JS

function find\_permutation(str, pattern) {

let windowStart = 0,

matched = 0,

charFrequency = {};

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

const chr = pattern[i];

if (!(chr in charFrequency)) {

charFrequency[chr] = 0;

}

charFrequency[chr] += 1;

}

// Our goal is to match all the characters from the 'charFrequency' with the current window

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

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

const rightChar = str[windowEnd];

if (rightChar in charFrequency) {

// Decrement the frequency of matched character

charFrequency[rightChar] -= 1;

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

matched += 1;

}

}

if (matched === Object.keys(charFrequency).length) {

return true;

}

// Shrink the sliding window

if (windowEnd >= pattern.length - 1) {

leftChar = str[windowStart];

windowStart += 1;

if (leftChar in charFrequency) {

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

matched -= 1;

}

charFrequency[leftChar] += 1;

}

}

}

return false;

}

console.log(`Permutation exist: ${find\_permutation('oidbcaf', 'abc')}`);

console.log(`Permutation exist: ${find\_permutation('odicf', 'dc')}`);

console.log(`Permutation exist: ${find\_permutation('bcdxabcdy', 'bcdyabcdx')}`);

console.log(`Permutation exist: ${find\_permutation('aaacb', 'abc')}`);

C++

using namespace std;

#include <iostream>

#include <string>

#include <unordered\_map>

class StringPermutation {

public:

static bool findPermutation(const string &str, const string &pattern) {

int windowStart = 0, matched = 0;

unordered\_map<char, int> charFrequencyMap;

for (auto chr : pattern) {

charFrequencyMap[chr]++;

}

// our goal is to match all the characters from the 'charFrequencyMap' with the current window

// 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()) {

// decrement the frequency of the matched character

charFrequencyMap[rightChar]--;

if (charFrequencyMap[rightChar] == 0) { // character is completely matched

matched++;

}

}

if (matched == (int)charFrequencyMap.size()) {

return true;

}

if (windowEnd >= pattern.length() - 1) { // shrink the window

char leftChar = str[windowStart++];

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

if (charFrequencyMap[leftChar] == 0) {

matched--; // before putting the character back, decrement the matched count

}

// put the character back for matching

charFrequencyMap[leftChar]++;

}

}

}

return false;

}

};

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

cout << "Permutation exist: " << StringPermutation::findPermutation("oidbcaf", "abc") << endl;

cout << "Permutation exist: " << StringPermutation::findPermutation("odicf", "dc") << endl;

cout << "Permutation exist: " << StringPermutation::findPermutation("bcdxabcdy", "bcdyabcdx") << endl;

cout << "Permutation exist: " << StringPermutation::findPermutation("aaacb", "abc") << endl;

}

Python:

def find\_permutation(str1, pattern):

window\_start, matched = 0, 0

char\_frequency = {}

for chr in pattern:

if chr not in char\_frequency:

char\_frequency[chr] = 0

char\_frequency[chr] += 1

# our goal is to match all the characters from the 'char\_frequency' with the current window

# 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:

# decrement the frequency of matched character

char\_frequency[right\_char] -= 1

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

matched += 1

if matched == len(char\_frequency):

return True

# shrink the window by one character

if window\_end >= len(pattern) - 1:

left\_char = str1[window\_start]

window\_start += 1

if left\_char in char\_frequency:

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

matched -= 1

char\_frequency[left\_char] += 1

return False

def main():

print('Permutation exist: ' + str(find\_permutation("oidbcaf", "abc")))

print('Permutation exist: ' + str(find\_permutation("odicf", "dc")))

print('Permutation exist: ' + str(find\_permutation("bcdxabcdy", "bcdyabcdx")))

print('Permutation exist: ' + str(find\_permutation("aaacb", "abc")))

main()

Java:

import java.util.\*;

class StringPermutation {

public static boolean findPermutation(String str, String pattern) {

int windowStart = 0, matched = 0;

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

for (char chr : pattern.toCharArray())

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

// our goal is to match all the characters from the 'charFrequencyMap' with the current window

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

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

char rightChar = str.charAt(windowEnd);

if (charFrequencyMap.containsKey(rightChar)) {

// decrement the frequency of the matched character

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

if (charFrequencyMap.get(rightChar) == 0) // character is completely matched

matched++;

}

if (matched == charFrequencyMap.size())

return true;

if (windowEnd >= pattern.length() - 1) { // shrink the window by one character

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

if (charFrequencyMap.containsKey(leftChar)) {

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

matched--; // before putting the character back, decrement the matched count

// put the character back for matching

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

}

}

}

return false;

}

public static void main(String[] args) {

System.out.println("Permutation exist: " + StringPermutation.findPermutation("oidbcaf", "abc"));

System.out.println("Permutation exist: " + StringPermutation.findPermutation("odicf", "dc"));

System.out.println("Permutation exist: " + StringPermutation.findPermutation("bcdxabcdy", "bcdyabcdx"));

System.out.println("Permutation exist: " + StringPermutation.findPermutation("aaacb", "abc"));

}

}