Solution 1

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

package main

import "math"

// 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
func SmallestSubstringContaining(bigString, smallString string) string {
  targetCharCounts := getCharCounts(smallString)
  substringBounds := getSubstringBounds(bigString, targetCharCounts)
  return getStringFromBounds(bigString, substringBounds)
}

func getCharCounts(str string) map[byte]int {
  charCounts := map[byte]int{}
  for _, char := range str {
    increaseCharCount(byte(char), charCounts)
  }
  return charCounts
}

func getSubstringBounds(str string, targetCharCounts map[byte]int) []int {
  substringBounds := []int{0, math.MaxInt32}
  substringCharCounts := map[byte]int{}
  numUniqueChars := len(targetCharCounts)
  numUniqueCharsDone := 0
  leftIdx := 0
  rightIdx := 0

  // Move the rightIdx to the right in the string until you've counted
  // all of the target characters enough times.
  for rightIdx < len(str) {
    rightChar := str[rightIdx]
    if _, found := targetCharCounts[rightChar]; !found {
      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.
    for numUniqueCharsDone == numUniqueChars && leftIdx <= rightIdx {
      substringBounds = getCloserBounds(
        leftIdx, rightIdx, substringBounds[0], substringBounds[1],
      )
      leftChar := str[leftIdx]
      if _, found := targetCharCounts[leftChar]; !found {
        leftIdx++
        continue
      }
      if substringCharCounts[leftChar] == targetCharCounts[leftChar] {
        numUniqueCharsDone--
      }
      decreaseCharCount(leftChar, substringCharCounts)
      leftIdx++
    }
    rightIdx++
  }
  return substringBounds
}

func getCloserBounds(idx1, idx2, idx3, idx4 int) []int {
  if idx2-idx1 < idx4-idx3 {
    return []int{idx1, idx2}
  }
  return []int{idx3, idx4}
}

func getStringFromBounds(str string, bounds []int) string {
  start, end := bounds[0], bounds[1]
  if end == math.MaxInt32 {
    return ""
  }
  return str[start : end+1]
}

func increaseCharCount(char byte, charCounts map[byte]int) {
  charCounts[char]++
}

func decreaseCharCount(char byte, charCounts map[byte]int) {
  charCounts[char]--
}

```