AlgoExpert Quad Layout JavaScript 12px Sublime Monokai 00:00:00

Prompt Scratchpad Our Solution(s) Video Explanation Run Code

Solution 1

```javascript
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

// O(b + s) time | O(b + s) space - where b is the length of the big
// input string and s is the length of the small input string
function smallestSubstringContaining(bigString, smallString) {
  const targetCharCounts = getCharCounts(smallString);
  const substringBounds = getSubstringBounds(bigString, targetCharCounts);
  return getStringFromBounds(bigString, substringBounds);
}

function getCharCounts(string) {
  const charCounts = {};
  for (const char of string) {
    increaseCharCount(char, charCounts);
  }
  return charCounts;
}

function getSubstringBounds(string, targetCharCounts) {
  let substringBounds = [0, Infinity];
  const substringCharCounts = {};
  const numUniqueChars = Object.keys(targetCharCounts).length;
  let numUniqueCharsDone = 0;
  let leftIdx = 0;
  let rightIdx = 0;
  // Move the rightIdx to the right in the string until you've counted
  // all of the target characters enough times.
  while (rightIdx < string.length) {
    const rightChar = string[rightIdx];
    if (!(rightChar in targetCharCounts)) {
      rightIdx++;
      continue;
    }
    increaseCharCount(rightChar, substringCharCounts);
    if (substringCharCounts[rightChar] === targetCharCounts[rightChar]) {
      numUniqueCharsDone++;
    }
    // Move the leftIdx to the right in the string until you no longer
    // have enough of the target characters in between the leftIdx and
    // the rightIdx. Update the substringBounds accordingly.
    while (numUniqueCharsDone === numUniqueChars && leftIdx <= rightIdx) {
      substringBounds = getCloserBounds(leftIdx, rightIdx, substringBounds[0], substringBounds[1]);
      const leftChar = string[leftIdx];
      if (!(leftChar in targetCharCounts)) {
        leftIdx++;
        continue;
      }
      if (substringCharCounts[leftChar] === targetCharCounts[leftChar]) {
        numUniqueCharsDone--;
      }
      decreaseCharCount(leftChar, substringCharCounts);
      leftIdx++;
    }
    rightIdx++;
  }
  return substringBounds;
}

function getCloserBounds(idx1, idx2, idx3, idx4) {
  return idx2 - idx1 < idx4 - idx3 ? [idx1, idx2] : [idx3, idx4];
}

function getStringFromBounds(string, bounds) {
  const [start, end] = bounds;
  if (end === Infinity) return '';
  return string.slice(start, end + 1);
}

function increaseCharCount(char, charCounts) {
  charCounts[char] = (charCounts[char] || 0) + 1;
}

function decreaseCharCount(char, charCounts) {
  charCounts[char]--;
}

exports.smallestSubstringContaining = smallestSubstringContaining;
```