Daily Coding Problem   Blog

Daily Coding Problem #231

Problem

This problem was asked by IBM.

Given a string with repeated characters, rearrange the string so that no two adjacent characters are the same. If this is not possible, return None.

For example, given "aaabbc", you could return "ababac". Given "aaab", return None.

Solution

Intuitively, the following greedy algorithm may come to mind: keep popping the most frequent character from the string (that is different from the last one) and adding it to a new list, until there are no more letters left. If we have used all the letters, return the joined list; otherwise, return None.

One tricky part is that we must make sure to pop each new letter before adding back the last one, to avoid repeats.

Here is how we could implement this.

```python
from collections import defaultdict

def rearrange(string):
    frequencies = defaultdict(int)
    for letter in string:

        frequencies[letter] += 1
```

```python
        char, count = max(frequencies.items(), key=lambda x: x[1])
        frequencies.pop(char)
        result = [char]

        while frequencies:
            last_char, last_count = char, count

            char, count = max(frequencies.items(), key=lambda x: x[1])
            frequencies.pop(char)
            result.append(char)

            if last_count > 1:
                frequencies[last_char] = last_count - 1

        if len(result) == len(string):
            return "".join(result)
        else:
            return None
```

This will take $O(N^2)$, where N is the number of letters in the string. This is because finding the maximum value in the dictionary is $O(N)$, and we must do this every time we locate the next letter to add to our result.

We can improve this by using a better data structure. In particular, a heap will allow us to pop and push the most frequent letter in $O(\log N)$, reducing our time complexity to $O(N * \log N)$.

Similar to before, our algorithm will be divided into two steps. First we insert the counts of each character into a heap. Then, we pop the most frequent element one at a time and add it to the result. Once this element is popped, we reinsert the previous element, decrementing its count by one.

```python
from collections import defaultdict
import heapq

def rearrange(string):
    frequencies = defaultdict(int)
    for letter in string:
        frequencies[letter] += 1

    heap = []
```

```python
    for char, count in frequencies.items():
        heapq.heappush(heap, (-count, char))

    count, char = heapq.heappop(heap)
    result = [char]

    while heap:
        last = (count + 1, char)
        count, char = heapq.heappop(heap)
        result.append(char)

        if last[0] < 0:
            heapq.heappush(heap, last)

    if len(result) == len(string):
        return "".join(result)
    else:
        return None
```