)

```python
from collections import OrderedDict
class LRUCache:
    def __init__(self, capacity: int):
        self.cache = OrderedDict(); self.capacity =
        capacity
    def get(self, key: int) -> int:
```

```
        if key not in self.cache: return -1
        self.cache.move_to_end(key); return
        self.cache[key]
    def put(self, key: int, value: int) -> None:
        if key in self.cache:
            self.cache[key] = value;
            self.cache.move_to_end(key)
        else:
            if len(self.cache) >= self.capacity:
                self.cache.popitem(last=False)
            self.cache[key] = value
```

9.6. Анализ Сложности

Временная и Пространственная Сложность

Анализируется для каждой операции (`get`, `put`, etc.) отдельно. Цель — удовлетворить ограничениям. Определяется сложностью базовых структур.

9.7. Задачи из Списка

Примеры Задач LeetCode

Easy: `Number of Recent Calls` (*), `Max Stack` (*) ● **Medium:** `Zigzag Iterator` (*), `Insert Delete GetRandom O(1)` (*), `LRU Cache` (*), `Design Hit Counter` (*), `Flatten Nested List Iterator` (*)

10 Recursion / Backtracking / DP

10.1. Суть Подходов

Рекурсия

Функция вызывает сама себя для решения подзадачи меньшего размера. Требует базового случая.

Backtracking

Систематический перебор кандидатов. Строит решение пошагово; если шаг неудачен, "откатывается" (backtracks).

Динамическое Программирование (DP)

Решение через разбиение на *перекрывающиеся подзадачи*. Решения подзадач сохраняются (мемоизация или табуляция).

10.2. Простое Объяснение

Аналогии

Рекурсия: Матрешка. ● **Backtracking:** Идти по лабиринту. ● **DP:** Строить из Lego, повторно используя мелкие блоки.

10.3. Распознавание Паттерна

Сигналы: Рекурсия/Backtracking

Структура задачи рекурсивна (деревья). Генерация *всех* перестановок, комбинаций, подмножеств. Поиск *всех* путей. Головоломки.
Ключевые слова: "generate all", "find all combinations/permutations/subsets".

Сигналы: Динамическое Программирование

Найти *оптимальное* (min/max, longest/shortest). Подсчитать *количество способов*. Задача разбивается на *пересекающиеся* подзадачи.
Ключевые слова: "minimum/maximum cost/path/value", "longest/shortest", "number of ways".

10.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С Деревьями:** Рекурсия - основной способ обхода (`Max Path Sum`).
- **С Массивами/Строками:** DP для последовательностей. Backtracking для комбинаций.
- **С Стеком:** Итеративный Backtracking.
- **С HashTable/Массивом:** Мемоизация в DP.

10.5. Пример Python Сниппета

Рекурсия / Backtracking / DP (Мемоизация)

```
# Рекурсия (Факториал)
def factorial(n):
    if n == 0: return 1; return n * factorial(n - 1)
# Backtracking (Подмножества)
def generate_subsets(nums):
    res, subset = [], []
    def backtrack(start):
        res.append(subset[:])
        for i in range(start, len(nums)):
            subset.append(nums[i]); backtrack(i + 1);
            subset.pop()
    backtrack(0); return res
# DP (Фибоначчи с мемоизацией)
memo = {}
def fib(n):
    if n in memo: return memo[n]
    if n <= 1: return n
    memo[n] = fib(n - 1) + fib(n - 2); return memo[n]
```

10.6. Анализ Сложности

Время: Рекурсия/Backtracking

Часто экспоненциальное $O(c^N)$ или факториальное $O(N!)$. Зависит от ветвления и глубины.

Время: DP

Обычно полиномиальное ($O(N)$, $O(N^2)$, $O(N \cdot M)$). (Кол-во подзадач) * (Время решения одной).

Память: Рекурсия/Backtracking

$O(H)$, где H - макс. глубина рекурсии (часто $O(N)$ для стека вызовов).

Память: DP

$O(S)$, где S - кол-во состояний (размер таблицы/мемо, часто $O(N)$ или $O(N \cdot M)$).

10.7. Задачи из Списка

Примеры Задач LeetCode

Medium: `Generate Parentheses` (*) (Backtracking), `Perfect Squares` (DP или BFS) ● **Hard:** `Binary Tree Maximum Path Sum` (*) (Рекурсия), `Maximal Rectangle` (*) (DP)

11 Math / Geometry / Bit Manipulation / Other

11.1. Суть Категории

Академическое Описание

Задачи, решение которых опирается на математические теоремы, формулы, геометрию, побитовые операции или специфические трюки, не укладывающиеся в стандартные паттерны.

11.2. Простое Объяснение

Аналогия

Иногда нужна школьная математика, геометрия или хитрые трюки с числами/битами.

11.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Работа с *координатами*, точками, линиями. *Делимость*, простые числа. Манипуляции с *битами*. *Вероятность*. Прямая *симуляция*. *Сжатие строк*.
Ключевые слова: "coordinates", "geometry", "prime", "bits", "random", "probability", "simulate", "compress".

11.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

- **С HashTable:** Хранение геом. объектов/промежут. результатов (`Line Reflection`).
- **С Массивами/Строками:** Основа для мат. манипуляций/симуляций (`String Compression`).

11.5. Пример Python Сниппета

Проверка на степень двойки / Rand10() Идея

```python
# Проверка на степень двойки (битовая магия)
def is_power_of_two(n):
    return n > 0 and (n & (n - 1) == 0)
# Идея генерации Rand10 из Rand7 (Rejection Sampling)
# def rand10():
#     while True:
#         num = (rand7() - 1) * 7 + rand7() # 1..49
#         if num <= 40: return (num - 1) % 10 + 1
```

11.6. Анализ Сложности

Временная и Пространственная Сложность

Сильно варьируется: от $O(1)$, $O(\log N)$ (мат./бит.) до $O(N)$, $O(N^2)$ (симуляции, геометрия). Анализируется индивидуально.

11.7. Задачи из Списка

Примеры Задач LeetCode

Easy: `Summary Ranges` (*), `Consecutive Characters`, `Add Strings` ● **Medium:** `Line Reflection` (*), `String Compression` (*), `Implement Rand10() Using Rand7()` (*)

12 Sorting / Сортировка

12.1. Суть Процесса

Академическое Описание

Процесс упорядочивания элементов коллекции. Часто - **предварительный шаг** для других алгоритмов. Эффективные алгоритмы сравнения: $O(N \log N)$.

12.2. Простое Объяснение

Аналогия

Привести данные в порядок (как слова в словаре), чтобы легче работать дальше.

12.3. Распознавание Паттерна

Сигналы и Ключевые Слова

Явно требуется *отсортированный* вывод. Применение алгоритма для *отсортированных* данных (Binary Search, Two Pointers). Задачи с *интервалами* (слияние, пересечения). *Анаграммы* (сортировка строк).

12.4. Частые Комбинации

Комбинации с Другими Паттернами (в вашем списке)

Является **предварительным шагом** для:

- **Two Pointers:** Поиск пар, обработка.
- **Binary Search:** Обязательное условие.
- **Heap:** Иногда используется вместе (`Meeting Rooms II`).
- **Greedy:** Часто требует сортировки (`Merge Intervals`).

12.5. Пример Python Сниппета

Встроенная Сортировка / Сортировка по Ключу

```python
# Сортировка на месте
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
my_list.sort() # -> [1, 1, 2, 3, 4, 5, 6, 9]
# Создание нового отсортированного списка
my_tuple = (3, 1, 4, 1, 5)
sorted_list = sorted(my_tuple) # -> [1, 1, 3, 4, 5]
# Сортировка по ключу (по второму элементу)
data = [(1, 5), (3, 2), (2, 8)]
data.sort(key=lambda x: x[1]) # -> [(3, 2), (1, 5), (2, 8)]
```

12.6. Анализ Сложности

Время

$O(N \log N)$ для эффективных алгоритмов сравнения (Merge Sort, Heap Sort, Quick Sort avg, Timsort). Counting/Radix Sort могут быть $O(N)$ при ограничениях.

Память

$O(1)$ (in-place Heap Sort) до $O(N)$ (Merge Sort, Timsort). `sort()` стремится к O(N) в худшем, `sorted()` всегда $O(N)$ доп. памяти.

12.7. Задачи из Списка

Примеры Задач LeetCode

Medium: `Merge Intervals` (*) (как первый шаг), `Meeting Rooms II` (*) (как первый шаг)