# Longest Happy String

| | |
|---|---|
| solved by | Senan |
| Platform | LeetCode |
| difficulty | Medium |
| # Serial | 1405 |
| tags | Heaps String Manipulation |
| language | C++ |
| solved on | @16/10/2024 |
| link | https://leetcode.com/problems/longest-happy-string/description/ |
| Completion | ☑ |

## Intuition

The problem is to generate the longest possible string with at most two consecutive occurrences of any character ('a', 'b', or 'c') while respecting the number of available characters for each.

The greedy approach is optimal here: we always want to add the most frequent character that does not violate the two-consecutive-character restriction. Using a max-heap (or priority queue), we can efficiently retrieve the most frequent character.

## Approach

1. Use a max-heap (priority queue) to keep track of the available characters ('a', 'b', 'c') and their respective counts.
2. While the heap is not empty, pop the character with the largest count. If adding this character would result in three consecutive identical characters, choose the second most frequent character from the heap instead.
3. After using a character, decrement its count and push it back into the heap if there are still occurrences left.
4. Continue this process until no more characters can be added without violating the restriction.

## Complexity

### Time Complexity:

- Inserting and removing elements from the heap takes $\mathrm{O}(\log k)$, where $k = 3$ since there are only three types of characters ('a', 'b', 'c'). Thus, the time complexity is $\mathrm{O}(n \log 3) = \mathrm{O}(n)$, where $n$ is the total number of characters (i.e., $a + b + c$).

### Space Complexity:

- The space complexity is $\mathrm{O}(n)$, from the result string, since we are only using a fixed-size heap of at most three elements and a few variables for counting.

## Code

```
class Solution {
public:
    string longestDiverseString(int a, int b, int c) {
        string result;
        priority_queue<pair<int, char>> maxHeap;

        if (a > 0)
            maxHeap.push({a, 'a'});
        if (b > 0)
            maxHeap.push({b, 'b'});
        if (c > 0)
            maxHeap.push({c, 'c'});

        while (!maxHeap.empty()) {
            pair<int, char> first = maxHeap.top();
            maxHeap.pop();
            int count1 = first.first;
            char char1 = first.second;

            if (result.size() >= 2 && result.back() == char1 && result[result.size() -
2] == char1) {
                if (maxHeap.empty())
                    break;

                pair<int, char> second = maxHeap.top();
                maxHeap.pop();
                int count2 = second.first;
                char char2 = second.second;

                result += char2;
                if (--count2 > 0)
                    maxHeap.push({count2, char2});
                maxHeap.push({count1, char1});
            } else {
                result += char1;
                if (--count1 > 0)
                    maxHeap.push({count1, char1});
            }
        }

        return result;
    }
};
```