Daily Coding Problem Blog

Daily Coding Problem #13

Problem

This problem was asked by Amazon.

Given an integer k and a string s, find the length of the longest substring that contains at most k distinct characters.

For example, given s = "abcba" and k = 2, the longest substring with k distinct characters is "bcb".

Solution

The most obvious brute force solution here is to simply try every possible substring of the string and check whether it contains at most k distinct characters. If it does and it is greater than the current longest valid substring, then update the current one. This takes O(n^2 * k) time, since we use n^2 to generate each possible substring, and then take k to check each character.

```
def longest_substring_with_k_distinct_characters(s, k):
    current_longest_substring = ''
    for i in range(len(s)):
        for j in range(i + 1, len(s) + 1):
            substring = s[i:j]
            if len(set(substring)) <= k and len(substring) >
len(current_longest_substring):
                current_longest_substring = substring
    return len(current_longest_substring)
```

We can improve this by instead keeping a running window of our longest substring. We'll keep a dictionary that maps characters to the index of their last occurrence. Then, as we iterate over the string, we'll check the size of the dictionary. If it's larger than k, then it means our window is too big, so we have to pop the smallest item in the dictionary and recompute the bounds. If, when we add a character to the dictionary and it doesn't go over k, then we're safe -- the dictionary hasn't been filled up yet or it's a character we've seen before.

```
def longest_substring_with_k_distinct_characters(s, k):
    if k == 0:
        return 0

    # Keep a running window
    bounds = (0, 0)
    h = {}
    max_length = 0
    for i, char in enumerate(s):
        h[char] = i
        if len(h) <= k:
            new_lower_bound = bounds[0] # lower bound remains the same
        else:
            # otherwise, pop last occurring char
            key_to_pop = min(h, key=h.get)
            new_lower_bound = h.pop(key_to_pop) + 1

        bounds = (new_lower_bound, bounds[1] + 1)
        max_length = max(max_length, bounds[1] - bounds[0])

    return max_length
```

This takes O(n * k) time and O(k) space.