LCP and Suffix Array Implementation

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

void countSort(vector<int> &p, vector<int> &c) {
    int n = p.size();
    vector<int> cnt(n);
    for (auto x : c) {
        cnt[x]++;
    }
    vector<int> pos(n);
    pos[0] = 0;
    for (int x = 1; x < n; x++) {
        pos[x] = pos[x - 1] + cnt[x - 1];
    }
    vector<int> p_new(n);
    for (auto x : p) {
        p_new[pos[c[x]]] = x;
        pos[c[x]]++;
    }
    p = p_new;
}

pair<vector<int>, vector<int>> computeSuffixArray(string s) {
    s += "$";
    int n = s.size();
    vector<int> pos(n), class_(n);
    {
        vector<pair<char, int>> a(n);
        for (int i = 0; i < n; i++) {
            a[i] = {s[i], i};
        }
        sort(a.begin(), a.end());
        for (int i = 0; i < n; i++) {
            pos[i] = a[i].second;
        }
        class_[pos[0]] = 0;
        for (int i = 1; i < n; i++) {
            class_[pos[i]] = class_[pos[i - 1]] + (a[i -
               1].first != a[i].first);
        }
    }
    int k = 0;
```

```
    while ((1 << k) < n && class_[pos[n - 1]] < n - 1) {
        for (int i = 0; i < n; i++) {
            pos[i] = (pos[i] - (1 << k) + n) % n;
        }
        countSort(pos, class_);
        vector<int> class_new(n);
        class_new[pos[0]] = 0;
        for (int i = 1; i < n; i++) {
            pair<int, int> prev = {class_[pos[i - 1]],
               class_[(pos[i - 1] + (1 << k)) % n]};
            pair<int, int> curr = {class_[pos[i]],
               class_[(pos[i] + (1 << k)) % n]};
            class_new[pos[i]] = class_new[pos[i - 1]] + (prev
               != curr);
        }
        class_ = class_new;
        k++;
    }
    return {pos, class_};
}

vector<int> computeLCP(const vector<int> &p, const vector<int>
   &c, const string &s) {
    int n = p.size();
    int k = 0;
    vector<int> lcp(n - 1);
    for (int i = 0; i < n - 1; i++) {
        int pi = c[i];
        int j = p[pi - 1];
        while (s[i + k] == s[j + k]) {
            k++;
        }
        lcp[pi - 1] = k;
        k = max(k - 1, 0LL);
    }
    return lcp;
}
```