```cpp
#include "../bits/stdc++.h"
// verified: http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=3384713
// lcp verified: https://atcoder.jp/contests/kupc2016/submissions/4613878
class SuffixArray
{
    int n;
    std::string str;
    std::vector<int> sa, rank;

    template <typename Compare>
    int binarySearch(const std::string &t)
    {
        int m = t.size();
        int lb = -1, ub = n + 1;
        while (ub - lb > 1)
        {
            int mid = (ub + lb) / 2;
            if (Compare()(strncmp(str.c_str() + sa[mid], t.c_str(), m), 0))
            {
                lb = mid;
            }
            else
            {
                ub = mid;
            }
        }
        return ub;
    }

  public:
    // 構築 O(n (logn)^2)
    SuffixArray(const std::string _s) : n(_s.size()), str(_s), sa(n + 1), rank(n + 1)
    {
        for (int i = 0; i <= n; i++)
        {
            sa[i] = i;
            rank[i] = i < n ? str[i] : -1;
        }
        std::vector<int> tmp(n + 1);
        for (int k = 1; k <= n; k *= 2)
        {
            // (rank[i], rank[i+k]), (rank[j], rank[j+k]) を比較
            auto compare_sa = [=](int i, int j) {
                if (rank[i] != rank[j])
                    return rank[i] < rank[j];
                else
                {
                    int ri = i + k <= n ? rank[i + k] : -1;
                    int rj = j + k <= n ? rank[j + k] : -1;
                    return ri < rj;
                }
            };
            std::sort(sa.begin(), sa.end(), compare_sa);
            tmp[sa[0]] = 0;
            for (int i = 1; i <= n; i++)
            {
                tmp[sa[i]] = tmp[sa[i - 1]] + (compare_sa(sa[i - 1], sa[i]) ? 1 : 0);
            }
            for (int i = 0; i <= n; i++)
            {
                rank[i] = tmp[i];
            }
        }
    }
    // O(|t|logn)
    bool contain(const std::string &t)
    {
        int lb = 0, ub = n;
        while (ub - lb > 1)
        {
            int mid = (lb + ub) / 2;
            if (str.compare(sa[mid], t.size(), t) < 0)
                lb = mid;
            else
                ub = mid;
        }
        return str.compare(sa[ub], t.size(), t) == 0;
    }
    // 高さ配列 := 接尾辞配列における隣接要素で先頭何文字が共通しているか
    // つまり, lcp[i] = str[sa[i]..] と str[sa[i+1]..] の先頭共通文字数
    // O(n)
    std::vector<int> getLcp()
    {
        for (int i = 0; i <= n; i++)
            rank[sa[i]] = i;
        int h = 0;
        std::vector<int> lcp(n + 1);
        for (int i = 0; i < n; i++)
        {
            int j = sa[rank[i] - 1];
            if (h > 0)
                h--;
            for (; j + h < n && i + h < n; h++)
            {
                if (str[j + h] != str[i + h])
```

```
                    break;
            }
            lcp[rank[i] - 1] = h;
        }
        return lcp;
    }
    std::vector<int> getSa()
    {
        return sa;
    }
    int lowerBound(const std::string &t)
    {
        return binarySearch<std::less<int>>(t);
    }
    int upperBound(const std::string &t)
    {
        return binarySearch<std::less_equal<int>>(t);
    }
};

// 高さ配列 + セグ木 で任意の接尾辞同士で先頭共通文字数を求められる
// インデックス i,j の先頭共通文字数は rank[i]<rank[j] ならば
// min(lcp[rank[i]], lcp[rank[i]+1], ... , lcp[rank[j]-1]) に一致

```