20. 滑動窗口

2025資訊研究社 算法班
Made with ❤️ by jheanlee

什麼是滑動窗口

滑動窗口(sliding window)為雙指針(two pointers)的延伸，透過窗口的身長或縮短維持窗口內資料的特性。

滑動窗口常用於子序列、子字串或連續性資料之分析。

滑動窗口的架構

寫滑動窗口一定要先定義:

窗口的意義 - 窗口內的資料代表什麼？要記錄什麼？

窗口的型態 - 固定大小？動態變化？

窗口的資料變化 - 何時擴大窗口？何時縮小窗口？

Leetcode 1456 - Maximum Number of Vowels in a Substring of Given Length

給予字串s及正整數k，回傳任意長度為k之子字串中，具有最多英文母音字母的數量。

Example 1:

> **Input:** s = "abciiidef", k = 3
> **Output:** 3
> **Explanation:** The substring "iii" contains 3 vowel letters.

Example 2:

> **Input:** s = "aeiou", k = 2
> **Output:** 2
> **Explanation:** Any substring of length 2 contains 2 vowels.

Example 3:

> **Input:** s = "leetcode", k = 3
> **Output:** 2
> **Explanation:** "lee", "eet" and "ode" contain 2 vowels.

Leetcode 1456 - Maximum Number of Vowels in a Substring of Given Length

窗口的意義- 窗口為各個子字串，並記錄含有之母音字母數量

窗口的型態- 固定大小為k

窗口的資料變化- 每次均將窗口右移(右側邊界縮小、左側邊界擴大

Leetcode 1456 - Maximum Number of Vowels in a Substring of Given Length

```cpp
class Solution {
private:
  bool isVowel(char &c) {
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
  }
public:
  int maxVowels(string s, int k) {
    int max_c = 0, c = 0;
    int l = 0, r = 0;

    while (r < s.size() && r - l < k) {
      r++;
      c += (int) isVowel(s[r - 1]);
      max_c = max(max_c, c);
    }

    while (r < s.size()) {
      r++;
      c += (int) isVowel(s[r - 1]);

      while (r - l > k) {
        c -= (int) isVowel(s[l]);
        l++;
      }

      max_c = max(max_c, c);
    }

    return max_c;
  }
};
```

Leetcode 187 - Repeated DNA Sequences

給予由 'A', 'C', 'G', 'T' 組成的字串，請找出所有重複、10字元長的序列，順序不限。

Example 1:

```
Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
Output: ["AAAAACCCCC","CCCCCAAAAA"]
```

Example 2:

```
Input: s = "AAAAAAAAAAAAA"
Output: ["AAAAAAAAAA"]
```

Leetcode 187 - Repeated DNA Sequences

窗口的意義 - 窗口為各DNA子序列

窗口的型態 - 固定大小為10

窗口的資料變化 - 每次均將窗口右移(右側邊界縮小、左側邊界擴大

Leetcode 187 - Repeated DNA Sequences

作法一 unordered_map + string

把所有的子序列截下來放進map中計數

```
class Solution {
public:
  vector<string> findRepeatedDnaSequences(string s) {
    if (s.size() < 10) return vector<string> ();
    unordered_map<string, int> seq;
    vector<string> rtn;

    for (int i = 0; i <= s.size() - 10; i++) {
      seq[string(s, i, 10)]++;
    }

    for (const pair<string, int> &p: seq) {
      if (p.second > 1) {
        rtn.push_back(p.first);
      }
    }

    return rtn;
  }
};
```

Leetcode 187 - Repeated DNA Sequences

作法二 unordered_map + 位元運算

位元運算是壓縮空間常用的技巧

設'A'為0, 'C'為1, 'G'為2, 'T'為3

每次將數值乘以4（左移2位元）
並去掉第20位元以後的值（% 4^10）
再加上新加入窗口內的值

這樣可以**省去建構新字串的時間**、空間，以及減少 unordered_map 雜湊運算的時間

Leetcode 187 - Repeated DNA Sequences

```cpp
class Solution {
public:
  vector<string> findRepeatedDnaSequences(string s) {
    if (s.size() < 10) return vector<string> ();
    unordered_map<unsigned int, int> seq;
    vector<string> rtn;

    unsigned int val = 0;

    for (int i = 0; i < 10; i++) {
      val <<= 2;
      val &= 1048575; //  4^10 - 1

      switch (s[i]) {
      case 'A':
        val += 0;
        break;
      case 'C':
        val += 1;
        break;
      case 'G':
        val += 2;
        break;
      case 'T':
        val += 3;
        break;
      }
    }
    seq[val] = 1;
    int l = 0, r = 10;  //  window [l, r)
    while (r < s.size()) {
      l++; r++;
      val <<= 2;
      val &= 1048575;

      switch (s[r - 1]) {
      case 'A':
        val += 0;
        break;
      case 'C':
        val += 1;
        break;
      case 'G':
        val += 2;
        break;
      case 'T':
        val += 3;
        break;
      }

      seq[val]++;
      if (seq[val] == 2) {
        rtn.push_back(string(s, l, 10));
      }
    }

    return rtn;

  }
};
```

3. Longest Substring Without Repeating Characters

給予一個字串，回傳不含重複字元的最長子字串長度

字串可能含有字母、數字或符號

Example 1:

> **Input:** s = "abcabcbb"
> **Output:** 3
> **Explanation:** The answer is "abc", with the length of 3. Note that `"bca"` and `"cab"` are also correct answers.

Example 2:

> **Input:** s = "bbbbb"
> **Output:** 1
> **Explanation:** The answer is "b", with the length of 1.

Example 3:

> **Input:** s = "pwwkew"
> **Output:** 3
> **Explanation:** The answer is "wke", with the length of 3.
> Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

3. Longest Substring Without Repeating Characters

窗口的意義- 窗口為不含重複字元的子字串，紀錄最大長度

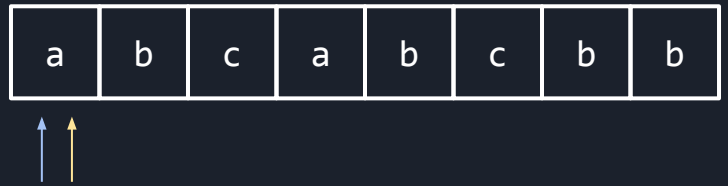
窗口的型態- 動態

窗口的資料變化- 持續擴大右側邊界，一但發現有重複便縮小左側邊界

3. Longest Substring Without Repeating Characters

a : 0個
b : 0個
c : 0個

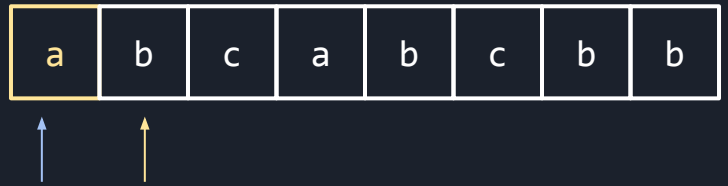
len : 0
max : 0



3. Longest Substring Without Repeating Characters

a : 1個
b : 0個
c : 0個

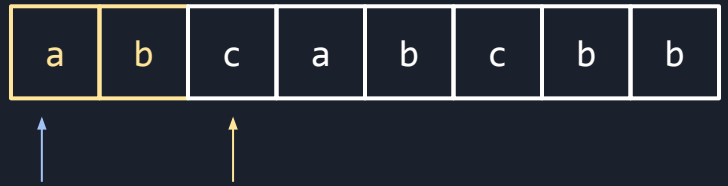
len : 1
max : 1



3. Longest Substring Without Repeating Characters

a : 1個
b : 1個
c : 0個

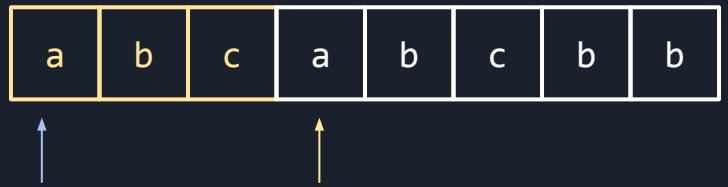
len : 2
max : 2



3. Longest Substring Without Repeating Characters

a : 1個
b : 1個
c : 1個

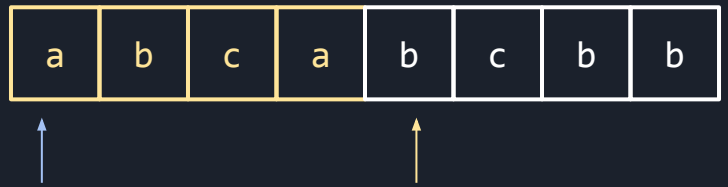
len : 3
max : 3



3. Longest Substring Without Repeating Characters

a : 2個
b : 1個
c : 1個

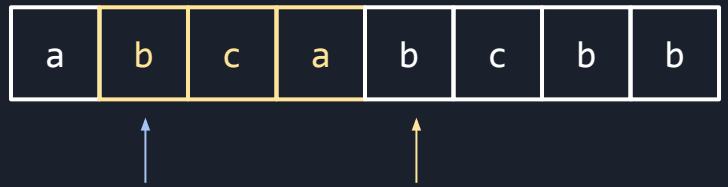
~~len : 4~~
max : 3



3. Longest Substring Without Repeating Characters

a : 2 -> 1個
b : 1個
c : 1個

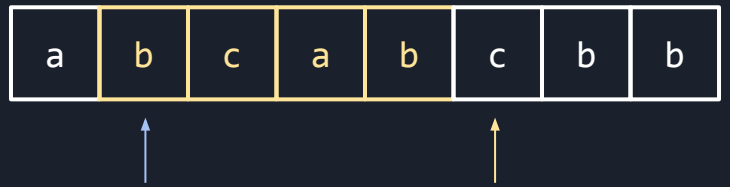
len : 3
max : 3



3. Longest Substring Without Repeating Characters

a : 1個
b : 2個
c : 1個

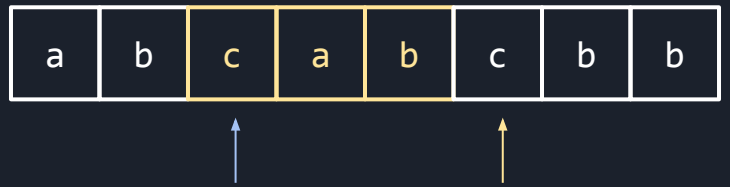
~~len : 4~~
max : 3



3. Longest Substring Without Repeating Characters

a : 1個
b : 2 -> 1個
c : 1個

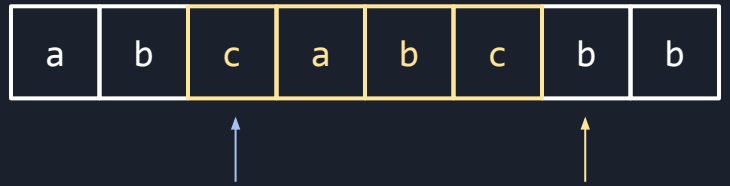
len : 3
max : 3



3. Longest Substring Without Repeating Characters

a : 1個
b : 1個
c : 2個

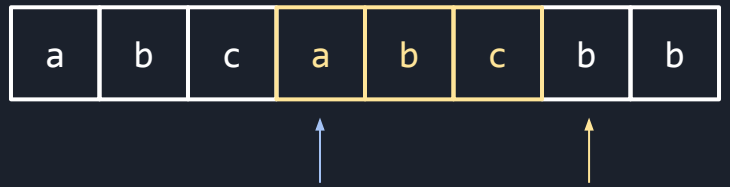
~~len : 4~~
max : 3



3. Longest Substring Without Repeating Characters

a : 1個
b : 1個
c : 2 -> 1個

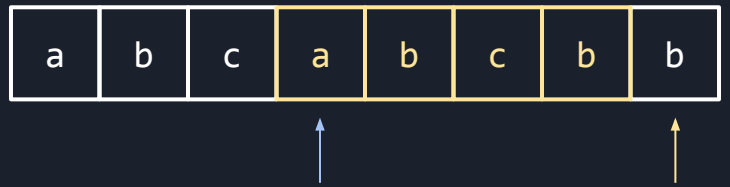
len : 3
max : 3



3. Longest Substring Without Repeating Characters

a : 1個
b : 2個
c : 1個

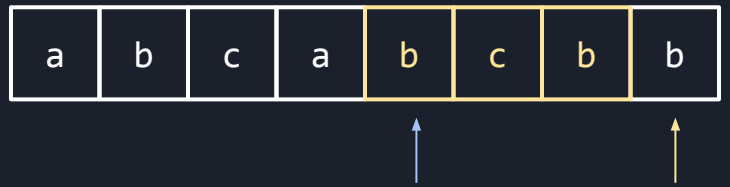
~~len : 4~~
max : 3



3. Longest Substring Without Repeating Characters

a : 1 -> 0個
b : 2個
c : 1個

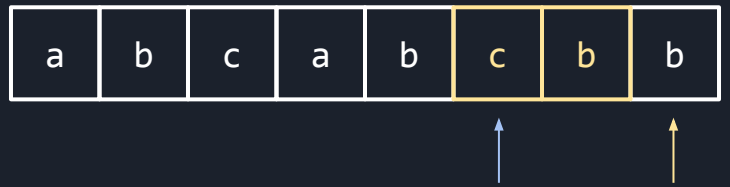
~~len : 3~~
max : 3



3. Longest Substring Without Repeating Characters

a : 0個
b : 2 -> 1個
c : 1個

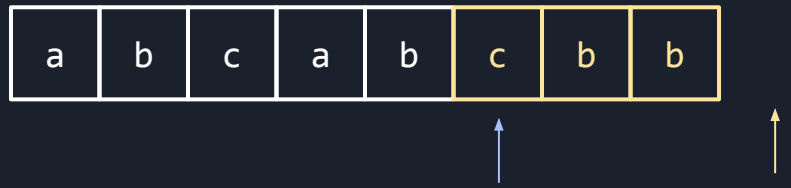
len : 2
max : 3



3. Longest Substring Without Repeating Characters

a : 0個
b : 2個
c : 1個

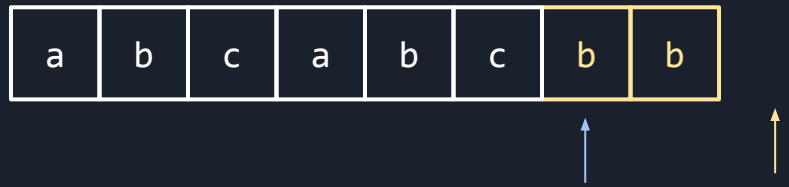
~~len : 3~~
max : 3



3. Longest Substring Without Repeating Characters

a : 0個
b : 2個
c : 1 -> 0個

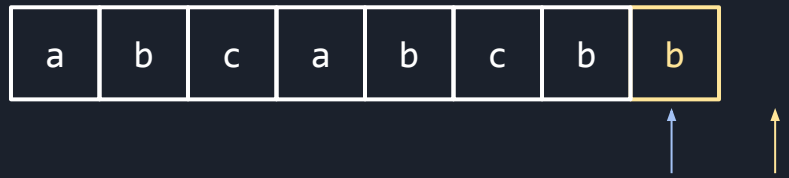
~~len : 2~~
max : 3



3. Longest Substring Without Repeating Characters

a : 0個
b : 2 -> 1個
c : 0個

len : 1
max : 3



3. Longest Substring Without Repeating Characters

```cpp
class Solution {
public:
  int lengthOfLongestSubstring(string s) {
    if (s.size() == 0 || s.size() == 1) {
      return s.size();
    }
    vector<bool> seen(128, false);
    int l = 0, r = 0, max_len = 0;

    while (r < s.size()) {
      while (seen[s[r]]) {
        seen[s[l]] = false;
        l++;
        max_len = max(max_len, r - l + 1);
      }
      max_len = max(max_len, r - l + 1);
      seen[s[r]] = true;
      r++;
    }

    return max_len;
  }
};
```