

Problem:

Write a function that takes a string as input and returns the string reversed.

Example:

Input: `"hello"`

Output: `"olleh"`

Solution:

java

```
public class ReverseString {
    public static String reverse(String s) {
        return new StringBuilder(s).reverse().toString();
    }

    public static void main(String[] args) {
        System.out.println(reverse("hello")); // Output: "olleh"
    }
}
```

2. Check if a String is a Palindrome

Problem:

Determine if a given string is a palindrome (reads the same backward as forward).

Example:

Input: `"racecar"`

Output: `true`

Solution:

java

```
public class PalindromeCheck {
    public static boolean isPalindrome(String s) {
        int left = 0, right = s.length() - 1;
        while (left < right) {
            if (s.charAt(left) != s.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    public static void main(String[] args) {
        System.out.println(isPalindrome("racecar")); // Output: true
    }
}
```

```
}
}
```

3. Count Vowels in a String

Problem:

Count the number of vowels in a given string.

Example:

Input: `"hello world"`

Output: `3`

Solution:

```
java
```

```
public class CountVowels {
    public static int countVowels(String s) {
        int count = 0;
        for (char c : s.toCharArray()) {
            if ("aeiouAEIOU".indexOf(c) != -1) {
                count++;
            }
        }
        return count;
    }

    public static void main(String[] args) {
        System.out.println(countVowels("hello world")); // Output: 3
    }
}
```

4. Find the First Non-Repeating Character

Problem:

Find the first non-repeating character in a string.

Example:

Input: `"leetcode"`

Output: `l`

Solution:

```
java
```

```
import java.util.LinkedHashMap;
import java.util.Map;
```

```
public class FirstNonRepeatingCharacter {
    public static char firstNonRepeating(String s) {
        Map<Character, Integer> charCount = new LinkedHashMap<>();
        for (char c : s.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }

        for (Map.Entry<Character, Integer> entry : charCount.entrySet()) {
            if (entry.getValue() == 1) {
                return entry.getKey();
            }
        }
        return '\0'; // return null character if not found
    }

    public static void main(String[] args) {
        System.out.println(firstNonRepeating("leetcode")); // Output: 'l'
    }
}
```

5. String Compression

Problem:

Implement a method to perform basic string compression using the counts of repeated characters.

Example:

Input: `"aabcccccaaa"`

Output: `"a2b1c5a3"`

Solution:

java

```
public class StringCompression {
    public static String compress(String s) {
        StringBuilder compressed = new StringBuilder();
        int count = 1;

        for (int i = 1; i <= s.length(); i++) {
            if (i < s.length() && s.charAt(i) == s.charAt(i - 1)) {
                count++;
            } else {
                compressed.append(s.charAt(i - 1)).append(count);
                count = 1;
            }
        }

        return compressed.toString();
    }

    public static void main(String[] args) {
        System.out.println(compress("aabcccccaaa")); // Output: "a2b1c5a3"
    }
}
```

6. Check if Two Strings are Anagrams

Problem:

Determine if two strings are anagrams of each other (contain the same characters in a different order).

Example:

Input: `"listen", "silent"`

Output: `true`

Solution:

```
java

import java.util.Arrays;

public class AnagramCheck {
    public static boolean areAnagrams(String s1, String s2) {
        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }

    public static void main(String[] args) {
        System.out.println(areAnagrams("listen", "silent")); // Output: true
    }
}
```

7. Longest Substring Without Repeating Characters

Problem:

Find the length of the longest substring without repeating characters.

Example:

Input: `"abcabcbb"`

Output: `3`

Explanation: The answer is `"abc"`.

Solution:

```
java

import java.util.HashSet;

public class LongestSubstring {
    public static int lengthOfLongestSubstring(String s) {
        HashSet<Character> set = new HashSet<>();
        int left = 0, maxLength = 0;

        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left));
                left++;
            }
            set.add(s.charAt(right));
            maxLength = Math.max(maxLength, right - left + 1);
        }

        return maxLength;
    }
}
```

```

        set.remove(s.charAt(left));
        left++;
    }
    set.add(s.charAt(right));
    maxLength = Math.max(maxLength, right - left + 1);
}

return maxLength;
}

public static void main(String[] args) {
    System.out.println(lengthOfLongestSubstring("abcabcbb")); // Output: 3
}

```

8. Rotate String

Problem:

Check if one string is a rotation of another string.

Example:

Input: `s1 = "waterbottle", s2 = "erbottlewat"`

Output: `true`

Solution:

java

```

public class RotateString {
    public static boolean isRotated(String s1, String s2) {
        return (s1.length() == s2.length()) && (s1 + s1).contains(s2);
    }

    public static void main(String[] args) {
        System.out.println(isRotated("waterbottle", "erbottlewat")); // Output: true
    }
}

```

9. Find the Longest Palindromic Substring

Problem:

Given a string, find the longest palindromic substring.

Example:

Input: `"babad"`

Output: `"bab"` (or `"aba"`)

Solution:

```
java
```

```
public class LongestPalindromicSubstring {
    public static String longestPalindrome(String s) {
        if (s.length() < 1) return "";
        int start = 0, end = 0;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expandFromCenter(s, i, i);    // Odd length
            int len2 = expandFromCenter(s, i, i + 1); // Even length
            int len = Math.max(len1, len2);
            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
        }
        return s.substring(start, end + 1);
    }

    private static int expandFromCenter(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }

    public static void main(String[] args) {
        System.out.println(longestPalindrome("babad")); // Output: "bab" or "aba"
    }
}
```

10. Group Anagrams

Problem:

Given an array of strings, group the anagrams together.

Example:

Input: `["eat", "tea", "tan", "ate", "nat", "bat"]`

Output: `[["bat"], ["nat", "tan"], ["ate", "eat", "tea"]`

Solution:

```
java
```

```
import java.util.*;

public class GroupAnagrams {
    public static List<List<String>> groupAnagrams(String[] strs) {
        Map<String, List<String>> map = new HashMap<>();

        for (String s : strs) {
            char[] arr = s.toCharArray();
            Arrays.sort(arr);
            String sorted = new String(arr);
            map.putIfAbsent(sorted, new ArrayList<>());
            map.get(sorted).add(s);
        }
    }
}
```

```

        return new ArrayList<>(map.values());
    }

    public static void main(String[] args) {
        String[] strs = {"eat", "tea", "tan", "ate", "nat", "bat"};
        System.out.println(groupAnagrams(strs)); // Output: [["bat"], ["nat", "tan"],
["ate", "eat", "tea"]]
    }
}

```

11. Implement strStr()

Problem:

Implement `strStr()`, which locates a substring within a string.

Example:

Input: `haystack = "hello", needle = "ll"`

Output: `2`

Solution:

java

```

public class StrStr {
    public static int strStr(String haystack, String needle) {
        if

```



continue from 11 till 20



Sure! Here are the remaining string interview questions from 11 to 20:

11. Implement strStr()

Problem:

Implement `strStr()`, which locates a substring within a string.

Example:

Input: `haystack = "hello", needle = "ll"`

Output: `2`

Solution:

```
java
```

```
public class StrStr {
    public static int strStr(String haystack, String needle) {
        if (needle.isEmpty()) return 0; // Edge case
        return haystack.indexOf(needle);
    }

    public static void main(String[] args) {
        System.out.println(strStr("hello", "ll")); // Output: 2
    }
}
```

12. Longest Common Prefix

Problem:

Write a function to find the longest common prefix string among an array of strings.

Example:

Input: `["flower", "flow", "flight"]`

Output: `"fl"`

Solution:

```
java
```

```
public class LongestCommonPrefix {
    public static String longestCommonPrefix(String[] strs) {
        if (strs.length == 0) return "";
        String prefix = strs[0];

        for (int i = 1; i < strs.length; i++) {
            while (strs[i].indexOf(prefix) != 0) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) return "";
            }
        }
        return prefix;
    }

    public static void main(String[] args) {
        String[] strs = {"flower", "flow", "flight"};
        System.out.println(longestCommonPrefix(strs)); // Output: "fl"
    }
}
```

13. Count and Say

Problem:

The count-and-say sequence is a sequence of digit strings defined by the following rules:

1. The first term is "1".

2. Each subsequent term is generated by describing the previous term.

Example:

Input: `n = 4`

Output: `"1211"` (the sequence is `1`, `11`, `21`, `1211`)

Solution:

java

```
public class CountAndSay {
    public static String countAndSay(int n) {
        String result = "1";

        for (int i = 1; i < n; i++) {
            StringBuilder sb = new StringBuilder();
            int count = 1;
            for (int j = 0; j < result.length(); j++) {
                if (j + 1 < result.length() && result.charAt(j) == result.charAt(j + 1)) {
                    count++;
                } else {
                    sb.append(count).append(result.charAt(j));
                    count = 1;
                }
            }
            result = sb.toString();
        }

        return result;
    }

    public static void main(String[] args) {
        System.out.println(countAndSay(4)); // Output: "1211"
    }
}
```

14. Valid Parentheses

Problem:

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

Example:

Input: `"()[]{}"`

Output: `true`

Solution:

java

```
import java.util.Stack;

public class ValidParentheses {
    public static boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
```

```

        for (char c : s.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            } else {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if ((c == ')' && top != '(') ||
                    (c == '}' && top != '{') ||
                    (c == ']' && top != '[')) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }

    public static void main(String[] args) {
        System.out.println(isValid("()[]{}")); // Output: true
    }
}

```

15. Substring with Concatenation of All Words

Problem:

You are given a string `s` and an array of strings `words`. Find all starting indices of substring(s) in `s` which is a concatenation of each word in `words` exactly once and without any intervening characters.

Example:

Input: `s = "barfoothefoobarman", words = ["foo","bar"]`

Output: `[0,9]`

Solution:

java

```

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class SubstringConcatenation {
    public static List<Integer> findSubstring(String s, String[] words) {
        List<Integer> result = new ArrayList<>();
        if (s.isEmpty() || words.length == 0) return result;

        int wordLength = words[0].length();
        int totalWordsLength = words.length * wordLength;
        HashMap<String, Integer> wordCount = new HashMap<>();

        for (String word : words) {
            wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }

        for (int i = 0; i <= s.length() - totalWordsLength; i++) {
            HashMap<String, Integer> seenWords = new HashMap<>();
            int j = 0;

            while (j < words.length) {
                String word = s.substring(i + j * wordLength, i + (j + 1) * wordLength);
                if (wordCount.containsKey(word)) {
                    seenWords.put(word, seenWords.getOrDefault(word, 0) + 1);
                    if (seenWords.get(word) > wordCount.get(word)) break;
                }
                j++;
            }

            if (j == words.length) result.add(i);
        }
    }
}

```

```

        } else {
            break;
        }
        j++;
    }

    if (j == words.length) {
        result.add(i);
    }
}

return result;
}

public static void main(String[] args) {
    String s = "barfoothefoobarman";
    String[] words = {"foo", "bar"};
    System.out.println(findSubstring(s, words)); // Output: [0, 9]
}
}

```

16. Replace All Spaces with %20

Problem:

Given a string, replace all spaces with `%20`.

Example:

Input: `"Mr John Smith ", 13`

Output: `"Mr%20John%20Smith"`

Solution:

java

```

public class ReplaceSpaces {
    public static String replaceSpaces(String s, int length) {
        char[] chars = s.toCharArray();
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < length; i++) {
            if (chars[i] == ' ') {
                result.append("%20");
            } else {
                result.append(chars[i]);
            }
        }

        return result.toString();
    }

    public static void main(String[] args) {
        String s = "Mr John Smith ";
        System.out.println(replaceSpaces(s, 13)); // Output: "Mr%20John%20Smith"
    }
}

```

17. Minimum Window Substring

Problem:

Given two strings `s` and `t`, return the minimum window substring of `s` such that every character in `t` (including duplicates) is included in the window.

Example:

Input: `s` = "ADOBECODEBANC", `t` = "ABC"

Output: `"BANC"`

Solution:

java

```
import java.util.HashMap;

public class MinimumWindowSubstring {
    public static String minWindow(String s, String t) {
        if (t.length() > s.length()) return "";

        HashMap<Character, Integer> tCount = new HashMap<>();
        for (char c : t.toCharArray()) {
            tCount.put(c, tCount.getOrDefault(c, 0) + 1);
        }

        int required = tCount.size();
        int left = 0, right = 0, formed = 0;
        HashMap<Character, Integer> windowCount = new HashMap<>();
        int[] ans = {-1, 0, 0}; // {length, left, right}

        while (right < s.length()) {
            char c = s.charAt(right);
            windowCount.put(c, windowCount.getOrDefault(c, 0) + 1);

            if (tCount.containsKey(c) && windowCount.get(c).intValue() ==
                tCount.get(c).intValue()) {
                formed++;
            }

            while (left <= right && formed == required) {
                c = s.charAt(left);
                if (ans[0] == -1 || right - left + 1 < ans[0]) {
                    ans[0] = right - left + 1;
                    ans[1] = left;
                    ans[2] = right;
                }
                windowCount.put(c, windowCount.get(c) - 1);
                if (tCount.containsKey(c) && windowCount.get(c).intValue() <
                    tCount.get(c).intValue()) {
                    formed--;
                }
                left++;
            }
            right++;
        }
        return ans[0] == -1 ? "" : s.substring(ans[1], ans[2] + 1);
    }

    public static void main(String[] args) {
        System.out.println(minWindow("ADOBECODEBANC", "ABC")); // Output: "BANC"
    }
}
```

18. Longest Repeating Character Replacement

Problem:

Given a string `s` and an integer `k`, find the length of the longest substring that can be obtained by replacing at most `k` characters.

Example:

Input: `s` = "AABABBA", `k` = 1`

Output: `4` (The longest substring is "ABAB")

Solution:

java

```
public class LongestRepeatingCharacterReplacement {
    public static int characterReplacement(String s, int k) {
        int[] count = new int[26];
        int maxCount = 0, left = 0, maxLength = 0;

        for (int right = 0; right < s.length(); right++) {
            maxCount = Math.max(maxCount, ++count[s.charAt(right) - 'A']);

            while (right - left + 1 - maxCount > k) {
                count[s.charAt(left) - 'A']--;
                left++;
            }
            maxLength = Math.max(maxLength, right - left + 1);
        }

        return maxLength;
    }

    public static void main(String[] args) {
        System.out.println(characterReplacement("AABABBA", 1)); // Output: 4
    }
}
```

19. Find All Anagrams in a String

Problem:

Given a string `s` and a non-empty string `p`, find all the start indices of `p`'s anagrams in `s`.

Example:

Input: `s` = "cbaebabacd", `p` = "abc"`

Output: `[0, 6]`

Solution:

```
java
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class FindAllAnagrams {
    public static List<Integer> findAnagrams(String s, String p) {
        List<Integer> result = new ArrayList<>();
        if (s.length() < p.length()) return result;

        HashMap<Character, Integer> pCount = new HashMap<>();
        HashMap<Character, Integer> sCount = new HashMap<>();

        for (char c : p.toCharArray()) {
            pCount.put(c, pCount.getOrDefault(c, 0) + 1);
        }

        for (int i = 0; i < p.length(); i++) {
            sCount.put(s.charAt(i), sCount.getOrDefault(s.charAt(i), 0) + 1);
        }

        if (sCount.equals(pCount)) result.add(0);

        for (int i = p.length(); i < s.length(); i++) {
            char newChar = s.charAt(i);
            char oldChar = s.charAt(i - p.length());
            sCount.put(newChar, sCount.getOrDefault(newChar, 0) + 1);
            sCount.put(oldChar, sCount.get(oldChar) - 1);
            if (sCount.get(oldChar) == 0) {
                sCount.remove(oldChar);
            }

            if (sCount.equals(pCount)) {
                result.add(i - p.length() + 1);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        System.out.println(findAnagrams("cbaebabacd", "abc")); // Output: [0, 6]
    }
}
```

20. Minimum Number of Steps to Make Two Strings Anagram

Problem:

Given two strings `s` and `t`, return the minimum number of steps required to make `s` and `t` anagrams of each other.

Example:

Input: `s` = "bab", `t` = "aba"

Output: `1` (Remove one 'b' from `s`)

Solution:

```
java
```

```
import java.util.HashMap;

public class MinStepsToAnagram {
    public static int minSteps(String s, String t) {
        int[] count = new int[26];
        for (char c : s.toCharArray()) count[c - 'a']++;
        for (char c : t.toCharArray()) count[c - 'a']--;

        int steps = 0;
        for (int c : count) {
            if (c > 0) steps += c; // Count extra characters in `s`
        }
        return steps;
    }

    public static void main(String[] args) {
        System.out.println(minSteps("bab", "aba")); // Output: 1
    }
}
```

Feel free to ask if you need further explanations or have any other requests!



add 5 more most asked question



Here are 5 more commonly asked string interview questions:

21. Group Anagrams

Problem:

Given an array of strings, group the anagrams together.

Example:

Input: `["eat", "tea", "tan", "ate", "nat", "bat"]`

Output: `[["bat"], ["nat", "tan"], ["ate", "eat", "tea"]`

Solution:

```
java
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class GroupAnagrams {
    public static List<List<String>> groupAnagrams(String[] strs) {
        HashMap<String, List<String>> map = new HashMap<>();

        for (String str : strs) {
```

```

        char[] chars = str.toCharArray();
        java.util.Arrays.sort(chars);
        String key = new String(chars);
        map.putIfAbsent(key, new ArrayList<>());
        map.get(key).add(str);
    }

    return new ArrayList<>(map.values());
}

public static void main(String[] args) {
    String[] strs = {"eat", "tea", "tan", "ate", "nat", "bat"};
    System.out.println(groupAnagrams(strs)); // Output: [["bat"], ["nat", "tan"],
["ate", "eat", "tea"]]
}
}

```

22. Reverse String

Problem:

Write a function that reverses a string.

Example:

Input: `"hello"`

Output: `"olleh"`

Solution:

java

```

public class ReverseString {
    public static String reverseString(String s) {
        StringBuilder reversed = new StringBuilder(s);
        return reversed.reverse().toString();
    }

    public static void main(String[] args) {
        System.out.println(reverseString("hello")); // Output: "olleh"
    }
}

```

23. Check Permutation

Problem:

Given two strings, write a method to decide if one is a permutation of the other.

Example:

Input: `"abc", "bca"`

Output: `true`

Solution:

```
java
```

```
import java.util.Arrays;

public class CheckPermutation {
    public static boolean isPermutation(String s1, String s2) {
        if (s1.length() != s2.length()) return false;
        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }

    public static void main(String[] args) {
        System.out.println(isPermutation("abc", "bca")); // Output: true
    }
}
```

24. Longest Palindromic Substring

Problem:

Given a string `s`, return the longest palindromic substring in `s`.

Example:

Input: `"babad"`

Output: `"bab"` or `"aba"` (both are valid)

Solution:

```
java
```

```
public class LongestPalindromicSubstring {
    public static String longestPalindrome(String s) {
        if (s.length() < 1) return "";
        int start = 0, end = 0;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expandFromCenter(s, i, i); // Odd length
            int len2 = expandFromCenter(s, i, i + 1); // Even length
            int len = Math.max(len1, len2);
            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
        }
        return s.substring(start, end + 1);
    }

    private static int expandFromCenter(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }
}
```

```
public static void main(String[] args) {
    System.out.println(longestPalindrome("babad")); // Output: "bab" or "aba"
}
}
```

25. Valid Anagram

Problem:

Given two strings `s` and `t`, return true if `t` is an anagram of `s`.

Example:

Input: `s` = "anagram", `t` = "nagaram"

Output: `true`

Solution:

java

```
import java.util.HashMap;

public class ValidAnagram {
    public static boolean isAnagram(String s, String t) {
        if (s.length() != t.length()) return false;
        HashMap<Character, Integer> count = new HashMap<>();

        for (char c : s.toCharArray()) {
            count.put(c, count.getOrDefault(c, 0) + 1);
        }

        for (char c : t.toCharArray()) {
            if (!count.containsKey(c) || count.get(c) == 0) return false;
            count.put(c, count.get(c) - 1);
        }

        return true;
    }

    public static void main(String[] args) {
        System.out.println(isAnagram("anagram", "nagaram")); // Output: true
    }
}
```

Feel free to ask if you need more questions or further assistance!



5 more



Here are 5 additional commonly asked string interview questions:

26. Count and Say

Problem:

The count-and-say sequence is a sequence of digit strings defined by the recursive formula:

- `countAndSay(1) = "1"`
- To generate the next term, count the number of digits in the previous term, then say the digits.

Example:

Input: `n = 4`

Output: `"1211"` (The sequence is: "1", "11", "21", "1211")

Solution:

java

```
public class CountAndSay {
    public static String countAndSay(int n) {
        String result = "1";
        for (int i = 1; i < n; i++) {
            StringBuilder next = new StringBuilder();
            char currentChar = result.charAt(0);
            int count = 1;

            for (int j = 1; j < result.length(); j++) {
                if (result.charAt(j) == currentChar) {
                    count++;
                } else {
                    next.append(count).append(currentChar);
                    currentChar = result.charAt(j);
                    count = 1;
                }
            }
            next.append(count).append(currentChar);
            result = next.toString();
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println(countAndSay(4)); // Output: "1211"
    }
}
```

27. Longest Common Prefix

Problem:

Write a function to find the longest common prefix string amongst an array of strings.

Example:

Input: `["flower", "flow", "flight"]`

Output: `"fl"`

Solution:

```
java
```

```
public class LongestCommonPrefix {
    public static String longestCommonPrefix(String[] strs) {
        if (strs.length == 0) return "";
        String prefix = strs[0];

        for (int i = 1; i < strs.length; i++) {
            while (strs[i].indexOf(prefix) != 0) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) return "";
            }
        }
        return prefix;
    }

    public static void main(String[] args) {
        String[] strs = {"flower", "flow", "flight"};
        System.out.println(longestCommonPrefix(strs)); // Output: "fl"
    }
}
```

28. String to Integer (atoi)

Problem:

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer.

Example:

Input: `" -42"`

Output: `-42`

Solution:

```
java
```

```
public class StringToInteger {
    public static int myAtoi(String s) {
        int index = 0, sign = 1, total = 0;
        int n = s.length();

        // Discard all leading whitespaces
        while (index < n && s.charAt(index) == ' ') index++;

        // Handle optional sign
        if (index < n && (s.charAt(index) == '-' || s.charAt(index) == '+')) {
            sign = s.charAt(index) == '-' ? -1 : 1;
            index++;
        }

        // Convert number and avoid overflow
        while (index < n && Character.isDigit(s.charAt(index))) {
            int digit = s.charAt(index) - '0';
            if (total > (Integer.MAX_VALUE - digit) / 10) {
                return sign == 1 ? Integer.MAX_VALUE : Integer.MIN_VALUE;
            }
            total = total * 10 + digit;
        }
    }
}
```

```

        index++;
    }

    return total * sign;
}

public static void main(String[] args) {
    System.out.println(myAtoi("  -42")); // Output: -42
}

```

29. Find the First Unique Character in a String

Problem:

Given a string, find the first non-repeating character in it and return its index. If it doesn't exist, return `-1`.

Example:

Input: `"leetcode"`

Output: `0`

Solution:

java

```

import java.util.HashMap;

public class FirstUniqueCharacter {
    public static int firstUniqChar(String s) {
        HashMap<Character, Integer> count = new HashMap<>();

        for (char c : s.toCharArray()) {
            count.put(c, count.getOrDefault(c, 0) + 1);
        }

        for (int i = 0; i < s.length(); i++) {
            if (count.get(s.charAt(i)) == 1) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        System.out.println(firstUniqChar("leetcode")); // Output: 0
    }
}

```

30. Minimum Window Substring

Problem:

Given two strings `s` and `t`, return the minimum window substring of `s` such that every character in `t` (including duplicates) is included in the window.

Example:

Input: `s = "ADOBECODEBANC", t = "ABC"`

Output: `"BANC"`

Solution:

java

```
import java.util.HashMap;

public class MinimumWindowSubstring {
    public static String minWindow(String s, String t) {
        if (t.length() > s.length()) return "";

        HashMap<Character, Integer> tCount = new HashMap<>();
        for (char c : t.toCharArray()) {
            tCount.put(c, tCount.getOrDefault(c, 0) + 1);
        }

        int left = 0, right = 0, minLength = Integer.MAX_VALUE, minLeft = 0, required =
tCount.size();
        HashMap<Character, Integer> windowCount = new HashMap<>();

        while (right < s.length()) {
            char c = s.charAt(right);
            windowCount.put(c, windowCount.getOrDefault(c, 0) + 1);
            if (tCount.containsKey(c) && windowCount.get(c).intValue() ==
tCount.get(c).intValue()) {
                required--;
            }
            while (required == 0) {
                if (right - left + 1 < minLength) {
                    minLength = right - left + 1;
                    minLeft = left;
                }
                windowCount.put(s.charAt(left), windowCount.get(s.charAt(left)) - 1);
                if (tCount.containsKey(s.charAt(left)) &&
windowCount.get(s.charAt(left)).intValue() < tCount.get(s.charAt(left)).intValue()) {
                    required++;
                }
                left++;
            }
            right++;
        }

        return minLength == Integer.MAX_VALUE ? "" : s.substring(minLeft, minLeft +
minLength);
    }

    public static void main(String[] args) {
        System.out.println(minWindow("ADOBECODEBANC", "ABC")); // Output: "BANC"
    }
}
```

Feel free to ask for more questions or assistance with anything else!

ChatGPT can make mistakes. Check important info.