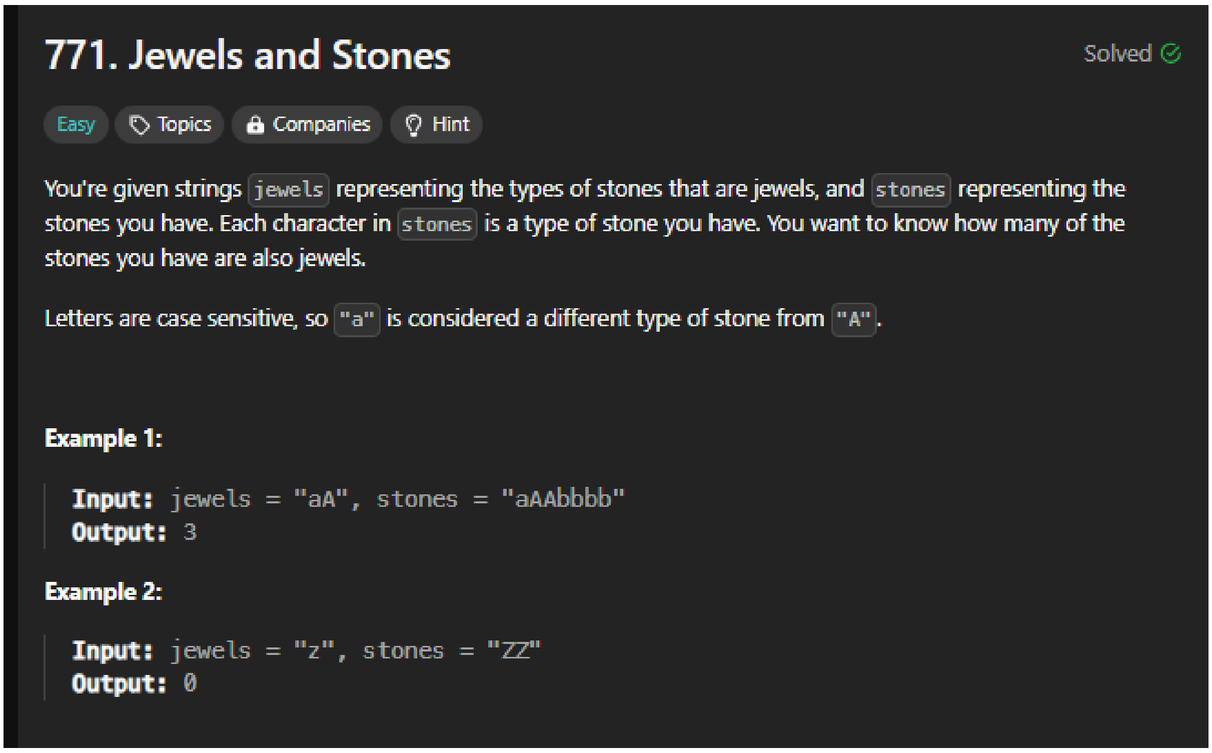
“LeetCode Problem 771: Jewels and Stones”

Question :



771. Jewels and Stones

You're given strings `jewels` representing the types of stones that are jewels, and `stones` representing the stones you have. Each character in `stones` is a type of stone you have. You want to know how many of the stones you have are also jewels.

Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"`.

Example 1:

```
Input: jewels = "aA", stones = "aAAbbbb"
Output: 3
```

Example 2:

```
Input: jewels = "z", stones = "ZZ"
Output: 0
```

You're given strings jewels representing the types of stones that are jewels, and stones representing the stones you have. Each character in stones is a type of stone you have. You want to know how many of the stones you have are also jewels.

Letters are case sensitive, so "a" is considered a different type of stone from "A".

Constraints :

- ➔ 1 <= jewels.length, stones.length <= 50
- ➔ jewels and stones consist of only English letters.
- ➔ All the characters of jewels are unique.

Inputs :

- ➔ jewels: A string where each character represents a type of jewel.
- ➔ stones: A string where each character represents a type of stone you possess.

Example 1:

Input: jewels = "aA", stones = "aAAbbbb"

Output: 3

Example 2:

Input: jewels = "z", stones = "ZZ"

Output: 0

Algorithm :

1. Sort the `jewels` string to allow efficient comparison.
2. Sort the `stones` string for linear traversal.
3. Use two pointers (`jindex` for `jewels` and `sindex` for `stones`) to compare characters.
4. If a match is found, increment the result counter and move the `sindex`.
5. If no match is found, adjust `jindex` and reset as necessary.

CODE :

```
int compare(const void* a, const void* b) { return *(char*)a - *(char*)b; }

// Custom comparator for sorting

int numJewelsInStones(char* jewels, char* stones) {


    qsort(jewels, strlen(jewels), sizeof(char), compare); // Sort jewels

    qsort(stones, strlen(stones), sizeof(char), compare); // Sort stones

    // Initialize variables

    int jlen = strlen(jewels);

    int jindex = 0;

    int sindex = 0;

    int res = 0;

    while (stones[sindex] != '\0') {

        if (stones[sindex] == jewels[jindex]) {
            res++;        // Count match
            sindex++;    // Move to next stone
        } else {
            jindex++;    // Check next jewel
            if (jindex == jlen) {
                sindex++;
                jindex = 0;     // Reset jewel index if no match
            }
        }
    }
    return res;    // Return the result
}
```

Time Complexity

- **Sorting the jewels string:**

  Sorting requires O(nlogn), where n is the length of the `jewels` string.
- **Sorting the stones string:**

  Sorting requires O(mlogm), where m is the length of the `stones` string.
- **Traversal of the stones string:**

  After sorting, we traverse the `stones` string with a two-pointer approach. The worst-case time for this traversal is O(m+n), where mmm is the length of `stones` and n is the length of `jewels`.
- **Total Time Complexity:**

  The dominant cost is the sorting step, so the overall time complexity is:

  O(nlogn+mlogm)

Edge Cases

- Mention edge cases and how the code handles them:
    - **No jewels match stones** (e.g., `jewels = "z"`, `stones = "abc"`).
    - **All stones are jewels** (e.g., `jewels = "a"`, `stones = "aaaa"`).
    - **Empty strings** (invalid due to constraints).