STRING

Problem – 3. Longest Substring Without Repeating Characters

Medium

LeetCode leetcode.com/problems/longest-substring-without-repeating-characters

Problem Statement

- You are given a string and the goal is to find the longest substring without repeating characters
- **Example**

  Input: “abcdb”

  Output: 4 (abcd since “b” is repeated)

Solution – 3. Longest Substring Without Repeating Characters

Medium

LeetCode leetcode.com/problems/longest-substring-without-repeating-characters

Solution

- Use sliding window algorithm (left and right)
- Loop through the string
- Try to find if the current character is already added by using unordered set or bitmap
- If added, remove from the set alongside with others using left pointer
- If not, add to the unordered set or bitmap
- Maximum length will be right – left + 1

Example – 3. Longest Substring Without Repeating Characters

Medium

LeetCode leetcode.com/problems/longest-substring-without-repeating-characters

Example

- String: abcbd. Our goal is to return 3 (**abc**bd)
- Initialize `maxLength = 0`
- Loop through the string

Iteration 1: `left = 0, right = 0, string[left] = 'a',`
`bitmap = ['a']` ('a' is not in bitmap, add), `maxLength = max(maxLength, right - left + 1) = 1`

Iteration 2: `left = 0, right = 1, string[right] = 'b'`
`bitmap = ['a','b'], maxLength = 2`

Iteration 3: `left = 0, right = 2, string[right] = 'c'`
`bitmap = ['a','b', 'c'], maxLength = 3`

Iteration 4: `left = 0, right = 3, string[right] = 'b'`
`bitmap = ['a','b','c','b']`
'b' is already in the bitmap. start "clearing" the character using left:

Iteration 4a: `left = 0, string[left] = 'a'` is different from 'b', so remove 'a'
`bitmap = ['b', 'c','b']`

Iteration 4b: `left = 1, string[left] = 'b'` is the same as the repeated one, remove
`bitmap = ['c','b']`

Iteration 5: `left = 1, right = 4, string[right] = 'd'`
`bitmap = ['c','b','d']`

Code – 3. Longest Substring Without Repeating Characters

Medium

Code (unordered_set)

- Use unordered_set when question requires unicode chars

```
int lengthOfLongestSubstring(string s) {
    int maxLength = 0;
    int left = 0, right = 0;
    // track the seen characters
    unordered_set<char> seen;
    for (right = 0; right < s.size(); ++right) {
        char currentChar = s[right];
        // if currentChar is in the set, clean
        // the character and everything from left of it
        // basically, reset the longest substring
        while (seen.count(currentChar)) {
            char c = s[left];
            seen.erase(c);
            left++;
        }
        // insert the current read character
        seen.insert(currentChar);
        // set max length
        maxLength = max(maxLength, right - left + 1);
    }
    return maxLength;
}
```

Code – 3. Longest Substring Without Repeating Characters

Medium

Code (bitmap)

- Using bitset: create a bitmask with 128 bits where each bit represent a character
- Optimal solution for ASCII since ASCII size is 127 characters
- Unicode / UTF-8 can represent over 1.1 million characters, so use **unordered_set** approach instead

```
int lengthOfLongestSubstring(string s) {
    std::bitset<128> bitmask;
    uint32_t left = 0;
    uint32_t maxLength = 0;

    for (uint32_t right = 0; right < s.length(); ++right) {
        uint32_t bitIndex = s[right];
        // if char is already in the bitmask, move left until we reset the bits
        while (bitmask.test(bitIndex)) {
            bitmask.reset(s[left]);
            ++left;
        }

        bitmask.set(bitIndex);
        maxLength = std::max(maxLength, right - left + 1);
    }
    return maxLength;
}
```

Problem – Valid Parentheses

Easy

LeetCode leetcode.com/problems/valid-parentheses

Problem Statement

- You are given a string containing only the characters **'(', ')', '{', '}', '[' and ‘]’**
- A valid input have closed brackets by its own type
- **Example**

  ()[]{} → valid

  []{}(→ invalid

  {()} → valid

Solution – Valid Parentheses

Easy

LeetCode leetcode.com/problems/valid-parentheses

Solution

- Loop through the string
- If **open** brackets **([{** push to a stack
- If **closed** brackets:

  pop the last added bracket

  check if the **closed** bracket corresponds to the **popped** bracket

  if not, return false
- after the loop, **return true** if the **size** of the stack is empty (all brackets closed)

Code – Valid Parentheses

Easy

LeetCode leetcode.com/problems/valid-parentheses

Code

Time: **O(n)** Space: **O(n)**

```
bool isValid(string s) {
    // stack (LIFO)
    std::stack<char> brackets;
    // O(n)
    for (int i = 0; i < s.size(); ++i) {
        char bracket = s[i];
        if (bracket == '(' || bracket == '[' || bracket == '{') {
            brackets.push(bracket);
        } else {
            if (brackets.size() == 0) return false;
            char lastBracket = brackets.top();
            if (bracket == ')' && lastBracket != '(') return false;
            if (bracket == '}' && lastBracket != '{') return false;
            if (bracket == ']' && lastBracket != '[') return false;
            brackets.pop();
        }
    }
    // all brackets must be closed
    return brackets.size() == 0;
}
```

Problem – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Problem Statement

- You are given an array of integers initialized with zeros (e.g. **[0,0,0,0]**)
- The goal is to reach some target (e.g. **[1, 2, 2, 3]**)
- The valid operations is to increment a subarray by one
- The output is the total number of operations

  In this case:

  [1,1,1,1] → increment the subarray starting from 0 to total size

  [1,2,2,2] → increment the subarray starting from 1 to total size

  [1,2,2,3] → increment the subarray starting and ending from the last element

  Output: 3 (total number of operations)

Solution – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Solution

- Explain…

Code (2) – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Code (optimized)

```
int minNumberOperations(vector<int>& target) {
    return target[0] +
        inner_product(target.begin() + 1, target.end(),
                target.begin(), 0,
                plus<int>(),
                [](int curr, int prev) { return max(curr - prev, 0); });
}
```

Code – Minimum Number of Increments on Subarrays

Hard

LeetCode leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array

Code

```
int minNumberOperations(vector<int>& target) {
    int totalOp = target[0];
    for (int i = 1; i < target.size(); ++i) {
        // can't reuse
        if (target[i - 1] < target[i]) {
            totalOp += target[i] - target[i - 1];
        }
    }
    return totalOp;
}
```