**Longest Substring Without Repeating Characters**

**Problem:** Given a string, find the length of the **longest substring** without repeating characters

String

Hash Table

Sliding Window

Keywords:

**Example:** “abcbcacbb”

**Solutions:** abc, bca, acb

**Approach:** Forget about the algorithms and coding, how do you solve this manually using a pen and paper?

* Start collecting the letters from the beginning of the string until the already visited node appears again. So, it is “abc”.
* Now change your start point to already visited node + 1 and again repeat the same. So, it is “cb”
* Repeat this process till you reach the end of the string

Let’s understand the algorithm step by step: We need three four variables to track the information. Start Point, End Point, Visited Nodes and Output.

Initialize, start = 0, end = 0, visited = {}, output = ‘’

start = 0; end = 0, visited = {a: 0}, output = ‘a’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

start = 0; end = 1, visited = {a: 0, b: 1}, output = ‘ab’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

start = 0; end = 2, visited = {a: 0, b: 1, c: 2}, output = ‘abc’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

start = 0; end = 3, visited = {a: 0, b: 3, c: 2}, output = ‘abc’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

Since, next visiting node “b” already exists in the visited dictionary now we will move starting point to visited[b] + 1 and we will not update output till we see the at least same length string.

start = visited[b] + 1 = 2; end = 4, visited = {a: 0, b: 3, c: 4}, output = ‘abc’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

start = visited[c] + 1 = 3; end = 5, visited = {a: 5, b: 3, c: 4}, output = ‘bca’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

**Caution:** “a” is already in the visited dictionary if we are tempted to use visited[a]+1 as starting point then our starting point goes back to 1. So, we need to use maximum of visited[state]+1 and start point

start = 3; end = 6, visited = {a: 5, b: 3, c: 6}, output = ‘bca’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

start = visited[c] + 1 = 5, end = 7, visited = {a: 5, b: 7, c: 6}, output = ‘acb’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

start = 5, end = 8, visited = {a: 5, b: 8, c: 6}, output = ‘acb’

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | b | c | a | c | b | b |

**Time Complexity:** O(n). End pointer moves till the end of the string