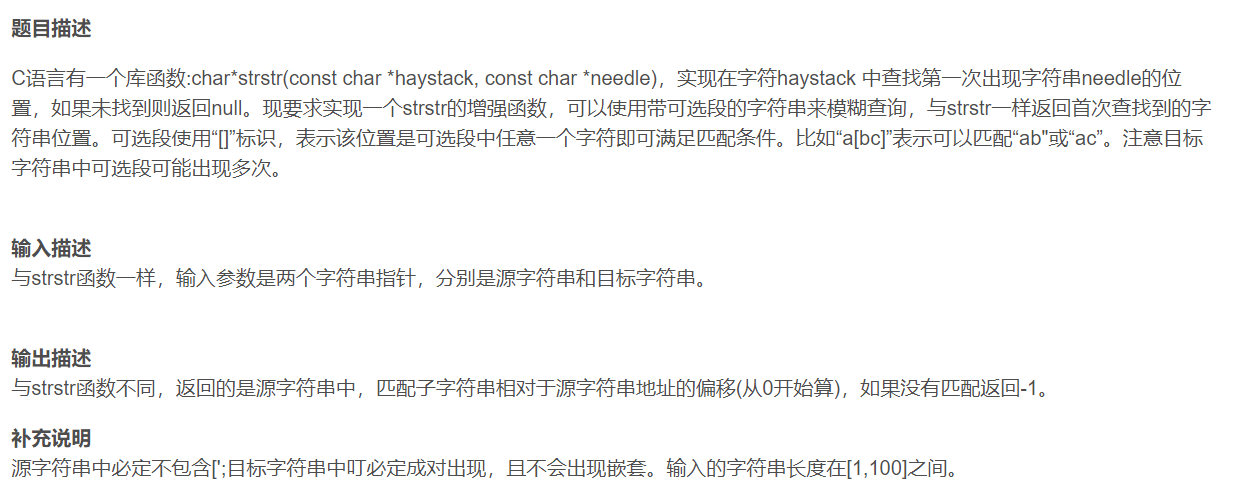
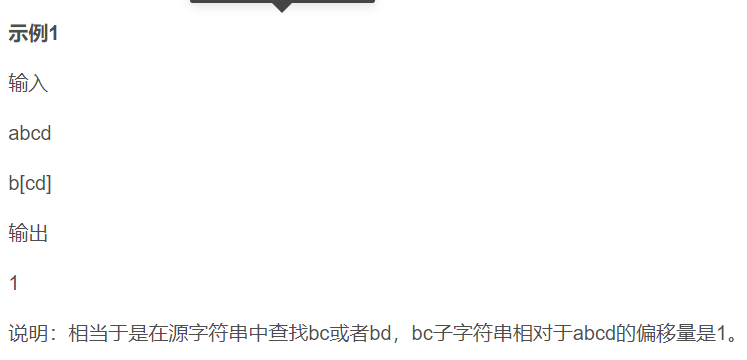
# **E卷-增强的strstr[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**







import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// 读取第一行输入，存储在字符串 s 中

String s = sc.nextLine();

// 读取第二行输入，存储在字符串 t 中

String t = sc.nextLine();

// 交换 s 和 t 的值

String tmp = s;

s = t;

t = tmp;

// 初始化装整数集合的 ArrayList，每个集合对应字符串 s 的一个位置

ArrayList<HashSet<Integer>> a = new ArrayList<>();

// 获取字符串 s 的长度

int sz = s.length();

// 遍历字符串 s

for (int i = 0; i < sz; ++i) {

// 如果当前字符不是 '['，创建一个包含该字符的集合

if (s.charAt(i) != '[') {

HashSet<Integer> now = new HashSet<>();

now.add((int) s.charAt(i));

a.add(now);

} else {

// 如果当前字符是 '['，处理括号内的所有字符

int j = i + 1;

HashSet<Integer> now = new HashSet<>();

while (s.charAt(j) != ']') {

now.add((int) s.charAt(j));

j += 1;

}

a.add(now);

i = j; // 更新 i 到 ']' 的位置

}

}

// 如果 a 集合的大小大于 t 的长度，说明匹配不可能

if (a.size() > t.length()) {

System.out.println(-1);

return;

}

// 在 t 中寻找是否有符合条件的子串

for (int i = 0; i <= t.length() - a.size(); i++) {

boolean found = true;

// 检查从 t 的第 i 位开始的子串是否满足要求

for (int j = 0; j < a.size(); j++) {

HashSet<Integer> now = a.get(j);

// 该位置的字符是否在当前集合中

if (!now.contains((int) t.charAt(i + j))) {

found = false;

break;

}

}

// 如果找到满足条件的子串，打印起始位置 i

if (found) {

System.out.print(i);

return;

}

}

// 如果没有找到符合要求的子串，输出 -1

System.out.print(-1);

}

}



import sys

# 读取两行输入，s 和 t

s = input().strip()

t = input().strip()

# 交换 s 和 t 的值

s, t = t, s

# 初始化装整数集合的列表，每个集合对应字符串 s 的一个位置

a = []

# 遍历字符串 s

sz = len(s)

i = 0

while i < sz:

if s[i] != "[":

now = set()

now.add(ord(s[i]))

a.append(now)

else:

j = i + 1

now = set()

while s[j] != "]":

now.add(ord(s[j]))

j += 1

a.append(now)

i = j

i += 1

# 如果 a 集合的大小大于 t 的长度，说明匹配不可能

if len(a) > len(t):

print(-1)

sys.exit()

# 在 t 中寻找是否有符合条件的子串

for i in range(len(t) - len(a) + 1):

found = True

for j in range(len(a)):

now = a[j]

if ord(t[i + j]) not in now:

found = False

break

if found:

print(i)

sys.exit()

# 如果没有找到符合要求的子串，输出 -1

print(-1)



#include <iostream>

#include <vector>

#include <unordered\_set>

#include <string>

using namespace std;

int main() {

string s, t;

getline(cin, s);

getline(cin, t);

// 交换 s 和 t 的值

string tmp = s;

s = t;

t = tmp;

// 初始化装整数集合的 vector，每个集合对应字符串 s 的一个位置

vector<unordered\_set<int>> a;

int sz = s.length();

// 遍历字符串 s

for (int i = 0; i < sz; ++i) {

if (s[i] != '[') {

unordered\_set<int> now;

now.insert((int)s[i]);

a.push\_back(now);

} else {

int j = i + 1;

unordered\_set<int> now;

while (s[j] != ']') {

now.insert((int)s[j]);

j += 1;

}

a.push\_back(now);

i = j; // 更新 i 到 ']' 的位置

}

}

// 如果 a 集合的大小大于 t 的长度，说明匹配不可能

if (a.size() > t.length()) {

cout << -1 << endl;

return 0;

}

// 在 t 中寻找是否有符合条件的子串

for (int i = 0; i <= t.length() - a.size(); i++) {

bool found = true;

for (int j = 0; j < a.size(); j++) {

unordered\_set<int> now = a[j];

if (now.find((int)t[i + j]) == now.end()) {

found = false;

break;

}

}

if (found) {

cout << i << endl;

return 0;

}

}

// 如果没有找到符合要求的子串，输出 -1

cout << -1 << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// 定义动态数组结构用于动态 growing arrays

typedef struct {

int\* data;

int size;

int capacity;

} IntArray;

// 初始化动态数组

void initArray(IntArray\* a) {

a->size = 0;

a->capacity = 10;

a->data = (int\*)malloc(sizeof(int) \* a->capacity);

}

// 向动态数组添加元素

void addToArray(IntArray\* a, int value) {

if (a->size == a->capacity) {

a->capacity \*= 2;

a->data = (int\*)realloc(a->data, sizeof(int) \* a->capacity);

}

a->data[a->size++] = value;

}

// 释放动态数组的内存

void freeArray(IntArray\* a) {

free(a->data);

}

int main() {

char s[1000], t[1000];

fgets(s, 1000, stdin);

fgets(t, 1000, stdin);

s[strlen(s) - 1] = '\0'; // 去掉换行符

t[strlen(t) - 1] = '\0';

// 交换 s 和 t 的值

char tmp[1000];

strcpy(tmp, s);

strcpy(s, t);

strcpy(t, tmp);

IntArray a[1000];

int n = 0;

// 遍历字符串 s

for (int i = 0; i < strlen(s); i++) {

if (s[i] != '[') {

initArray(&a[n]);

addToArray(&a[n], (int)s[i]);

n++;

} else {

initArray(&a[n]);

int j = i + 1;

while (s[j] != ']') {

addToArray(&a[n], (int)s[j]);

j += 1;

}

n++;

i = j; // 更新 i 到 ']' 的位置

}

}

// 如果 a 集合的大小大于 t 的长度，说明匹配不可能

if (n > strlen(t)) {

printf("-1\n");

for (int i = 0; i < n; i++) {

freeArray(&a[i]);

}

return 0;

}

// 在 t 中寻找是否有符合条件的子串

int found = 0;

for (int i = 0; i <= strlen(t) - n; i++) {

found = 1;

for (int j = 0; j < n; j++) {

int include = 0;

for (int k = 0; k < a[j].size; k++) {

if (a[j].data[k] == (int)t[i + j]) {

include = 1;

break;

}

}

if (!include) {

found = 0;

break;

}

}

if (found) {

printf("%d\n", i);

for (int i = 0; i < n; i++) {

freeArray(&a[i]);

}

return 0;

}

}

// 如果没有找到符合要求的子串，输出 -1

if (!found) printf("-1\n");

for (int i = 0; i < n; i++) {

freeArray(&a[i]);

}

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let input = [];

rl.on("line", function (line) {

input.push(line);

if (input.length === 2) {

let s = input[0];

let t = input[1];

// 交换 s 和 t 的值

[s, t] = [t, s];

// 初始化装字符集合的数组

let a = [];

let i = 0;

while (i < s.length) {

if (s[i] !== "[") {

let now = new Set();

now.add(s.charCodeAt(i));

a.push(now);

} else {

let j = i + 1;

let now = new Set();

while (s[j] !== "]") {

now.add(s.charCodeAt(j));

j += 1;

}

a.push(now);

i = j;

}

i += 1;

}

// 如果 a 的大小大于 t 的长度，说明匹配不可能

if (a.length > t.length) {

console.log(-1);

rl.close();

return;

}

// 在 t 中寻找是否有符合条件的子串

for (let i = 0; i <= t.length - a.length; i++) {

let found = true;

for (let j = 0; j < a.length; j++) {

let now = a[j];

if (!now.has(t.charCodeAt(i + j))) {

found = false;

break;

}

}

if (found) {

console.log(i);

rl.close();

return;

}

}

// 如果没有找到符合要求的子串，输出 -1

console.log(-1);

rl.close();

}

});



package main

import (

"bufio"

"fmt"

"os"

)

func main() {

reader := bufio.NewReader(os.Stdin)

s, \_ := reader.ReadString('\n')

t, \_ := reader.ReadString('\n')

// 去掉换行符

s = s[:len(s)-1]

t = t[:len(t)-1]

// 交换 s 和 t 的值

s, t = t, s

a := make([]map[int]bool, 0)

i := 0

sz := len(s)

// 遍历字符串 s

for i < sz {

if s[i] != '[' {

now := make(map[int]bool)

now[int(s[i])] = true

a = append(a, now)

} else {

now := make(map[int]bool)

j := i + 1

for s[j] != ']' {

now[int(s[j])] = true

j++

}

a = append(a, now)

i = j

}

i++

}

// 如果 a 集合的大小大于 t 的长度，说明匹配不可能

if len(a) > len(t) {

fmt.Println(-1)

return

}

// 在 t 中寻找是否有符合条件的子串

for i = 0; i <= len(t)-len(a); i++ {

found := true

for j := 0; j < len(a); j++ {

now := a[j]

if !now[int(t[i+j])] {

found = false

break

}

}

if found {

fmt.Println(i)

return

}

}

// 如果没有找到符合要求的子串，输出 -1

fmt.Println(-1)

}