Solution 1

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

class Program {
    class Chain {
        var nextString: String
        var maxChainLength: Int

        init(_ nextString: String, _ maxChainLength: Int) {
            self.nextString = nextString
            self.maxChainLength = maxChainLength
        }
    }

    // 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
    func longestStringChain(_ strings: [String]) -> [String] {
        // 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.
        var stringChains = [String: Chain]()
        for str in strings {
            stringChains[str] = Chain("", 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.
        let sortedStrings = strings.sorted {
            $0.length < $1.length
        }

        for str in sortedStrings {
            findLongestStringChain(str, &stringChains)
        }
        return buildLongestStringChain(strings, &stringChains)
    }

    func findLongestStringChain(_ string: String, _ stringChains: inout [String: Chain]) {
        // Try removing every letter of the current string to see if the
        // remaining strings form a string chain.
        for i in 0 ..< string.length {
            let smallerString = getSmallerString(string, i)
            if let _ = stringChains[smallerString] {
                tryUpdateLongestStringChain(string, smallerString, &stringChains)
            }
        }
    }

    func getSmallerString(_ string: String, _ index: Int) -> String {
        var s = string
        let i = s.index(s.startIndex, offsetBy: index)
        s.remove(at: i)
        return s
    }

    func tryUpdateLongestStringChain(_ currentString: String, _ smallerString: String, _ stringChains: inout [String: Chain]) {
        let smallerStringChainLength = stringChains[smallerString]!.maxChainLength
        let 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
        }
    }

    func buildLongestStringChain(_ strings: [String], _ stringChains: inout [String: Chain]) -> [String] {
        // Find the string that starts the longest string chain.
        var maxChainLength = 0
        var chainStartingString = ""
        for str in strings {
            if stringChains[str]!.maxChainLength > maxChainLength {
                maxChainLength = stringChains[str]!.maxChainLength
                chainStartingString = str
            }
        }

        // Starting at the string found above, build the longest string chain.
        var ourLongestStringChain = [String]()
        var currentString = chainStartingString
        while currentString != "" {
            ourLongestStringChain.append(currentString)
            currentString = stringChains[currentString]!.nextString
        }

        if ourLongestStringChain.count == 1 {
            return [String]()
        }
        return ourLongestStringChain
    }
}

```