ChatGPT - DSA Mastery - Python

String

Below is your **String DSA Mastery – Python Edition**, crafted in the same style as your other DSA roadmaps (DP, Trees, Graphs, LL, Bit, CP).
This gives you:

✅ Patterns
✅ When to apply
✅ How to identify in interviews
✅ Core templates (Python)
✅ 0–100 roadmap
✅ Problem-classification cheat-sheet

🔥 STRING DSA MASTERY — PATTERNS, TEMPLATES & ROADMAP (Python)

1️⃣ Why string problems feel tricky

String problems often mix:

- Indexing + sliding windows
- Hashing (freq maps)
- Two pointers
- Greedy
- Stack
- DP (subsequences, palindromes)
- String hashing / KMP / Z-algorithm

Once you know **patterns**, 90% of problems fall into predictable buckets.

2️⃣ String Patterns (Ultimate Cheat-Sheet)

Pattern 1: Frequency Counting + Hash Maps

When used?

- Anagrams

- Isomorphic strings
- Character replacement
- Count substrings w/ constraints
- Palindrome permutation checks

Identify:

- Words like: “same characters”, “rearrange”, “permutation”, “anagram”, “frequency”

Template (Python)

python

```
from collections import Counter

freq = Counter(s)
freq[c] += 1
```

Pattern 2: Two Pointers

When used?

- Remove adjacent duplicates
- Compare strings after backtracking (#)
- Valid palindrome with modifications

Identify:

- “Move from left and right”
- “Skip characters”
- “Check substring without rebuilding string”

Template

python

```
l, r = 0, len(s) - 1
while l < r:
    if s[l] != s[r]:
        return False
    l += 1
    r -= 1
```

Pattern 3: Sliding Window

When used?

- Longest substring without repeating characters
- Minimum window substring
- Fixed window size substring checks

Identify:

- “Longest substring…”
- “Smallest substring…”
- “K distinct characters”
- “Window moves left to right”

Template

```python
freq = {}
l = 0

for r, ch in enumerate(s):
    freq[ch] = freq.get(ch, 0) + 1

    # while window invalid → shrink
    while condition_to_shrink:
        freq[s[l]] -= 1
        l += 1
```

Pattern 4: Stack-based String Problems

Used for

- Decode string “3[a2[c]]”
- Remove adjacent duplicates
- Simplify path
- Backspace operations

Template

```python
stack = []
for ch in s:
    if stack and stack[-1] == ch:
        stack.pop()
    else:
        stack.append(ch)
return "".join(stack)
```

Pattern 5: Greedy String Construction

Used for

- Check palindrome formation
- Lexicographically smallest string
- Removing k characters

Identify

- “Lexicographically smallest”
- “Remove k characters”
- “Rearrange to be smallest/largest”

Pattern 6: Dynamic Programming

Sub-patterns:

DP1 – Longest Common Subsequence (LCS)

Used for:

- LCS
- Min insertions to make palindrome
- Edit distance

DP2 – Palindromic Substrings

Used for:

- Count palindromic substrings
- Longest palindromic substring
- Palindrome partitioning

Pattern 7: String Searching Algorithms

Used for:

- Pattern search
- Repeated substring
- Prefix-suffix problems
- “Find pattern in text”

Includes:

- **KMP (Prefix function)**

- **Rabin-Karp (Hashing)**
- **Z-Algorithm**

Pattern 8: Trie

Used for:

- Word search
- Autocomplete
- Longest common prefix
- Prefix queries

Pattern 9: Sorting-based

Used for:

- Group anagrams
- Sort characters by frequency

Pattern 10: Backtracking

Used for:

- Restore IP addresses
- Generate letter combinations
- Permutations of string

3️⃣ 0–100 String Roadmap (Beginner → Advanced)

Level 0–10: Basics

- String slicing
- Reverse string
- Check palindrome
- Character frequency

Level 10–20: Two pointers

- Valid palindrome
- Palindrome II

- Reverse prefix

Level 20–35: Sliding Window

- Longest substring without repeating
- Longest repeating replacement
- Minimum window substring

Level 35–50: Hashing + Anagram Patterns

- Group anagrams
- Find all anagrams in string
- Isomorphic strings

Level 50–65: Stack Patterns

- Remove duplicates
- Decode string
- Backspace compare

Level 65–80: DP on Strings

- LCS
- Edit distance
- Longest palindromic substring
- Palindrome partitioning

Level 80–95: String Algorithms

- KMP prefix table
- Z-algorithm
- Rabin-Karp

Level 95–100: Serious Interview Level

- Hard DP string problems
- String automata
- Lexicographically minimal rotations
- Suffix arrays (optional)

4️⃣ Python Templates for All Important Algorithms

✔️ KMP Prefix Table

```python
def prefix(s):
    n = len(s)
    pi = [0]*n
    j = 0

    for i in range(1, n):
        while j > 0 and s[i] != s[j]:
            j = pi[j-1]
        if s[i] == s[j]:
            j += 1
        pi[i] = j

    return pi
```

✔️ Rabin-Karp (Rolling Hash)

```python
def rabin_karp(text, pattern):
    if len(pattern) > len(text):
        return -1

    base = 256
    mod = 10**9 + 7

    m, n = len(pattern), len(text)
    ph = th = 0
    power = pow(base, m-1, mod)

    for i in range(m):
        ph = (ph*base + ord(pattern[i])) % mod
        th = (th*base + ord(text[i])) % mod

    for i in range(n-m+1):
        if ph == th:
            if text[i:i+m] == pattern:
                return i
        if i+m < n:
            th = ((th - ord(text[i])*power)*base + ord(text[i+m])) % mod

    return -1
```

✔️ Longest Palindromic Substring (Expand Around Center)

```python
def longestPalindrome(s):
    res = ""

    def expand(l, r):
        while l >= 0 and r < len(s) and s[l] == s[r]:
            l -= 1
            r += 1
        return s[l+1:r]

    for i in range(len(s)):
        odd  = expand(i, i)
        even = expand(i, i+1)
        res = max(res, odd, even, key=len)

    return res
```

5️⃣ String Problem → Pattern Classification (Interview Guide)

Problem Type	Identify From	Use Pattern
“Longest substring…”	non-repeating, k distinct	Sliding window
“Is anagram?”	same characters	Hash map
“Remove adjacent”	duplicates	Stack
“Compare processed string”	backspace	Stack
“Check palindrome”	two-sided	Two pointers
“Smallest substring containing…”	contains all chars	Sliding window
“Pattern search”	find occurences	KMP / hashing
“Construct smallest/largest string”	lexicographically	Greedy
“Edit distance / LCS”	convert string into another	DP
“Partition string”	cuts / palindromes	DP or backtracking
“Repeated substring pattern”	prefix = suffix	KMP

6️⃣ Want me to generate 👉

✔️ 100 String Problems Mapped to Patterns

✔️ Interview-style Templates

✔️ Detailed Cheat-sheet PDF

✔️ Pattern-wise drills (beginner → advanced)

Just tell me: **“Generate full String DSA pack”** or specify any part.

Prefix, Suffix Patterns

Below is your **Prefix–Suffix Patterns Mastery Cheat-Sheet for String DSA**, designed in the same style as your DSA Mastery collections (LinkedList, Graph, DP, Bit, Stack/Queue).

This is the **ultimate guide** to identifying, solving, and coding prefix/suffix–based problems.

🔥 PREFIX–SUFFIX PATTERNS — DSA MASTERY (Python)

Why Prefix/Suffix Patterns Matter

They appear in **KMP**, **string matching**, **longest repeating substring**, **border problems**, **prefix arrays**, **Z-algorithm**, **palindrome checks**, **prefix/suffix sum problems**, etc.

Nearly all advanced string algorithms rely on:
Longest proper prefix = suffix (LPS or π array).

1️⃣ Prefix → Suffix Pattern Types

Prefix–suffix problems fall into these categories:

Pattern 1: KMP Prefix Table (LPS)

Used for:

- Pattern matching
- Repeated substring check
- Detect borders
- Longest prefix that’s also suffix

Pattern 2: Z-Algorithm (Longest prefix matches at each index)

Used for:

- Pattern matching
- Repeated substring
- String rotations
- Prefix lookups

Pattern 3: Hashing (Prefix Hash + Suffix Hash)

Used for:

- Palindrome checks
- Finding repeated substrings fast
- Substring equality
- Rolling hash

Pattern 4: Prefix/Suffix Sum Arrays

Used for:

- Range operations
- Replacement operations
- Balancing parentheses
- Prefix difference trick

Pattern 5: Longest Prefix Also a Suffix

Used for:

- Repeated substring pattern
- Smallest rotation
- Border problems

Pattern 6: Prefix Expansion

Used for:

- Word break type
- Prefix tries
- Autocomplete

2️⃣ How to Identify Prefix–Suffix Problems (Interview Clues)

Problem Hint	Pattern
“prefix appears again later in string”	KMP or Z
“find longest prefix also suffix”	LPS
“pattern search inside text”	KMP or Z
“string is made by repeating substring”	LPS
“check rotations / cyclic”	Z / KMP
“equal substrings?”	hash
“check palindrome substring fast”	prefix+suffix hash
“range update / prefix increments”	prefix sum

3️⃣ KMP PREFIX TABLE (LPS Array) — Core Pattern

🔹 LPS Meaning

For string `s`, `lps[i]` = length of the **longest proper prefix which is also a suffix** for substring `s[:i+1]`.

🔹 Core Template (Python)

python

```
def build_lps(s):
    n = len(s)
    lps = [0] * n
    j = 0  # length of prefix match

    for i in range(1, n):
        while j > 0 and s[i] != s[j]:
            j = lps[j - 1]

        if s[i] == s[j]:
            j += 1

        lps[i] = j

    return lps
```

4️⃣ KMP Pattern Search Using Prefix Table

python

```
def kmp_search(text, pattern):
    lps = build_lps(pattern)
    i = j = 0

    while i < len(text):
        if text[i] == pattern[j]:
            i += 1
            j += 1

            if j == len(pattern):
                return True
        else:
            if j > 0:
                j = lps[j - 1]
            else:
                i += 1

    return False
```

5️⃣ Classic Prefix–Suffix Problems (with LPS Patterns)

✔️ Problem 1: Longest Prefix Also a Suffix

python

```
lps = build_lps(s)
ans = lps[-1]
```

✔️ Problem 2: Detect Repeated Substring Pattern

String `s` is repeated if:

`lps[-1] > 0` and
`len(s) % (len(s) - lps[-1]) == 0`

```python
def repeatedSubstringPattern(s):
    lps = build_lps(s)
    n = len(s)
    return lps[-1] > 0 and n % (n - lps[-1]) == 0
```

✔️ Problem 3: Count Borders of a String

All borders = repeatedly follow LPS.

```python
def all_borders(s):
    lps = build_lps(s)
    borders = []
    j = lps[-1]
    while j > 0:
        borders.append(j)
        j = lps[j-1]
    return borders
```

✔️ Problem 4: Shortest Palindrome Using Prefix–Suffix (KMP Trick)

Make `s + "#" + reverse(s)`
The border of this tells palindrome prefix.

```python
def shortestPalindrome(s):
    t = s + "#" + s[::-1]
    lps = build_lps(t)
    return s[lps[-1]:][::-1] + s
```

6️⃣ Z-Algorithm Pattern (Prefix Matching at Every Position)

✔️ Z-array Meaning

`Z[i] = length of longest substring starting at i that matches prefix of S`

Template

```python
def z_array(s):
    n = len(s)
    z = [0] * n
    l = r = 0
    for i in range(1, n):
        if i <= r:
            z[i] = min(r - i + 1, z[i - l])
        while i + z[i] < n and s[z[i]] == s[i + z[i]]:
            z[i] += 1
        if i + z[i] - 1 > r:
            l, r = i, i + z[i] - 1

    return z
```

Applications of Z for Prefix–Suffix

✔️ Pattern Search

`pattern + "$" + text`

✔️ Find repeated prefix

Z-values equal to prefix length = repeated prefix positions

✔️ Count occurrences of prefix

Just count entries in Z where `z[i] == len(prefix)`.

7️⃣ Prefix & Suffix Hashing (Rabin–Karp Rolling Hash)

Template for prefix hash array:

```
python

def prefix_hash(s):
    n = len(s)
    mod = 10**9 + 7
    base = 131

    ph = [0] * (n + 1)
    power = [1] * (n + 1)

    for i in range(n):
        ph[i+1] = (ph[i] * base + ord(s[i])) % mod
        power[i+1] = (power[i] * base) % mod

    return ph, power

def substring_hash(ph, power, l, r):
    return (ph[r] - ph[l] * power[r-l]) % mod
```

Used for:

- Palindrome check
- Substring equality in O(1)
- Longest repeated substring (binary search + hash)

8️⃣ Prefix–Suffix Sums Pattern (not string-specific but common)

Prefix sum array

```
python

pref[i] = sum of a[0..i]
```

Suffix sum array

```
python

suff[i] = sum of a[i..n-1]
```

Applications:

- Balance parentheses
- Range sum queries
- Difference arrays

9️⃣ Prefix–Suffix Interview Problem Classification

Problem Name	Pattern Used
Repeated Substring Pattern	LPS
Longest Happy Prefix	LPS
Shortest Palindrome	KMP
Count Prefix Repetitions	Z
Pattern Search	KMP / Z
Smallest Rotation	Z / Suffix array
Check string rotations	KMP / hashing
Compare substrings	prefix hashing
Longest Repeat Substring	hashing (BS)
Palindrome test	prefix + suffix hash

🔟 Ultra-Short Prefix–Suffix Recognition Rules

If question says:

✔️ “Longest prefix which is suffix” → **LPS**
✔️ “Pattern inside text” → **KMP / Z**
✔️ “Compare substrings” → **Hashing**

✔️ "Repeated pattern" → **LPS or Z**
✔️ "Check if rotation" → **KMP on s + s**
✔️ "Longest prefix match starting at i" → **Z-array**
✔️ "Check palindrome fast" → **prefix hash = reverse suffix hash**
✔️ "Border of string" → **LPS**

Want the Prefix–Suffix 75 Problem Set grouped by patterns?

Or a **PDF cheat-sheet**?

String Coding Patterns

Below is your 🔥 **STRING CODING PATTERNS — DSA MASTERY PACK**, designed exactly in the style of your previous master cheat-sheets (DP, Graph, LL, Bit, Stack, etc.).
This pack contains:

✅ 20+ core patterns
✅ How to identify each pattern in interviews
✅ Python templates
✅ Famous problems mapped to each pattern
✅ Micro–logic notes + tricky corner cases

🔥 STRING CODING PATTERNS (Ultimate Cheat-Sheet)

1️⃣ Sliding Window Patterns

Used for:

- Longest substring problems
- K distinct characters
- Minimum window substring
- Anagram windows

Pattern 1: Variable Window Size

When?
"Longest substring … constraint"

Template

```python
freq = {}
l = 0
best = 0

for r, ch in enumerate(s):
    freq[ch] = freq.get(ch, 0) + 1

    while violation(freq):
        freq[s[l]] -= 1
        l += 1

    best = max(best, r - l + 1)
```

Problems

- Longest substring without repeating characters
- Longest substring with K distinct
- Longest repeating char replacement

Pattern 2: Fixed Window Size

When?
“Substring of length k…”

```python
freq = {}
l = 0

for r in range(len(s)):
    freq[s[r]] = freq.get(s[r], 0) + 1

    if r - l + 1 > k:
        freq[s[l]] -= 1
        l += 1
```

Problems

- Find anagrams in string
- Count substrings with equal vowels

2️⃣ Two-Pointer Patterns

Used for:

- Palindrome checks
- Compare modified strings
- Removing characters

Pattern 3: Bidirectional Two Pointers

```
python

l, r = 0, len(s)-1
while l < r:
    if s[l] != s[r]:
        return False
    l += 1
    r -= 1
return True
```

Problems

- Valid Palindrome
- Valid Palindrome II
- Reverse Words in place

Pattern 4: One Forward Pointer + Jump

Used in:

- Skip # (backspace compare)
- Remove adjacent chars without stack

```
python

i, skip = len(s)-1, 0
while i >= 0:
    if s[i] == '#':
        skip += 1
    elif skip > 0:
        skip -= 1
    else:
        yield s[i]
    i -= 1
```

3️⃣ Hash Map Patterns

Used for:

- Anagrams
- Frequency-matching problems
- Isomorphic strings

Pattern 5: Frequency Match

python

```python
from collections import Counter
Counter(s1) == Counter(s2)
```

Problems

- Valid Anagram
- Group Anagrams
- Find all anagrams in string

Pattern 6: Hashing Characters to First Occurrence

Used for:

- Isomorphic Strings
- Pattern matching “abba” with words

python

```python
mapping = {}
for i, ch in enumerate(s):
    if ch not in mapping:
        mapping[ch] = i
```

4️⃣ Stack-Based String Patterns

Used for:

- Parsing
- Nested structures
- Remove duplicates
- Decode string

Pattern 7: Remove Adjacent Duplicates

python

```python
stack = []
for ch in s:
    if stack and stack[-1] == ch:
        stack.pop()
    else:
        stack.append(ch)
return ''.join(stack)
```

Pattern 8: Decode String (3[a2[c]])

```python
stack = []
num = 0

for ch in s:
    if ch.isdigit():
        num = num * 10 + int(ch)
    elif ch == '[':
        stack.append((num, ""))
        num = 0
    elif ch == ']':
        n, prev = stack.pop()
        prev += n * curr
        curr = prev
    else:
        curr += ch
```

5️⃣ Prefix–Suffix & KMP Patterns

Used for:

- Repeated substring pattern
- Longest prefix that is suffix
- String search

Pattern 9: KMP Prefix Table (LPS)

```python
def build_lps(s):
    lps = [0]*len(s)
    j = 0
    for i in range(1, len(s)):
        while j > 0 and s[i] != s[j]:
            j = lps[j-1]
        if s[i] == s[j]:
            j += 1
        lps[i] = j
    return lps
```

Problems

- Repeated substring pattern
- Longest happy prefix
- Pattern search

Pattern 10: Z-Algorithm

```python
def z_algo(s):
    z = [0]*len(s)
    l = r = 0
    for i in range(1, len(s)):
        if i <= r:
            z[i] = min(r - i + 1, z[i-l])
        while i+z[i] < len(s) and s[z[i]] == s[i+z[i]]:
            z[i] += 1
        if i + z[i] - 1 > r:
            l, r = i, i + z[i] - 1
    return z
```

6️⃣ Prefix/Suffix Hashing Patterns (Rabin–Karp)

Used for:

- Substring comparison
- Longest palindrome by hashing
- Fast rotation checks

Pattern 11: Prefix Hash

```python
def prefix_hash(s):
    mod = 10**9 + 7
    base = 131
    n = len(s)

    ph = [0]*(n+1)
    powb = [1]*(n+1)

    for i in range(n):
        ph[i+1] = (ph[i]*base + ord(s[i])) % mod
        powb[i+1] = (powb[i]*base) % mod

    return ph, powb
```

7️⃣ Dynamic Programming Patterns

Pattern 12: DP – Palindrome Expand Around Center

```python
def expand(i, j):
    while i >= 0 and j < n and s[i] == s[j]:
        i -= 1
        j += 1
    return j - i - 1
```

Problems

- Longest palindromic substring
- Count palindromic substrings

Pattern 13: DP – Subsequence (LCS / Edit Distance)

LCS Template:

```python
dp = [[0]*(m+1) for _ in range(n+1)]

for i in range(n):
    for j in range(m):
        if s1[i] == s2[j]:
            dp[i+1][j+1] = dp[i][j] + 1
        else:
            dp[i+1][j+1] = max(dp[i][j+1], dp[i+1][j])
```

Pattern 14: DP – Partitioning

Used in:

- Palindrome partitioning
- Word break

8️⃣ Trie Patterns

Used for:

- Prefix matching
- Word dictionary
- Longest common prefix
- Auto-complete

Pattern 15: Basic Trie Insert/Search

```python
class Trie:
    def __init__(self):
        self.child = {}
        self.end = False

    def insert(self, word):
        cur = self
        for c in word:
            cur = cur.child.setdefault(c, Trie())
        cur.end = True
```

9️⃣ Greedy String Patterns

Used for:

- Lexicographically smallest string
- Remove k digits
- Construct smallest string from freq

Pattern 16: Monotonic Stack for Lexicographic Removal

```python
stack = []
for ch in s:
    while k and stack and stack[-1] > ch:
        stack.pop()
        k -= 1
    stack.append(ch)
```

🔟 Backtracking Patterns

Used for:

- Restore IP
- Letter combinations
- Phone keypad
- Permutations of string

Pattern 17: Standard Backtracking

```python
res = []

def dfs(path, idx):
    if idx == len(s):
        res.append(path)
        return

    for c in choices:
        dfs(path + c, idx+1)
```

1️⃣1️⃣ String Compression / Construction Patterns

Pattern 18: Run-Length Encoding

```python
res = []
count = 1
for i in range(1, len(s)):
    if s[i] == s[i-1]:
        count += 1
    else:
        res.append(s[i-1] + str(count))
        count = 1
res.append(s[-1] + str(count))
return ''.join(res)
```

1️⃣2️⃣ Matrix + String Patterns

Used for:

- Word search
- DFS paths
- Trie + grid

🧩 MASTER STRING PATTERN TABLE

Pattern Type	When to Use	Core Technique
Sliding Window	Longest/Shortest substring	Window + freq
Two Pointers	Palindrome, compare	l++, r--
Hash Maps	Anagrams, isomorphic	freq maps
Stack	Decode, remove dup	push/pop
Prefix–Suffix	repeated/rotation	LPS or Z
Hashing	Compare substrings	rolling hash
DP	LCS, Edit, Palindromes	table/expand
Trie	prefix problems	multi-branch tree
Greedy	lexicographically best	monotonic stack
Backtracking	generate strings	dfs

🎯 Want the FULL PACKAGE?

I can generate:

✅ **100 String Problems grouped by patterns**
✅ **Pattern-wise coding templates**

✅ **Interview-style solutions**
✅ **String DSA Roadmap PDF**

Just say:
"Generate String 100-Problem Pattern Set" or
"Make this into a PDF cheat-sheet".

Would you like that?

✅ 100 String Problems grouped by patterns

Here is your 🔥 **100 STRING DSA PROBLEMS — GROUPED BY PATTERNS**
This is the **interview-optimized master list**, aligned with your DSA Mastery structure.

Perfect for **practice** → **mastery** → **interviews** → **competitive coding**.

🔥 STRING PATTERN MASTERLIST — 100 Problems (Grouped by Patterns)

➡️ **12 String Patterns**
➡️ **100 curated problems**
➡️ **Zero duplicates**
➡️ Covers LC, GFG, CP, FAANG patterns

1️⃣ Sliding Window (Variable + Fixed Window) — *15 Problems*

Longest/Shortest substring & frequency windows

1. Longest Substring Without Repeating Characters
2. Longest Substring with At Most K Distinct Characters
3. Longest Substring with At Most Two Distinct Characters
4. Longest Repeating Character Replacement
5. Minimum Window Substring
6. Find All Anagrams in a String
7. Permutation in String (Check Inclusion)
8. Substrings of Size K with K Distinct Characters
9. Count Occurrences of Anagrams
10. Longest Nice Substring
11. Binary Substrings With Sum Constraint

12. Substrings with K Distinct Vowels
13. Fruits into Baskets
14. Max Consecutive Ones III
15. Longest Balanced Substring after K flips

2️⃣ Two Pointers (Forward, Bidirectional) — *10 Problems*

Palindrome checks, reversed strings, comparisons

16. Valid Palindrome
17. Valid Palindrome II (one removal)
18. Reverse Words in a String
19. Reverse Words in a String III
20. Backspace String Compare
21. Compare Strings with Skip Characters
22. Move Vowels to End / Front
23. String Compression II (pointer-based)
24. Merge Strings Alternately
25. Long Pressed Name

3️⃣ Hash Map / Character Frequency Patterns — *10 Problems*

Anagrams, isomorphic, mappings

26. Valid Anagram
27. Group Anagrams
28. Isomorphic Strings
29. Word Pattern
30. Find Common Characters
31. First Unique Character in a String
32. Custom Sort String
33. Frequency Sort
34. Ransom Note
35. Maximum Occurring Character (frequency-based)

4️⃣ Stack-Based String Problems — *10 Problems*

Decode, parse, remove duplicates

36. Remove All Adjacent Duplicates in String
37. Remove All Adjacent Duplicates in String II (count-based)
38. Decode String (3[a2[c]])
39. Simplify Path
40. Minimum Remove to Make Valid Parentheses
41. Make Parentheses Valid
42. Remove Outermost Parentheses
43. Score of Parentheses
44. Basic Calculator II (string + stack)
45. Postfix Expression Evaluator

5️⃣ Prefix–Suffix / KMP Patterns — *10 Problems*

Borders, repeated substrings, pattern search

46. Implement KMP (Prefix Function / LPS Array)
47. Longest Happy Prefix
48. Repeated Substring Pattern
49. Shortest Palindrome
50. Find Pattern in Text using KMP
51. Count Prefix Repetitions
52. Cyclic String Shifts Check
53. Detect String Rotation using KMP
54. Longest Prefix Which Is Also Suffix
55. Remove Border Characters

6️⃣ Z-Algorithm Patterns — *5 Problems*

Prefix matching at each index

56. Z-Algorithm Pattern Search
57. Count Prefix Occurrences
58. Find Repeating Prefix Blocks
59. Smallest String Rotation (Z trick)
60. Border Lengths via Z-array

7️⃣ String Hashing (Rabin–Karp / Double Hash) — *10 Problems*

Faster substring checks, palindrome hashing

61. Implement Rabin–Karp
62. Find All Occurrences of a Pattern
63. Longest Repeated Substring (binary search + hashing)
64. Substring Equality Queries
65. Longest Palindromic Substring (hash + binary search)
66. Check If Two Substrings Are Equal
67. Count Distinct Substrings
68. Detect Rotation via Hashing
69. Palindrome Check (prefix+suffix hash)
70. Good Splits of String by Hash

8️⃣ Dynamic Programming on Strings — *15 Problems*

DP subsequences, palindromes, partitions

71. Longest Palindromic Substring
72. Palindromic Substrings
73. Longest Palindromic Subsequence
74. Count Palindromic Subsequences
75. Edit Distance
76. Distinct Subsequences
77. Shortest Common Supersequence
78. Interleaving String
79. Minimum Insertions to Form Palindrome
80. Minimum Deletions to Make Strings Equal
81. Regular Expression Matching
82. Wildcard Matching
83. Scramble String
84. Word Break
85. Palindrome Partitioning (DP version)

9️⃣ Greedy String Patterns — *5 Problems*

Lexicographically smallest strings

86. Remove K Digits (monotonic stack)

87. Build Lexicographically Smallest String from freq
88. Rearrange String Without Adjacent Equals
89. Largest Variance Substring
90. Split String into Descending Consecutive Values

🔟 Trie + String Problems — *5 Problems*

91. Implement Trie (Prefix Tree)
92. Longest Common Prefix
93. Search Suggestions System
94. Replace Words (dictionary trie)
95. Word Search II (grid + trie)

1️⃣1️⃣ Backtracking / Recursion String Problems — *7 Problems*

96. Letter Combinations of a Phone Number
97. Generate Parentheses
98. Restore IP Addresses
99. Permutations of String
100. Split String into Fibonacci Sequence
101. Palindrome Partitioning
102. Binary Watch Combinations

(We included 7 here to extend beyond 100 total; choose any 100 subset.)

1️⃣2️⃣ Advanced / Rare Patterns (Optional Add-ons)

(For CP/FAANG hard rounds)

103. Smallest K-periodic String
104. Prefix Automaton (Aho-Corasick prerequisites)
105. Compressed String Matching
106. Suffix Array Construction
107. Suffix Tree Ukkonen Algorithm
108. Searching Patterns with Wildcards
109. Lexicographically Minimal String Rotation (Booth's Algorithm)
110. Binary String DP w/ modular constraints

These are optional for mastery beyond LC.

🎯 Summary Table (100 Problems by Pattern)

Pattern	Count
Sliding Window	15
Two Pointers	10
Hash Map	10
Stack	10
Prefix–Suffix / KMP	10
Z-Algorithm	5
String Hashing	10
DP on Strings	15
Greedy	5
Trie	5
Backtracking	7

✅ Next Step (Highly Recommended)

I can generate any of these:

- 🔹 **Full PDF Cheat-Sheet of 100 String Patterns**
- 🔹 **100 Problems + Pattern-wise Python Templates**
- 🔹 **100 Problems + Solutions (short/optimal/interview-ready)**
- 🔹 **Pattern-by-Pattern practice schedule (30 days)**
- 🔹 **Each problem mapped to specific code templates**

Just tell me:

👉 **"Generate 100 String Problems with Python templates"** or
👉 **"Create PDF version"** or
👉 **"Give 30-day String Mastery Plan"**