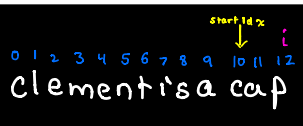
Input: A string

string: "clementisa cap"

Output: longest substring without duplicate characters

"mentisac"



lastSeen = {
"c": 10, "i": 7,
"l": 1, "s": 8,
"e": 4, "a": 11,
"m": 3, "p": 12,
"n": 5,
"t": 6

longest = [3, 11]

- Create a lastSeen variable that holds the last seen index of a certain char (a hash table)
- Set the longest variable to be the first char (0 to 1] (exclusive)
- Assign the startIdx to 0

- Iterate thru string. Grab char at index i
  - if char in lastSeen, set the startIdx to the max of (startIdx, lastSeen[char] + 1)
  - if (longest[1] - longest[0] < i + 1 - startIdx) update longest to [startIdx, i + 1]
  - Add char to lastSeen with its index value
- Return string.slice(longest[0], longest[1])

```
// O(n) time | O(min(n, a)) space
function longestSubstringWithoutDuplication(string) {
  const lastSeen = {};
  let startIdx = 0;
  let longest = [0, 1];

  for (let i = 0; i < string.length; i++) {
    const char = string[i];
    if (char in lastSeen) {
      startIdx = Math.max(startIdx, lastSeen[char] + 1);
    }
    if (longest[1] - longest[0] < i + 1 - startIdx) {
      longest = [startIdx, i + 1];
    }
    lastSeen[char] = i;
  }

  return string.slice(longest[0], longest[1]);
}
```

Time: O(n) (where n is the length of the input string) bc we are iterating through the string (rest of operations are O(1))

Space: O(min(n, a)) (where a is the length of the alphabet represented in our string) bc:

- If all letters in string are unique, space would be O(n)
- Otherwise O(a) since our hash table, we would not store duplicates