1.1 Implement an algorithm to determine if a string has all unique characters. What if you can not use additional data structures?

pg 48

SOLUTION

For simplicity, assume char set is ASCII (if not, we need to increase the storage size. The rest of the logic would be the same). *NOTE*: This is a *great* thing to point out to your interviewer!

```
1   public static boolean isUniqueChars2(String str) {
2       boolean[] char_set = new boolean[256];
3       for (int i = 0; i < str.length(); i++) {
4           int val = str.charAt(i);
5           if (char_set[val]) return false;
6           char_set[val] = true;
7       }
8       return true;
9   }
```

Time complexity is O(n), where n is the length of the string, and space complexity is O(n).

We can reduce our space usage a little bit by using a bit vector. We will assume, in the below code, that the string is only lower case 'a' through 'z'. This will allow us to use just a single int

```
1   public static boolean isUniqueChars(String str) {
2       int checker = 0;
3       for (int i = 0; i < str.length(); ++i) {
4           int val = str.charAt(i) - 'a';
5           if ((checker & (1 << val)) > 0) return false;
6           checker |= (1 << val);
7       }
8       return true;
9   }
```

Alternatively, we could do the following:

1. Check every char of the string with every other char of the string for duplicate occurrences. This will take O(n^2) time and no space.
2. If we are allowed to destroy the input string, we could sort the string in O(n log n) time and then linearly check the string for neighboring characters that are identical. Careful, though - many sorting algorithms take up extra space.