JS:

function find\_substring(str, pattern) {

let windowStart = 0,

matched = 0,

substrStart = 0,

minLength = str.length + 1,

charFrequency = {};

for (i = 0; i < pattern.length; i++) {

const chr = pattern[i];

if (!(chr in charFrequency)) {

charFrequency[chr] = 0;

}

charFrequency[chr] += 1;

}

// try to extend the range [windowStart, windowEnd]

for (windowEnd = 0; windowEnd < str.length; windowEnd++) {

const rightChar = str[windowEnd];

if (rightChar in charFrequency) {

charFrequency[rightChar] -= 1;

if (charFrequency[rightChar] >= 0) { // Count every matching of a character

matched += 1;

}

}

// Shrink the window if we can, finish as soon as we remove a matched character

while (matched === pattern.length) {

if (minLength > windowEnd - windowStart + 1) {

minLength = windowEnd - windowStart + 1;

substrStart = windowStart;

}

const leftChar = str[windowStart];

windowStart += 1;

if (leftChar in charFrequency) {

// Note that we could have redundant matching characters, therefore we'll decrement the

// matched count only when a useful occurrence of a matched character is going out of the window

if (charFrequency[leftChar] === 0) {

matched -= 1;

}

charFrequency[leftChar] += 1;

}

}

}

if (minLength > str.length) {

return '';

}

return str.substring(substrStart, substrStart + minLength);

}

console.log(find\_substring('aabdec', 'abc'));

console.log(find\_substring('abdabca', 'abc'));

console.log(find\_substring('adcad', 'abc'));

C++:

using namespace std;

#include <iostream>

#include <string>

#include <unordered\_map>

class MinimumWindowSubstring {

public:

static string findSubstring(const string &str, const string &pattern) {

int windowStart = 0, matched = 0, minLength = str.length() + 1, subStrStart = 0;

unordered\_map<char, int> charFrequencyMap;

for (auto chr : pattern) {

charFrequencyMap[chr]++;

}

// try to extend the range [windowStart, windowEnd]

for (int windowEnd = 0; windowEnd < str.length(); windowEnd++) {

char rightChar = str[windowEnd];

if (charFrequencyMap.find(rightChar) != charFrequencyMap.end()) {

charFrequencyMap[rightChar]--;

if (charFrequencyMap[rightChar] >= 0) { // count every matching of a character

matched++;

}

}

// shrink the window if we can, finish as soon as we remove a matched character

while (matched == pattern.length()) {

if (minLength > windowEnd - windowStart + 1) {

minLength = windowEnd - windowStart + 1;

subStrStart = windowStart;

}

char leftChar = str[windowStart++];

if (charFrequencyMap.find(leftChar) != charFrequencyMap.end()) {

// note that we could have redundant matching characters, therefore we'll decrement the

// matched count only when a useful occurrence of a matched character is going out of the

// window

if (charFrequencyMap[leftChar] == 0) {

matched--;

}

charFrequencyMap[leftChar]++;

}

}

}

return minLength > str.length() ? "" : str.substr(subStrStart, minLength);

}

};

int main(int argc, char \*argv[]) {

cout << MinimumWindowSubstring::findSubstring("aabdec", "abc") << endl;

cout << MinimumWindowSubstring::findSubstring("abdabca", "abc") << endl;

cout << MinimumWindowSubstring::findSubstring("adcad", "abc") << endl;

}

Python

def find\_substring(str1, pattern):

window\_start, matched, substr\_start = 0, 0, 0

min\_length = len(str1) + 1

char\_frequency = {}

for chr in pattern:

if chr not in char\_frequency:

char\_frequency[chr] = 0

char\_frequency[chr] += 1

# try to extend the range [window\_start, window\_end]

for window\_end in range(len(str1)):

right\_char = str1[window\_end]

if right\_char in char\_frequency:

char\_frequency[right\_char] -= 1

if char\_frequency[right\_char] >= 0: # Count every matching of a character

matched += 1

# Shrink the window if we can, finish as soon as we remove a matched character

while matched == len(pattern):

if min\_length > window\_end - window\_start + 1:

min\_length = window\_end - window\_start + 1

substr\_start = window\_start

left\_char = str1[window\_start]

window\_start += 1

if left\_char in char\_frequency:

# Note that we could have redundant matching characters, therefore we'll decrement the

# matched count only when a useful occurrence of a matched character is going out of the window

if char\_frequency[left\_char] == 0:

matched -= 1

char\_frequency[left\_char] += 1

if min\_length > len(str1):

return ""

return str1[substr\_start:substr\_start + min\_length]

def main():

print(find\_substring("aabdec", "abc"))

print(find\_substring("abdabca", "abc"))

print(find\_substring("adcad", "abc"))

main()

Java:

import java.util.\*;

class MinimumWindowSubstring {

public static String findSubstring(String str, String pattern) {

int windowStart = 0, matched = 0, minLength = str.length() + 1, subStrStart = 0;

Map<Character, Integer> charFrequencyMap = new HashMap<>();

for (char chr : pattern.toCharArray())

charFrequencyMap.put(chr, charFrequencyMap.getOrDefault(chr, 0) + 1);

// try to extend the range [windowStart, windowEnd]

for (int windowEnd = 0; windowEnd < str.length(); windowEnd++) {

char rightChar = str.charAt(windowEnd);

if (charFrequencyMap.containsKey(rightChar)) {

charFrequencyMap.put(rightChar, charFrequencyMap.get(rightChar) - 1);

if (charFrequencyMap.get(rightChar) >= 0) // count every matching of a character

matched++;

}

// shrink the window if we can, finish as soon as we remove a matched character

while (matched == pattern.length()) {

if (minLength > windowEnd - windowStart + 1) {

minLength = windowEnd - windowStart + 1;

subStrStart = windowStart;

}

char leftChar = str.charAt(windowStart++);

if (charFrequencyMap.containsKey(leftChar)) {

// note that we could have redundant matching characters, therefore we'll decrement the

// matched count only when a useful occurrence of a matched character is going out of the window

if (charFrequencyMap.get(leftChar) == 0)

matched--;

charFrequencyMap.put(leftChar, charFrequencyMap.get(leftChar) + 1);

}

}

}

return minLength > str.length() ? "" : str.substring(subStrStart, subStrStart + minLength);

}

public static void main(String[] args) {

System.out.println(MinimumWindowSubstring.findSubstring("aabdec", "abc"));

System.out.println(MinimumWindowSubstring.findSubstring("abdabca", "abc"));

System.out.println(MinimumWindowSubstring.findSubstring("adcad", "abc"));

}

}