Solution 1

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

#include <climits>
#include <unordered_map>
#include <vector>

using namespace std;

string smallestSubstringContaining(string bigString, string smallString);
unordered_map<char, int> getCharCounts(string str);
vector<int> getSubstringBounds(string str,
                               unordered_map<char, int> targetCharCounts);
vector<int> getCloserBounds(int idx1, int idx2, int idx3, int idx4);
string getStringFromBounds(string str, vector<int> bounds);
void increaseCharCount(char c, unordered_map<char, int> &charCounts);
void decreaseCharCount(char c, unordered_map<char, int> &charCounts);

// 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
string smallestSubstringContaining(string bigString, string smallString) {
  unordered_map<char, int> targetCharCounts = getCharCounts(smallString);
  vector<int> substringBounds = getSubstringBounds(bigString, targetCharCounts);
  return getStringFromBounds(bigString, substringBounds);
}

unordered_map<char, int> getCharCounts(string str) {
  unordered_map<char, int> charCounts;
  for (auto c : str) {
    increaseCharCount(c, charCounts);
  }
  return charCounts;
}

vector<int> getSubstringBounds(string str,
                               unordered_map<char, int> targetCharCounts) {
  vector<int> substringBounds = {0, INT_MAX};
  unordered_map<char, int> substringCharCounts;
  int numUniqueChars = targetCharCounts.size();
  int numUniqueCharsDone = 0;
  int leftIdx = 0;
  int rightIdx = 0;
  // Move the rightIdx to the right in the string until you've counted
  // all of the target characters enough times.
  while (rightIdx < str.size()) {
    char rightChar = str[rightIdx];
    if (targetCharCounts.find(rightChar) == targetCharCounts.end()) {
      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]);
      char leftChar = str[leftIdx];
      if (targetCharCounts.find(leftChar) == targetCharCounts.end()) {
        leftIdx++;
        continue;
      }
      if (substringCharCounts[leftChar] == targetCharCounts[leftChar]) {
        numUniqueCharsDone--;
      }
      decreaseCharCount(leftChar, substringCharCounts);
      leftIdx++;
    }
    rightIdx++;
  }
  return substringBounds;
}

vector<int> getCloserBounds(int idx1, int idx2, int idx3, int idx4) {
  return idx2 - idx1 < idx4 - idx3 ? vector<int>{idx1, idx2}
                                   : vector<int>{idx3, idx4};
}

string getStringFromBounds(string str, vector<int> bounds) {
  int start = bounds[0];
  int end = bounds[1];
  if (end == INT_MAX)
    return "";
  return str.substr(start, end - start + 1);
}

void increaseCharCount(char c, unordered_map<char, int> &charCounts) {
  if (charCounts.find(c) == charCounts.end()) {
    charCounts[c] = 1;
  } else {
    charCounts[c]++;
  }
}

void decreaseCharCount(char c, unordered_map<char, int> &charCounts) {
  charCounts[c]--;
}
```