Solution 1

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

#include <vector>
#include <algorithm>
#include <unordered_map>

using namespace std;

struct stringChain {
  string nextString;
  int maxChainLength;
};

void findLongestStringChain(string str,
                            unordered_map<string, stringChain> &stringChains);
string getSmallerString(string str, int index);
void tryUpdateLongestStringChain(
    string currentString, string smallerString,
    unordered_map<string, stringChain> &stringChains);
vector<string>
buildLongestStringChain(vector<string> strings,
                        unordered_map<string, stringChain> stringChains);

// O(n * m^2 + nlog(n)) time | O(nm) space - where n is the number of strings
// and m is the length of the longest string
vector<string> longestStringChain(vector<string> strings) {
  // For every string, imagine the longest string chain that starts with it.
  // Set up every string to point to the next string in its respective longest
  // string chain. Also keep track of the lengths of these longest string
  // chains.
  unordered_map<string, stringChain> stringChains = {};
  for (auto string : strings) {
    stringChains[string] = {"", 1};
  }

  // Sort the strings based on their length so that whenever we visit a
  // string (as we iterate through them from left to right), we can
  // already have computed the longest string chains of any smaller strings.
  vector<string> sortedStrings = strings;
  sort(sortedStrings.begin(), sortedStrings.end(),
       [](string a, string b) -> bool { return a.size() < b.size(); });

  for (auto string : sortedStrings) {
    findLongestStringChain(string, stringChains);
  }

  return buildLongestStringChain(strings, stringChains);
}

void findLongestStringChain(string str,
                            unordered_map<string, stringChain> &stringChains) {
  // Try removing every letter of the current string to see if the
  // remaining strings form a string chain.
  for (int i = 0; i < str.size(); i++) {
    string smallerString = getSmallerString(str, i);
    if (stringChains.find(smallerString) == stringChains.end())
      continue;
    tryUpdateLongestStringChain(str, smallerString, stringChains);
  }
}

string getSmallerString(string str, int index) {
  return str.substr(0, index) + str.substr(index + 1);
}

void tryUpdateLongestStringChain(
    string currentString, string smallerString,
    unordered_map<string, stringChain> &stringChains) {
  int smallerStringChainLength = stringChains[smallerString].maxChainLength;
  int currentStringChainLength = stringChains[currentString].maxChainLength;
  // Update the string chain of the current string only if the smaller string
  // leads to a longer string chain.
  if (smallerStringChainLength + 1 > currentStringChainLength) {
    stringChains[currentString].maxChainLength = smallerStringChainLength + 1;
    stringChains[currentString].nextString = smallerString;
  }
}

vector<string>
buildLongestStringChain(vector<string> strings,
                        unordered_map<string, stringChain> stringChains) {
  // Find the string that starts the longest string chain.
  int maxChainLength = 0;
  string chainStartingString = "";
  for (auto string : strings) {
    if (stringChains[string].maxChainLength > maxChainLength) {
      maxChainLength = stringChains[string].maxChainLength;
      chainStartingString = string;
    }
  }

  // Starting at the string found above, build the longest string chain.
  vector<string> ourLongestStringChain;
  string currentString = chainStartingString;
  while (currentString != "") {
    ourLongestStringChain.push_back(currentString);
    currentString = stringChains[currentString].nextString;
  }

  return ourLongestStringChain.size() == 1 ? vector<string>{}
                                           : ourLongestStringChain;
}
```