**1. Count Vowels in a String**

**Problem:** Count the number of vowels (a, e, i, o, u) in a string using recursion.

**Examples:**

* **Input:** "hello"  
  **Expected Output:** 2 (e, o)
* **Input:** "beautiful"  
  **Expected Output:** 5 (e, a, u, i, u)

**2. Remove Spaces from a String**

**Problem:** Remove all spaces from a given string using recursion.

**Examples:**

* **Input:** "hello world"  
  **Expected Output:** "helloworld"
* **Input:** " a b c d "  
  **Expected Output:** "abcd"

**3. Replace Vowels with '\*'**

**Problem:** Replace all vowels in a string with \* using recursion.

**Examples:**

* **Input:** "apple"  
  **Expected Output:** "\*ppl\*"
* **Input:** "education"  
  **Expected Output:** "\*d\*c\*t\*\*n"

**4. Count Character Occurrences**

**Problem:** Count the number of occurrences of a specific character in a string using recursion.

**Examples:**

* **Input:** "programming", 'g'  
  **Expected Output:** 2
* **Input:** "mississippi", 's'  
  **Expected Output:** 4

**5. Find First Occurrence of a Character**

**Problem:** Find the first occurrence of a character in a string using recursion. Return -1 if not found.

**Examples:**

* **Input:** "recursion", 'r'  
  **Expected Output:** 0 (first 'r' at index 0)
* **Input:** "testing", 'x'  
  **Expected Output:** -1 (not found)

**6. Reverse a String**

**Problem:** Reverse the given string using recursion.

**Examples:**

* **Input:** "hello"  
  **Expected Output:** "olleh"
* **Input:** "world"  
  **Expected Output:** "dlrow"

**7. Remove Duplicates from a String**

**Problem:** Remove duplicate characters from a string using recursion.

**Examples:**

* **Input:** "aabccba"  
  **Expected Output:** "abc"
* **Input:** "mississippi"  
  **Expected Output:** "misp"

**8. Expand Compressed String**

**Problem:** Expand a compressed string like "a3b2c4" into "aaabbcccc".

**Examples:**

* **Input:** "a2b3"  
  **Expected Output:** "aabbb"
* **Input:** "x4y2"  
  **Expected Output:** "xxxxyy"

**9. Check for Anagrams**

**Problem:** Check if two strings are anagrams using recursion.

**Examples:**

* **Input:** "listen", "silent"  
  **Expected Output:** true
* **Input:** "hello", "world"  
  **Expected Output:** false

**10. Capitalize the First Letter of Every Word**

**Problem:** Capitalize the first letter of every word in a string using recursion.

**Examples:**

* **Input:** "hello world"  
  **Expected Output:** "Hello World"
* **Input:** "good morning"  
  **Expected Output:** "Good Morning"

**11. Count Consonants in a String**

**Problem:** Count consonants in a string using recursion.

**Examples:**

* **Input:** "hello" → **Expected Output:** 3 (h, l, l)
* **Input:** "abcde" → **Expected Output:** 3 (b, c, d)

**12. Remove All Digits from a String**

**Problem:** Remove all numeric characters from a string using recursion.

**Examples:**

* **Input:** "a1b2c3" → **Expected Output:** "abc"
* **Input:** "hello123" → **Expected Output:** "hello"

**13. Convert to Uppercase**

**Problem:** Convert all lowercase letters to uppercase using recursion.

**Examples:**

* **Input:** "hello" → **Expected Output:** "HELLO"
* **Input:** "Java" → **Expected Output:** "JAVA"

**14. Convert to Lowercase**

**Problem:** Convert all uppercase letters to lowercase using recursion.

**Examples:**

* **Input:** "HELLO" → **Expected Output:** "hello"
* **Input:** "Java" → **Expected Output:** "java"

**15. Find Maximum Occurring Character**

**Problem:** Find the character that occurs most frequently in a string using recursion.

**Examples:**

* **Input:** "success" → **Expected Output:** 's' (appears 3 times)
* **Input:** "apple" → **Expected Output:** 'p' (appears 2 times)

**16. Remove Non-Alphabetic Characters**

**Problem:** Remove all non-alphabetic characters from a string using recursion.

**Examples:**

* **Input:** "a1b@c#d" → **Expected Output:** "abcd"
* **Input:** "he!llo@wo#rld" → **Expected Output:** "helloworld"

**17. Find Last Occurrence of a Character**

**Problem:** Find the last occurrence of a character in a string using recursion.

**Examples:**

* **Input:** "banana", 'a' → **Expected Output:** 5
* **Input:** "testing", 't' → **Expected Output:** 3

**18. Count Words in a String**

**Problem:** Count the number of words in a string using recursion.

**Examples:**

* **Input:** "hello world" → **Expected Output:** 2
* **Input:** "this is recursion" → **Expected Output:** 3

**19. Print Characters Separately**

**Problem:** Print each character of a string on a new line using recursion.

**Examples:**

* **Input:** "hello"  
  **Expected Output:**

Copy code

h

e

l

l

o

**20. Find All Substrings**

**Problem:** Find all substrings of a string using recursion.

**Examples:**

* **Input:** "abc"  
  **Expected Output:** ["a", "b", "c", "ab", "bc", "abc"]
* **Input:** "xy"  
  **Expected Output:** ["x", "y", "xy"]

**21. Remove Consecutive Duplicates**

**Problem:** Remove consecutive duplicate characters from a string using recursion.

**Examples:**

* **Input:** "aabbcc" → **Expected Output:** "abc"
* **Input:** "xxxyyyzzz" → **Expected Output:** "xyz"

**22. Find the Shortest Word in a Sentence**

**Problem:** Find the shortest word in a sentence using recursion.

**Examples:**

* **Input:** "this is a test" → **Expected Output:** "a"
* **Input:** "short long word" → **Expected Output:** "long"

**23. Remove Specific Character**

**Problem:** Remove all occurrences of a specific character from a string using recursion.

**Examples:**

* **Input:** "hello", 'l' → **Expected Output:** "heo"
* **Input:** "apple", 'p' → **Expected Output:** "ale"

**24. Check If String Contains Only Letters**

**Problem:** Check if a string contains only alphabetic characters using recursion.

**Examples:**

* **Input:** "hello" → **Expected Output:** true
* **Input:** "hello123" → **Expected Output:** false

**25. Replace Multiple Spaces with One**

**Problem:** Replace multiple spaces with a single space using recursion.

**Examples:**

* **Input:** "hello world" → **Expected Output:** "hello world"
* **Input:** " a b " → **Expected Output:** "a b"

**26. Generate All Valid Parentheses Combinations**

**Problem:** Given n, generate all valid combinations of n pairs of parentheses.

**Examples:**

* **Input:** n = 2 → **Expected Output:** ["()()", "(())"]
* **Input:** n = 3 → **Expected Output:** ["((()))", "(()())", "(())()", "()(())", "()()()"]

**Hint:** Use two counters (open, close) to track parentheses placement.

**27. Print All Permutations of a String with Duplicates**

**Problem:** Print all unique permutations of a string with duplicate characters.

**Examples:**

* **Input:** "aab" → **Expected Output:** ["aab", "aba", "baa"]
* **Input:** "abc" → **Expected Output:** ["abc", "acb", "bac", "bca", "cab", "cba"]

**Hint:** Use backtracking with a set to avoid duplicates.

**28. Generate All Subsets with Repeated Characters Allowed**

**Problem:** Generate all possible subsets of a string, including duplicates.

**Examples:**

* **Input:** "abc" → **Expected Output:** ["", "a", "b", "c", "ab", "ac", "bc", "abc"]

**Hint:** Think of including/excluding each character recursively.

**29. String Segmentation**

**Problem:** Check if a string can be segmented into valid words from a dictionary.

**Examples:**

* **Input:** "applepenapple", dict = ["apple", "pen"] → **Expected Output:** true
* **Input:** "catsandog", dict = ["cats", "dog", "sand", "and", "cat"] → **Expected Output:** false

**Hint:** Use recursion to check prefixes and remainders.

**30. Find Longest Common Subsequence (LCS)**

**Problem:** Find the longest common subsequence between two strings.

**Examples:**

* **Input:** "abcde", "ace" → **Expected Output:** 3 (ace)
* **Input:** "abc", "abc" → **Expected Output:** 3 (abc)

**Hint:** Use recursion to compare characters or skip one of them.

**31. Count Palindromic Subsequences**

**Problem:** Count all palindromic subsequences in a string.

**Examples:**

* **Input:** "aaa" → **Expected Output:** 6 (a, a, a, aa, aa, aaa)

**Hint:** Check every substring recursively.

**32. Find Minimum Insertions to Make a String Palindrome**

**Problem:** Find the minimum number of insertions required to make a string a palindrome.

**Examples:**

* **Input:** "abca" → **Expected Output:** 1 (insert 'b' at the end)

**Hint:** Use a two-pointer recursive approach.

**33. Wildcard Matching (\* and ?)**

**Problem:** Implement a wildcard matching function for strings where ? matches one character and \* matches any sequence.

**Examples:**

* **Input:** "adceb", "\*a\*b" → **Expected Output:** true

**Hint:** Use a combination of matching single characters or skipping sequences.

**34. Decode Encoded Messages**

**Problem:** Given an encoded message as digits (A=1, B=2, ... Z=26), count all possible decodings.

**Examples:**

* **Input:** "12" → **Expected Output:** 2 ("AB", "L")

**Hint:** Check single-digit and two-digit combinations.

**35. Interleaving Strings**

**Problem:** Check if a string is an interleaving of two other strings.

**Examples:**

* **Input:** "aab", "axy", "aaxaby" → **Expected Output:** true

**Hint:** Compare characters from both strings recursively.

**36. Generate All Palindromic Partitions**

**Problem:** Partition a string into palindromic substrings.

**Examples:**

* **Input:** "aab" → **Expected Output:** [["a","a","b"], ["aa","b"]]

**Hint:** Use backtracking and a palindrome check.

**37. Word Break with All Possible Combinations**

**Problem:** Print all possible segmentations of a string based on a dictionary.

**Examples:**

* **Input:** "catsanddog", dict=["cats", "cat", "and", "sand", "dog"]  
  **Expected Output:** ["cat sand dog", "cats and dog"]

**Hint:** Similar to word segmentation but requires storing all paths.

**38. Print All Balanced Strings of Length N**

**Problem:** Print all binary strings of length N with equal numbers of 0s and 1s.

**Examples:**

* **Input:** N=4 → **Expected Output:** ["1100", "1010", "1001", "0110", "0101", "0011"]

**Hint:** Use counters for balance checks.

**39. Count Distinct Subsequences**

**Problem:** Count all distinct subsequences of a string.

**Examples:**

* **Input:** "abc" → **Expected Output:** 7 (a, b, c, ab, ac, bc, abc)

**Hint:** Consider including/excluding each character recursively.

### 436. Generate All Valid Parentheses Combinations

**Problem:** Given n, generate all valid combinations of n pairs of parentheses.

**Examples:**

* **Input:** n = 2 → **Expected Output:** ["()()", "(())"]
* **Input:** n = 3 → **Expected Output:** ["((()))", "(()())", "(())()", "()(())", "()()()"]

**Hint:** Use two counters (open, close) to track parentheses placement.

**37. Print All Permutations of a String with Duplicates**

**Problem:** Print all unique permutations of a string with duplicate characters.

**Examples:**

* **Input:** "aab" → **Expected Output:** ["aab", "aba", "baa"]
* **Input:** "abc" → **Expected Output:** ["abc", "acb", "bac", "bca", "cab", "cba"]

**Hint:** Use backtracking with a set to avoid duplicates.

**38. Generate All Subsets with Repeated Characters Allowed**

**Problem:** Generate all possible subsets of a string, including duplicates.

**Examples:**

* **Input:** "abc" → **Expected Output:** ["", "a", "b", "c", "ab", "ac", "bc", "abc"]

**Hint:** Think of including/excluding each character recursively.

**39. String Segmentation**

**Problem:** Check if a string can be segmented into valid words from a dictionary.

**Examples:**

* **Input:** "applepenapple", dict = ["apple", "pen"] → **Expected Output:** true
* **Input:** "catsandog", dict = ["cats", "dog", "sand", "and", "cat"] → **Expected Output:** false

**Hint:** Use recursion to check prefixes and remainders.

**40. Find Longest Common Subsequence (LCS)**

**Problem:** Find the longest common subsequence between two strings.

**Examples:**

* **Input:** "abcde", "ace" → **Expected Output:** 3 (ace)
* **Input:** "abc", "abc" → **Expected Output:** 3 (abc)

**Hint:** Use recursion to compare characters or skip one of them.

**41. Count Palindromic Subsequences**

**Problem:** Count all palindromic subsequences in a string.

**Examples:**

* **Input:** "aaa" → **Expected Output:** 6 (a, a, a, aa, aa, aaa)

**Hint:** Check every substring recursively.

**42. Find Minimum Insertions to Make a String Palindrome**

**Problem:** Find the minimum number of insertions required to make a string a palindrome.

**Examples:**

* **Input:** "abca" → **Expected Output:** 1 (insert 'b' at the end)

**Hint:** Use a two-pointer recursive approach.

**43. Wildcard Matching (\* and ?)**

**Problem:** Implement a wildcard matching function for strings where ? matches one character and \* matches any sequence.

**Examples:**

* **Input:** "adceb", "\*a\*b" → **Expected Output:** true

**Hint:** Use a combination of matching single characters or skipping sequences.

**44. Decode Encoded Messages**

**Problem:** Given an encoded message as digits (A=1, B=2, ... Z=26), count all possible decodings.

**Examples:**

* **Input:** "12" → **Expected Output:** 2 ("AB", "L")

**Hint:** Check single-digit and two-digit combinations.

**45. Interleaving Strings**

**Problem:** Check if a string is an interleaving of two other strings.

**Examples:**

* **Input:** "aab", "axy", "aaxaby" → **Expected Output:** true

**Hint:** Compare characters from both strings recursively.

**46. Generate All Palindromic Partitions**

**Problem:** Partition a string into palindromic substrings.

**Examples:**

* **Input:** "aab" → **Expected Output:** [["a","a","b"], ["aa","b"]]

**Hint:** Use backtracking and a palindrome check.

**47. Word Break with All Possible Combinations**

**Problem:** Print all possible segmentations of a string based on a dictionary.

**Examples:**

* **Input:** "catsanddog", dict=["cats", "cat", "and", "sand", "dog"]  
  **Expected Output:** ["cat sand dog", "cats and dog"]

**Hint:** Similar to word segmentation but requires storing all paths.

**48. Print All Balanced Strings of Length N**

**Problem:** Print all binary strings of length N with equal numbers of 0s and 1s.

**Examples:**

* **Input:** N=4 → **Expected Output:** ["1100", "1010", "1001", "0110", "0101", "0011"]

**Hint:** Use counters for balance checks.

**49. Count Distinct Subsequences**

**Problem:** Count all distinct subsequences of a string.

**Examples:**

* **Input:** "abc" → **Expected Output:** 7 (a, b, c, ab, ac, bc, abc)

**Hint:** Consider including/excluding each character recursively.

**50. Shortest Common Supersequence**

**Problem:** Find the shortest string that contains two strings as subsequences.

**Examples:**

* **Input:** "geek", "eke" → **Expected Output:** "geeke"

**Hint:** Similar to LCS but construct a merged string.

**0. Shortest Common Supersequence**

**Problem:** Find the shortest string that contains two strings as subsequences.

**Examples:**

* **Input:** "geek", "eke" → **Expected Output:** "geeke"

**Hint:** Similar to LCS but construct a merged string.