<https://leetcode.com/problems/string-without-aaa-or-bbb/description/>

Given two integers a and b, return any string s such that:

s has length a + b and contains exactly a 'a' letters, and exactly b 'b' letters,

The substring 'aaa' does not occur in s, and

The substring 'bbb' does not occur in s.

**Example 1:**

**Input:** a = 1, b = 2

**Output:** "abb"

**Explanation:** "abb", "bab" and "bba" are all correct answers.

**Example 2:**

**Input:** a = 4, b = 1

**Output:** "aabaa"

**Constraints:**

0 <= a, b <= 100

It is guaranteed such an s exists for the given a and b.

**Attempt 1: 2024-02-05**

**Solution 1: Greedy + Heap (10 min, exactly same as**[L1405.Longest Happy String (Ref.L1642,L984)](note://WEB763090169ff4f39cc98fd2e2a07dd80a)**)**

class Solution {

    public String strWithout3a3b(int a, int b) {

        PriorityQueue<Node> maxPQ = new PriorityQueue<>((x, y) -> y.count - x.count);

        if(a > 0) {

            maxPQ.offer(new Node('a', a));

        }

        if(b > 0) {

            maxPQ.offer(new Node('b', b));

        }

        StringBuilder sb = new StringBuilder();

        while(!maxPQ.isEmpty()) {

            Node first = maxPQ.poll();

            if(sb.length() >= 2 && sb.charAt(sb.length() - 1) == first.ch

                && sb.charAt(sb.length() - 2) == first.ch) {

                // No need check since it is guaranteed such an s exists

                // for the given a and b.

                //if(maxPQ.isEmpty()) {

                //    break;

                //}

                Node second = maxPQ.poll();

                sb.append(second.ch);

                second.count--;

                if(second.count > 0) {

                    maxPQ.offer(second);

                }

                maxPQ.offer(first);

            } else {

                sb.append(first.ch);

                first.count--;

                if(first.count > 0) {

                    maxPQ.offer(first);

                }

            }

        }

        return sb.toString();

    }

}

class Node {

    char ch;

    int count;

    public Node(char ch, int count) {

        this.ch = ch;

        this.count = count;

    }

}

Time Complexity: O(a + b)

Space Complexity: O(n)

**Solution 2: Greedy (No Heap required)**

class Solution {

    public String strWithout3a3b(int a, int b) {

        StringBuilder sb = new StringBuilder();

        while(a > 0 || b > 0) {

            boolean writeA = false;

            if(sb.length() >= 2 && sb.charAt(sb.length() - 1) == sb.charAt(sb.length() - 2)) {

                if(sb.charAt(sb.length() - 1) == 'b') {

                    writeA = true;

                }

            } else {

                if(a >= b) {

                    writeA = true;

                }

            }

            if(writeA) {

                a--;

                sb.append('a');

            } else {

                b--;

                sb.append('b');

            }

        }

        return sb.toString();

    }

}

Time Complexity: O(a + b)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/string-without-aaa-or-bbb/editorial/>

**Approach 1: Greedy**

**Intuition**

Intuitively, we should write the most common letter first. For example, if we have A = 6, B = 2, we want to write 'aabaabaa'. The only time we don't write the most common letter is if the last two letters we have written are also the most common letter

**Algorithm**

Let's maintain A, B: the number of 'a' and 'b''s left to write.

**If we have already written the most common letter twice, we'll write the other letter. Otherwise, we'll write the most common letter.**

class Solution {

public String strWithout3a3b(int A, int B) {

StringBuilder ans = new StringBuilder();

while (A > 0 || B > 0) {

boolean writeA = false;

int L = ans.length();

if (L >= 2 && ans.charAt(L-1) == ans.charAt(L-2)) {

if (ans.charAt(L-1) == 'b')

writeA = true;

} else {

if (A >= B)

writeA = true;

}

if (writeA) {

A--;

ans.append('a');

} else {

B--;

ans.append('b');

}

}

return ans.toString();

}

}

**Complexity Analysis**

Time Complexity: *O*(*A*+*B*).

Space Complexity: *O*(*A*+*B*).

**Refer to Deepseek**

LeetCode 984, "String Without AAA or BBB," is similar to LeetCode 1405 ("Longest Happy String"). The problem requires constructing a string using the characters 'A' and 'B' such that no three identical characters are consecutive. The input provides the counts of 'A' and 'B', and we need to build the longest possible string that satisfies the condition.

Here’s the Java solution using a greedy approach:

**Java Solution:**

class Solution {

public String strWithout3a3b(int a, int b) {

StringBuilder result = new StringBuilder();

// Continue until we have characters left

while (a > 0 || b > 0) {

// Determine which character to append next

boolean appendA = false;

// If 'A' has a higher count and the last two characters are not 'A', append 'A'

if (a > b) {

if (result.length() >= 2 &&

result.charAt(result.length() - 1) == 'A' &&

result.charAt(result.length() - 2) == 'A') {

appendA = false; // Cannot append 'A' again

} else {

appendA = true;

}

}

// If 'B' has a higher count and the last two characters are not 'B', append 'B'

else if (b > a) {

if (result.length() >= 2 &&

result.charAt(result.length() - 1) == 'B' &&

result.charAt(result.length() - 2) == 'B') {

appendA = true; // Cannot append 'B' again, so append 'A'

} else {

appendA = false;

}

}

// If counts are equal, append the character that is not the same as the last two

else {

if (result.length() >= 2 &&

result.charAt(result.length() - 1) == 'A' &&

result.charAt(result.length() - 2) == 'A') {

appendA = false; // Append 'B'

} else {

appendA = true; // Append 'A'

}

}

// Append the chosen character

if (appendA) {

result.append('A');

a--;

} else {

result.append('B');

b--;

}

}

return result.toString();

}

}

**Explanation:**

Greedy Approach:

At each step, we decide whether to append 'A' or 'B' based on the remaining counts and the last two characters in the result string.

We prioritize the character with the higher remaining count, but we ensure that no three identical characters are consecutive.

Key Conditions:

If a > b, we try to append 'A', but only if the last two characters are not already 'A'.

If b > a, we try to append 'B', but only if the last two characters are not already 'B'.

If a == b, we alternate between 'A' and 'B' to avoid three consecutive identical characters.

Termination:

The loop continues until we have exhausted both 'A' and 'B'.

**Example Walkthrough:**

**Input:**

a = 4, b = 1

**Execution:**

Append 'A' (result = "A", a = 3, b = 1).

Append 'A' (result = "AA", a = 2, b = 1).

Cannot append 'A' again (would result in "AAA"), so append 'B' (result = "AAB", a = 2, b = 0).

Append 'A' (result = "AABA", a = 1, b = 0).

Append 'A' (result = "AABAA", a = 0, b = 0).

**Output:**

"AABAA"

**Time Complexity:**

Loop Iterations:

The loop runs until all characters ('A' and 'B') are exhausted, so it runs ( O(a + b) ) times.

Operations Inside the Loop:

Each iteration involves checking the last two characters of the result string (constant time ( O(1) )) and appending a character (constant time ( O(1) )).

Overall Time Complexity:

**Space Complexity:**

Result String:

The StringBuilder stores the result string, which can have up to ( a + b ) characters. Thus, the space required is ( O(a + b) ).

Overall Space Complexity:

**Why Does This Work?**

The greedy approach ensures that:

We always prioritize the character with the higher remaining count, which helps maximize the length of the string.

We avoid three consecutive identical characters by checking the last two characters before appending.

The solution is optimal because it balances the usage of 'A' and 'B' while adhering to the constraints.

This solution is efficient and works well for the given constraints.

**My solution is more elegant ?**

Yes, the solution you provided is indeed more elegant and concise! It follows the same \*\*greedy approach\*\* as the previous solution but simplifies the logic by focusing on the key conditions for appending 'a' or 'b'. Let’s break it down and analyze it in detail.

**Elegant Solution:**

class Solution {

public String strWithout3a3b(int A, int B) {

StringBuilder ans = new StringBuilder();

while (A > 0 || B > 0) {

boolean writeA = false;

int L = ans.length();

// Check if the last two characters are the same

if (L >= 2 && ans.charAt(L-1) == ans.charAt(L-2)) {

// If the last two characters are 'b', append 'a'

if (ans.charAt(L-1) == 'b')

writeA = true;

} else {

// Otherwise, append the character with the higher count

if (A >= B)

writeA = true;

}

// Append the chosen character

if (writeA) {

A--;

ans.append('a');

} else {

B--;

ans.append('b');

}

}

return ans.toString();

}

}

**Explanation:**

Key Idea:

The solution uses a \*\*greedy approach\*\* to decide whether to append 'a' or 'b' at each step.

It ensures that no three identical characters are consecutive by checking the last two characters in the result string.

Logic:

If the last two characters are the same:

If they are 'b', append 'a' to avoid three consecutive 'b's.

If they are 'a', append 'b' to avoid three consecutive 'a's.

If the last two characters are not the same:

Append the character with the higher remaining count ('a' if A >= B, otherwise 'b').

Termination:

The loop continues until both A and B are exhausted.

**Example Walkthrough:**

**Input:**

A = 4, B = 1

**Execution:**

Append 'a' (result = "a", A = 3, B = 1).

Append 'a' (result = "aa", A = 2, B = 1).

Last two characters are 'a', so append 'b' (result = "aab", A = 2, B = 0).

Append 'a' (result = "aaba", A = 1, B = 0).

Append 'a' (result = "aabaa", A = 0, B = 0).

**Output:**

"aabaa"

**Time Complexity:**

Loop Iterations:

The loop runs until all characters ('a' and 'b') are exhausted, so it runs ( O(A + B) ) times.

Operations Inside the Loop:

Each iteration involves checking the last two characters of the result string (constant time ( O(1) )) and appending a character (constant time ( O(1) )).

Overall Time Complexity:

**Space Complexity:**

Result String:

The StringBuilder stores the result string, which can have up to ( A + B ) characters. Thus, the space required is ( O(A + B) ).

Overall Space Complexity:

**Why Is This Solution Elegant?**

Simplified Logic:

The solution avoids redundant checks and focuses on the key conditions for appending 'a' or 'b'.

It uses a single if-else structure to handle all cases, making the code concise and easy to understand.

Efficient Greedy Strategy:

By always appending the character with the higher remaining count (unless it would violate the "no three consecutive" rule), the solution maximizes the length of the string.

Readable and Maintainable:

The code is clean and self-explanatory, making it easy to maintain and extend if needed.

**Comparison with the Previous Solution:**

|  |  |  |
| --- | --- | --- |
| Feature | Previous Solution | Elegant Solution |
| **Logic** | Explicitly checks for  'A'  and  'B' | Generalizes logic for  'a'  and  'b' |
| **Conditions** | Handles  a > b  ,  b > a  , and  a == b  separately | Uses a single condition for all cases |
| **Code Length** | Longer and more verbose | Shorter and more concise |
| **Readability** | Slightly harder to follow | Easier to understand |

**Conclusion:**

The elegant solution you provided is a great example of how to simplify code while maintaining efficiency and correctness. It uses a \*\*greedy approach\*\* effectively and avoids unnecessary complexity, making it a better choice for solving this problem.

**Refer to**

[L1405.Longest Happy String (Ref.L1642,L984)](note://WEB763090169ff4f39cc98fd2e2a07dd80a)

[L1642.Furthest Building You Can Reach (Ref.L1405)](note://WEB63f41eede3d2b75fae17e8adc3dd4d8e)