JS

function length\_of\_longest\_substring(str, k) {

let windowStart = 0,

maxLength = 0,

maxRepeatLetterCount = 0,

frequencyMap = {};

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

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

const rightChar = str[windowEnd];

if (!(rightChar in frequencyMap)) {

frequencyMap[rightChar] = 0;

}

frequencyMap[rightChar] += 1;

maxRepeatLetterCount = Math.max(maxRepeatLetterCount, frequencyMap[rightChar]);

// Current window size is from windowStart to windowEnd, overall we have a letter which is

// repeating 'maxRepeatLetterCount' times, this means we can have a window which has one letter

// repeating 'maxRepeatLetterCount' times and the remaining letters we should replace.

// if the remaining letters are more than 'k', it is the time to shrink the window as we

// are not allowed to replace more than 'k' letters

if ((windowEnd - windowStart + 1 - maxRepeatLetterCount) > k) {

leftChar = str[windowStart];

frequencyMap[leftChar] -= 1;

windowStart += 1;

}

maxLength = Math.max(maxLength, windowEnd - windowStart + 1);

}

return maxLength;

}

console.log(length\_of\_longest\_substring('aabccbb', 2));

console.log(length\_of\_longest\_substring('abbcb', 1));

console.log(length\_of\_longest\_substring('abccde', 1));

C++

using namespace std;

#include <iostream>

#include <string>

#include <unordered\_map>

class CharacterReplacement {

public:

static int findLength(const string &str, int k) {

int windowStart = 0, maxLength = 0, maxRepeatLetterCount = 0;

unordered\_map<char, int> letterFrequencyMap;

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

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

char rightChar = str[windowEnd];

letterFrequencyMap[rightChar]++;

maxRepeatLetterCount = max(maxRepeatLetterCount, letterFrequencyMap[rightChar]);

// current window size is from windowStart to windowEnd, overall we have a letter which is

// repeating 'maxRepeatLetterCount' times, this means we can have a window which has one

// letter repeating 'maxRepeatLetterCount' times and the remaining letters we should replace.

// if the remaining letters are more than 'k', it is the time to shrink the window as we

// are not allowed to replace more than 'k' letters

if (windowEnd - windowStart + 1 - maxRepeatLetterCount > k) {

char leftChar = str[windowStart];

letterFrequencyMap[leftChar]--;

windowStart++;

}

maxLength = max(maxLength, windowEnd - windowStart + 1);

}

return maxLength;

}

};

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

cout << CharacterReplacement::findLength("aabccbb", 2) << endl;

cout << CharacterReplacement::findLength("abbcb", 1) << endl;

cout << CharacterReplacement::findLength("abccde", 1) << endl;

}

Python:

def length\_of\_longest\_substring(str1, k):

window\_start, max\_length, max\_repeat\_letter\_count = 0, 0, 0

frequency\_map = {}

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

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

right\_char = str1[window\_end]

if right\_char not in frequency\_map:

frequency\_map[right\_char] = 0

frequency\_map[right\_char] += 1

max\_repeat\_letter\_count = max(

max\_repeat\_letter\_count, frequency\_map[right\_char])

# Current window size is from window\_start to window\_end, overall we have a letter which is

# repeating 'max\_repeat\_letter\_count' times, this means we can have a window which has one letter

# repeating 'max\_repeat\_letter\_count' times and the remaining letters we should replace.

# if the remaining letters are more than 'k', it is the time to shrink the window as we

# are not allowed to replace more than 'k' letters

if (window\_end - window\_start + 1 - max\_repeat\_letter\_count) > k:

left\_char = str1[window\_start]

frequency\_map[left\_char] -= 1

window\_start += 1

max\_length = max(max\_length, window\_end - window\_start + 1)

return max\_length

def main():

print(length\_of\_longest\_substring("aabccbb", 2))

print(length\_of\_longest\_substring("abbcb", 1))

print(length\_of\_longest\_substring("abccde", 1))

main()

Java:

import java.util.\*;

class CharacterReplacement {

public static int findLength(String str, int k) {

int windowStart = 0, maxLength = 0, maxRepeatLetterCount = 0;

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

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

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

char rightChar = str.charAt(windowEnd);

letterFrequencyMap.put(rightChar, letterFrequencyMap.getOrDefault(rightChar, 0) + 1);

maxRepeatLetterCount = Math.max(maxRepeatLetterCount, letterFrequencyMap.get(rightChar));

// current window size is from windowStart to windowEnd, overall we have a letter which is

// repeating 'maxRepeatLetterCount' times, this means we can have a window which has one letter

// repeating 'maxRepeatLetterCount' times and the remaining letters we should replace.

// if the remaining letters are more than 'k', it is the time to shrink the window as we

// are not allowed to replace more than 'k' letters

if (windowEnd - windowStart + 1 - maxRepeatLetterCount > k) {

char leftChar = str.charAt(windowStart);

letterFrequencyMap.put(leftChar, letterFrequencyMap.get(leftChar) - 1);

windowStart++;

}

maxLength = Math.max(maxLength, windowEnd - windowStart + 1);

}

return maxLength;

}

public static void main(String[] args) {

System.out.println(CharacterReplacement.findLength("aabccbb", 2));

System.out.println(CharacterReplacement.findLength("abbcb", 1));

System.out.println(CharacterReplacement.findLength("abccde", 1));

}

}