Suffix_Array_and_LCP_Array.cpp

```cpp
/**
*    author:  devesh95
*
**/
#include <bits/stdc++.h>
using namespace std;

#define int             long long int
#define double          long double
#define F               first
#define S               second
#define pb              push_back
#define lb              lower_bound
#define ub              upper_bound
#define si              set <int>
#define vi              vector <int>
#define vvi             vector <vi>
#define pii             pair <int, int>
#define vpi             vector <pii>
#define mii             map <int, int>
#define sz(v)           ((int) v.size())
#define form(i, a, b)   for (int i=a; i<(b); i++)
#define forn(i, a)      for (int i=0; i<(a); i++)

////////////////////////////////////////////////////////
// Suffix Array and LCP Array - Detailed Explanation
//
// Purpose: Efficiently handle string processing tasks such as
//          substring search, pattern matching, and finding
//          the longest repeated substring.
//
// ------------------------------------------------------
// Problem Statement:
// Given a string `s`, construct the suffix array and the LCP
// (Longest Common Prefix) array.
//
// ------------------------------------------------------
// Steps:
// 1. Build the Suffix Array:
//    - A suffix array is an array of integers giving the starting positions
//      of suffixes of a string in lexicographical order.
//    - For example, for the string "banana", the suffix array is [5, 3, 1, 0, 4, 2].
//      This means the suffixes starting at indices 5, 3, 1, 0, 4, and 2 are in
//      lexicographical order.
//
//    - To build the suffix array:
//      a) Initialize the suffix array with the indices of the string.
//      b) Sort the suffixes based on the first character.
//      c) Iteratively sort the suffixes based on the first 2^k characters.
//         This is done by comparing pairs of ranks and updating the ranks
//         after each iteration.
```

```
//
// 2. Build the LCP Array using Kasai's Algorithm:
//    - The LCP array stores the lengths of the longest common prefixes
//      between consecutive suffixes in the suffix array.
//    - For example, for the string "banana" with suffix array [5, 3, 1, 0, 4, 2],
//      the LCP array is [0, 1, 3, 0, 0, 2].
//
//    - To build the LCP array:
//      a) Initialize the rank array, which stores the rank of each suffix.
//      b) Iterate through the string and compute the LCP values by comparing
//         characters of the suffixes and using previously computed LCP values
//         to optimize the process.
//
// ---------------------------------------------------------
// Applications:
// - Pattern matching in strings: Quickly find occurrences of a pattern.
//   Example: Using binary search on the suffix array to find the pattern.
//
// - Finding the longest repeated substring: Identify repeated sequences.
//   Example: The maximum value in the LCP array gives the length of the longest
//   repeated substring.
//
// - Solving various string-related problems efficiently:
//   Example: Finding the number of distinct substrings, finding the lexicographical
//   order of substrings, etc.
//
// Tips:
// i) Suffix array construction can be done in O(n log n) time using a combination
//    of sorting and rank updating techniques.
//
// ii) LCP array construction using Kasai's algorithm is O(n) and leverages the
//     previously computed suffix array and rank array to efficiently compute
//     the LCP values.
////////////////////////////////////////////////////////////

// Function to build the suffix array
vector<int> buildSuffixArray(const string &s) {
    int n = s.size();
    vector<int> suffixArray(n), rank(n), temp(n);

    // Initialize suffix array and rank array
    for (int i = 0; i < n; ++i) {
        suffixArray[i] = i;
        rank[i] = s[i];
    }

    // Sort suffixes based on first 2^k characters
    for (int k = 1; k < n; k *= 2) {
        auto cmp = [&](int a, int b) {
            if (rank[a] != rank[b])
                return rank[a] < rank[b];
            int ra = (a + k < n) ? rank[a + k] : -1;
            int rb = (b + k < n) ? rank[b + k] : -1;
            return ra < rb;
```

```
        };
        sort(suffixArray.begin(), suffixArray.end(), cmp);

        // Update rank array based on sorted suffixes
        temp[suffixArray[0]] = 0;
        for (int i = 1; i < n; ++i) {
            temp[suffixArray[i]] = temp[suffixArray[i - 1]] + cmp(suffixArray[i - 1], suffixArray[i]);
        }
        rank = temp;
    }

    return suffixArray;
}

// Function to build the LCP array using Kasai's algorithm
vector<int> buildLCPArray(const string &s, const vector<int> &suffixArray) {
    int n = s.size();
    vector<int> rank(n), lcp(n);

    // Build rank array from suffix array
    for (int i = 0; i < n; ++i) {
        rank[suffixArray[i]] = i;
    }

    int h = 0;
    // Build LCP array
    for (int i = 0; i < n; ++i) {
        if (rank[i] > 0) {
            int j = suffixArray[rank[i] - 1];
            while (i + h < n && j + h < n && s[i + h] == s[j + h]) {
                ++h;
            }
            lcp[rank[i]] = h;
            if (h > 0) --h;
        }
    }

    return lcp;
}

void solve() {
    string s;
    cin >> s;
    int n = s.size();

    // Build suffix array and LCP array
    vector<int> suffixArray = buildSuffixArray(s);
    vector<int> lcpArray = buildLCPArray(s, suffixArray);

    cout << n << ' ';
    // Output the suffix array
    for (int i = 0; i < n; ++i) {
        cout << suffixArray[i] << " ";
```

```cpp
    }
    cout << endl;

    // Output the LCP array
    for (int i = 0; i < n; ++i) {
        cout << lcpArray[i] << " ";
    }
    cout << endl;
}

int32_t main() {
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    clock_t z = clock();
    int t = 1;
    //cin >> t;
    while (t--) {
        solve();
    }
    cerr << "Run Time : " << ((double)(clock() - z) / CLOCKS_PER_SEC);
    return 0;
}
```