Custom Sort String

`S` and `T` are strings composed of lowercase letters. In `S`, no letter occurs more than once.

`S` was sorted in some custom order previously. We want to permute the characters of `T` so that they match the order that `S` was sorted. More specifically, if `x` occurs before `y` in `S`, then `x` should occur before `y` in the returned string.

Return any permutation of `T` (as a string) that satisfies this property.

```
Example :
Input:
S = "cba"
T = "abcd"
Output: "cbad"
Explanation:
"a", "b", "c" appear in S, so the order of "a", "b", "c" should be "c", "b", and "a".

Since "d" does not appear in S, it can be at any position in T. "dcba", "cdba", "cbda
" are also valid outputs.
```

Note:

- `S` has length at most `26`, and no character is repeated in `S`.
- `T` has length at most `200`.
- `S` and `T` consist of lowercase letters only.

Solution 1

```
    public String customSortString(String S, String T) {
        int[] count = new int[26];
        for (char c : T.toCharArray()) { ++count[c - 'a']; }  // count each char in T.
        StringBuilder sb = new StringBuilder();
        for (char c : S.toCharArray()) {
            while (count[c - 'a']-- > 0) { sb.append(c); }    // sort chars both in T and S by the order of S.
        }
        for (char c = 'a'; c <= 'z'; ++c) {
            while (count[c - 'a']-- > 0) { sb.append(c); }   // group chars in T but not in S.
        }
        return sb.toString();
    }
    ```
```

written by rock original link here
```

Solution 2

```
class Solution {
  public:
    string customSortString(string S, string T) {
        sort(T.begin(), T.end(),
             [&](char a, char b) { return S.find(a) < S.find(b); });
        return T;
    }
};
```

The second Edition. It is easier to understand.

```
class Solution {
  public:
    string customSortString(string S, string T) {
        unordered_map<char, int> dir;
        for (int i = 0; i < S.size(); i++) {
            dir[S[i]] = i + 1;
        }
        sort(T.begin(), T.end(),
             [&](char a, char b) { return dir[a] < dir[b]; });
        return T;
    }
};
```

we also can use stl

```
class Solution {
  public:
    string customSortString(string S, string T) {
        unordered_map<char, int> dir;
        int i = 0;
        transform(S.begin(), S.end(), inserter(dir, dir.end()),
                  [&](char &a) { return make_pair(a, ++i); });
        sort(T.begin(), T.end(),
             [&](char a, char b) { return dir[a] < dir[b]; });
        return T;
    }
};
```

written by lailianqi original link here

Solution 3

Use bucket sort to achieve linear time.

1. Put string T into bucket;
2. Scan each character of S and construct the result string using bucket;
3. Scan bucket again to append the rest of characters which are not existing in string S.
   `Actually, this problem has two follow up questions:`
   `Follow up 1: If the custom order S is too large, how to solve it efficiently?`
   `Follow up 2: If the string T is too large, how to solve it efficiently?`

```
class Solution {
    public String customSortString(String S, String T) {
        int[] bucket = new int[26];
        for (char c : T.toCharArray()) {
            bucket[c - 'a']++;
        }

        StringBuilder sb = new StringBuilder();
        for (char c : S.toCharArray()) {
            for (int i = 0; i < bucket[c - 'a']; i++) {
                sb.append(c);
            }
            bucket[c - 'a'] = 0;
        }

        for (int i = 0; i < 26; i++) {
            for (int j = 0; j < bucket[i]; j++) {
                sb.append((char) (i + 'a'));
            }
        }

        return sb.toString();
    }
}
```

written by FLAGbigoffer original link here