Who's the closest?

Given a string which might have multiple occurrences of the same character, return the closest same character of any indicated character in the string. You are given the string *s* and *n* number of queries. In each query, you are given an index *a* (where *0 ≤ a < |s|*) of a character, and you need to print the index of the closest same character. If there are multiple answers, print the smallest one, or if there is no such index print *-1* instead.

For example, for the string *s = babab,* with a given query *2,* there are two matching characters at indices *0* and *4,* each *2* away, so we choose the lower of the two: *0.*

Function Description

Complete the function *closest* in the editor below. The function must return an integer vector of size *n* denoting the answer of each query.

closest has the following parameters:

s: the original string

queries: an array of *n* integers, where the value of each element *queries[i]* is an index of a character whose closest same character's index needs to be found.

Constraints

- $|s|, |queries| \le 10^5$
- $1 \le n \le 10^5$
- *s* will contain only lowercase letters from the English alphabet, ascii[a-z]

▶ Input Format For Custom Testing

▼ Sample Case 0

Sample Input 0

```
hackerrank
4
4
1
6
8
```

Sample Output 0

```
-1
7
5
-1
```

Explanation 0

Query #0: Character at index-*4* is *'e'*. In this case, there is no other *'e'* present in *s*, so we print *-1*.

Query #1: Character at index-*1* is *'a'*. In this case, there is only one closest index (index-7) that contains *'a'*.

Query #2: Character at index-*6* is *'r'*. In this case, there is only one closest index (index-5) that contains *'r'*.

Query #3: Character at index-*8* is *'n'*. In this case, there is no other *'n'* present in *s*, so we print *-1*.

▶ Sample Case 1

▶ Sample Case 2

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour. Start tour

Original code | C

```
#include ↔

char* readline();
char* ltrim(char*);
char* rtrim(char*);


// Complete the closest function below.

/*
 * To return the integer array from the function, you should:
 *     - Store the size of the array to be returned in the result_count variable
 *     - Allocate the array statically or dynamically
 *
```

```c
 * For example,
 * int* return_integer_array_using_static_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     static int a[5] = {1, 2, 3, 4, 5};
 *
 *     return a;
 * }
 *
 * int* return_integer_array_using_dynamic_allocation(int* result_count) {
 *     *result_count = 5;
 *
 *     int *a = malloc(5 * sizeof(int));
 *
 *     for (int i = 0; i < 5; i++) {
 *         *(a + i) = i + 1;
 *     }
 *
 *     return a;
 * }
 *
 */
int* closest(char* s, int queries_count, int* queries, int* result_count) {


}


int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char* s = readline();

    char* queries_count_endptr;
    char* queries_count_str = ltrim(rtrim(readline()));
    int queries_count = strtol(queries_count_str, &queries_count_endptr, 10);

    if (queries_count_endptr == queries_count_str || *queries_count_endptr != '\0') { exit(EXIT_FAILURE); }

    int* queries = malloc(queries_count * sizeof(int));

    for (int i = 0; i < queries_count; i++) {
        char* queries_item_endptr;
        char* queries_item_str = ltrim(rtrim(readline()));
        int queries_item = strtol(queries_item_str, &queries_item_endptr, 10);

        if (queries_item_endptr == queries_item_str || *queries_item_endptr != '\0') { exit(EXIT_FAILURE); }

        *(queries + i) = queries_item;
    }

    int res_count;
    int* res = closest(s, queries_count, queries, &res_count);

    for (int i = 0; i < res_count; i++) {
        fprintf(fptr, "%d", *(res + i));

        if (i != res_count - 1) {
            fprintf(fptr, "\n");
        }
    }

    fprintf(fptr, "\n");

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

```

Line: 37 Col: 1

☐ Test against custom input

Run Code

Submit code & Continue

(You can submit any number of times)

Download sample test cases *The input/output files have Unix line endings. Do not use Notepad to edit them on windows.*