icpc International Collegiate Programming Contest

icpc.foundation

Standard Code Library

Part3 - String

Jiangxi Normal University

HeartFireY, eroengine, yezzz

November 7, 2022

Standard Code Library

HeartFireY, eroengine, yezzz

Jiangxi Normal University

November 7, 2022

Contents

Section.5 字符串

后缀自动机

- 广义后缀自动机如果直接使用以下代码的话会产生一些冗余状态（置 last 为 1），所以要用拓扑排序。用 len 基数排序不能。
- 字符集大的话要使用 ***map***。
- 树上 dp 时注意边界（root 和 null）。
- rsort 中的数组 a 是拓扑序 [1, sz)

```
struct SAM{
    int ch[N << 1][26], fa[N << 1], len[N << 1], vis[N << 1];
    int last, tot;
    SAM(): last(1), tot(1) {}
    inline void extend(int x){ //* 单字符扩展
        int p = last, np = last = ++tot;
        len[np] = len[p] + 1, vis[np] = 1;
        for(; p && !ch[p][x]; p = fa[p]) ch[p][x] = np;
        if(!p) fa[np] = 1;
        else{
            int q = ch[p][x];
            if(len[q] == len[p] + 1) fa[np] = q;
            else {
                int nq = ++tot;
                for(int i = 0; i < 26; i++) ch[nq][i] = ch[q][i]; //for(int i = 0; i < 26; i++) ch[nq][i] = ch[q][i];
                fa[nq] = fa[q], fa[np] = fa[q] = nq, len[nq] = len[p] + 1;
                for(; ch[p][x] == q; p = fa[p]) ch[p][x] = nq;
            }
        }
    }
}sam;
```

- 最长公共子串

```
//* 最长公共子串
string lcs(const string &T) {
    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < T.size(); i++) {
        while (v && !sam.ch[v][T[i] - 'a']) {
            v = sam.fa[v];
            l = sam.len[v];
        }
        if (sam.ch[v][T[i] - 'a']) {
            v = sam.ch[v][T[i] - 'a'];
            l++;
        }
        if (l > best) {
            best = l;
            bestpos = i;
        }
    }
    return T.substr(bestpos - best + 1, best);
}
```

- 真·广义后缀自动机

```
int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
LL cnt[M][2];
void ins(int ch, int id) {
    int p = last, np = 0, nq = 0, q = -1;
    if (!t[p][ch]) {
        np = sz++;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    }
    if (!p) fa[np] = 1;
    else {
        q = t[p][ch];
        if (len[p] + 1 == len[q]) fa[np] = q;
        else {
            nq = sz++; len[nq] = len[p] + 1;
            memcpy(t[nq], t[q], sizeof t[0]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }
    last = np ? np : nq ? nq : q;
    cnt[last][id] = 1;
}
```

- 按字典序建立后缀树注意逆序插入
- rsort2 里的 a 不是拓扑序，需要拓扑序就去树上做

```
void ins(int ch, int pp) {
    int p = last, np = last = sz++;
    len[np] = len[p] + 1; one[np] = pos[np] = pp;
    for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    if (!p) { fa[np] = 1; return; }
    int q = t[p][ch];
    if (len[q] == len[p] + 1) fa[np] = q;
    else {
        int nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
        memcpy(t[nq], t[q], sizeof t[0]);
        fa[nq] = fa[q];
        fa[q] = fa[np] = nq;
        for (; p && t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
    }
}

int up[M], c[256] = {2}, a[M];
void rsort2() {
    FOR (i, 1, 256) c[i] = 0;
    FOR (i, 2, sz) up[i] = s[one[i] + len[fa[i]]];
    FOR (i, 2, sz) c[up[i]]++;
    FOR (i, 1, 256) c[i] += c[i - 1];
    FOR (i, 2, sz) a[--c[up[i]]] = i;
    FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]);
}
```

- 广义后缀自动机建后缀树，必须反向插入

```
int t[M][26], len[M] = {0}, fa[M], sz = 2, last = 1;
char* one[M];
void ins(int ch, char* pp) {
    int p = last, np = 0, nq = 0, q = -1;
    if (!t[p][ch]) {
        np = sz++; one[np] = pp;
        len[np] = len[p] + 1;
        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
    }
    if (!p) fa[np] = 1;
    else {
        q = t[p][ch];
        if (len[p] + 1 == len[q]) fa[np] = q;
        else {
            nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
            memcpy(t[nq], t[q], sizeof t[0]);
            fa[nq] = fa[q];
            fa[np] = fa[q] = nq;
            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
        }
    }
    last = np ? np : nq ? nq : q;
}
int up[M], c[256] = {2}, aa[M];
vector<int> G[M];
void rsort() {
    FOR (i, 1, 256) c[i] = 0;
    FOR (i, 2, sz) up[i] = *(one[i] + len[fa[i]]);
    FOR (i, 2, sz) c[up[i]]++;
    FOR (i, 1, 256) c[i] += c[i - 1];
    FOR (i, 2, sz) aa[--c[up[i]]] = i;
    FOR (i, 2, sz) G[fa[aa[i]]].push_back(aa[i]);
}
```

- 匹配

```
int u = 1, l = 0;
FOR (i, 0, strlen(s)) {
    int ch = s[i] - 'a';
    while (u && !t[u][ch]) { u = fa[u]; l = len[u]; }
    ++l; u = t[u][ch];
    if (!u) u = 1;
    if (l) // do something...
}
```

- 获取子串状态

```
int get_state(int l, int r) {
    int u = rpos[r], s = r - l + 1;
    FORD (i, SP - 1, -1) if (len[pa[u][i]] >= s) u = pa[u][i];
    return u;
}
```

- 维护区间本质不同字串数目
  - 给你一个长度为 n 的字符串 s，m 次询问，第 i 次询问 s 上的一个区间 $[l_i, r_i]$ 上有多少个本质不同的子串
  - 将每个本质不同的字符串视为一个连续的区间 $[l, r]$，我们只需要维护左端点最后一次出现的位置即可

  因为需要知道每个子串最后一次出现的位置, 所以我们选择对字符串构造后缀自动机, 对于某个右端点 $ r SAM [1 , r] $parent$ r pre[i] $ 为节点 i 上一次出现时的右端点位置，我们只需要暴跳 $father$，每次将之前出现过的位置，在线段树上区间更新成 $-1 [1 , r] , [2 , r] ... [r , r] r [1 , r] $ 在线段树上 $+1 $ 就好了

  不过问题是，这样暴跳 $ father $ 的时间复杂度是不正确的，考虑优化，因为每次选择一条链，自下而上去更新，其本质就是:

  1. 令这条链每个节点相应的位置在线段树上区间更新
  2. 令每个节点都被端点 $ r pre r$）

  这个过程其实就是 $ LCT access logn splay log^2n nlog^2n $ 了

```
#include <bits/stdc++.h>
#pragma gcc optimize("O2")
#pragma g++ optimize("O2")
#define int long long
#define endl '\n'
using namespace std;

const int N = 2e5 + 10, MOD = 1e9 + 7;

int n = 0;

namespace DS{
    namespace SAM{
        int ch[N << 1][26], fa[N << 1], len[N << 1], vis[N << 1], pos[N << 1];
        int last, tot;
        inline void extend(int x){
            int p = last, np = last = ++tot;
            len[np] = len[p] + 1, vis[np] = 1;
            for(; p && !ch[p][x]; p = fa[p]) ch[p][x] = np;
            if(!p) fa[np] = 1;
            else{
                int q = ch[p][x];
                if(len[q] == len[p] + 1) fa[np] = q;
                else {
                    int nq = ++tot;
                    for(int i = 0; i < 26; i++) ch[nq][i] = ch[q][i];
                    fa[nq] = fa[q], fa[np] = fa[q] = nq, len[nq] = len[p] + 1;
                    for(; ch[p][x] == q; p = fa[p]) ch[p][x] = nq;
                }
            }
        }

        void build(string s) {
```

```
        last = tot = 1;
        int len = s.size();
        s = '@' + s;
        for(int i = 1; i <= len; i++) extend(s[i] - 'a'), pos[i] = last;
    }
}

namespace SegTree{
    #define ls rt << 1
    #define rs rt << 1 | 1
    #define lson ls, l, mid
    #define rson rs, mid + 1, r
    int tree[N << 2], lazy[N << 2];

    inline void push_up(int rt) { tree[rt] = tree[ls] + tree[rs]; }

    inline void push(int rt, int val, int c) { tree[rt] += val * c, lazy[rt] += val; }

    inline void push_down(int rt, int c) {
        if(lazy[rt]) {
            push(ls, lazy[rt], (c - (c >> 1)));
            push(rs, lazy[rt], (c >> 1));
            lazy[rt] = 0;
        }
    }

    void build(int rt, int l, int r){
        tree[rt] = lazy[rt] = 0;
        if(l == r) return;
        int mid = l + r >> 1;
        build(lson), build(rson);
    }

    void update(int rt, int l, int r, int L, int R, int val) {
        if(l >= L && r <= R) return push(rt, val, r - l + 1);
        push_down(rt, r - l + 1);
        int mid = l + r >> 1;
        if(mid >= L) update(lson, L, R, val);
        if(mid < R) update(rson, L, R, val);
        push_up(rt);
    }

    int query(int rt, int l, int r, int L, int R) {
        if(l >= L && r <= R) return tree[rt];
        push_down(rt, r - l + 1);
        int mid = l + r >> 1, sum = 0;
        if(mid >= L) sum += query(lson, L, R);
        if(mid < R) sum += query(rson, L, R);
        return sum;
    }
    #undef ls
    #undef rs
    #undef lson
    #undef rson
}

namespace LCT{
    #define ls ch[x][0]
    #define rs ch[x][1]

    struct Info{
        int len, minn, pre, tag_chg;
    }tree[N];

    int ch[N][2], f[N], tag[N];

    inline void push_up(int x) { tree[x].minn = min({tree[x].len, tree[ls].minn, tree[rs].minn}); }

    inline void push(int x) { swap(ls, rs), tag[x] ^= 1; }

    inline void push_chg(int x, int v) { tree[x].pre = tree[x].tag_chg = v; }
```

```

        inline void push_down(int x) {
            if(tag[x]) {
                if(ls) push(ls);
                if(rs) push(rs);
                tag[x] = 0;
            }
            if(tree[x].tag_chg) {
                if(ls) push_chg(ls, tree[x].tag_chg);
                if(rs) push_chg(rs, tree[x].tag_chg);
                tree[x].tag_chg = 0;
            }
        }

        #define get(x) (ch[f[x]][1] == x)
        #define isRoot(x) (ch[f[x]][0] != x && ch[f[x]][1] != x)

        inline void rotate(int x) {
            int y = f[x], z = f[y], k = get(x);
            if(!isRoot(y)) ch[z][ch[z][1] == y] = x;
            ch[y][k] = ch[x][!k], f[ch[x][!k]] = y;
            ch[x][!k] = y, f[y] = x, f[x] = z;
            push_up(y); push_up(x);
        }

        inline void update(int x) {
            if(!isRoot(x)) update(f[x]);
            push_down(x);
        }

        inline void splay(int x) {
            update(x);
            for(int fa = f[x]; !isRoot(x); rotate(x), fa = f[x]){
                if(!isRoot(fa)) rotate(get(fa) == get(x) ? fa : x);
            }
            push_up(x);
        }

        int access(int x, int pos) {
            int p;
            for(p = 0; x; x = f[p = x]){
                splay(x), ch[x][1] = p, push_up(x);
                if(tree[x].pre) {
                    int upl = tree[x].pre - SAM::len[x] + 1;
                    int upr = tree[x].pre - tree[x].minn + 1;
                    // cout << "LCT Operation SegTree -> Part(" << upl << ", " << upr << "), add value -1\n";
                    SegTree::update(1, 1, n, upl, upr, -1);
                }
            }
            splay(p);
            push_chg(p, pos);
            SegTree::update(1, 1, n, 1, pos, 1);
            // cout << endl;
            return p;
        }

        void build() {
            tree[0].minn = 1e18;
            for(int i = 1; i <= SAM::tot; i++) {
                f[i] = SAM::fa[i];
                tree[i].len = tree[i].minn = SAM::len[SAM::fa[i]] + 1;
                tree[i].pre = tree[i].tag_chg = ch[i][0] = ch[i][1] = 0;
            }
        }
        #undef ls
        #undef rs
    }
}

struct query{ int l, id; };
vector<query> qr[N];
```

```
int ans[N];

inline void solve(){
    string s; cin >> s, n = s.size();
    DS::SAM::build(s);
    DS::SegTree::build(1, 1, n);
    DS::LCT::build();
    int m = 0; cin >> m;
    for(int i = 1; i <= m; i++) {
        int l, r; cin >> l >> r;
        qr[r].emplace_back(query{l, i});
    }
    for(int i = 1; i <= n; i++) {
        DS::LCT::access(DS::SAM::pos[i], i);
        for(auto &[l, id] : qr[i]) ans[id] = DS::SegTree::query(1, 1, n, l, i);
    }
    for(int i = 1; i <= m; i++) cout << ans[i] << endl;
}

signed main(){
    ios_base::sync_with_stdio(false), cin.tie(0);
    cout << fixed << setprecision(12);
    int t = 1; // cin >> t;
    while(t--) solve();
    return 0;
}
```

回文自动机

- num 是该结点表示的前缀的回文后缀个数
- cnt 是该结点表示的回文串在原串中的出现次数（使用前需要向父亲更新）

```
namespace pam {
    int t[N][26], fa[N], len[N], rs[N], cnt[N], num[N];
    int sz, n, last;
    int _new(int l) {
        len[sz] = l; cnt[sz] = num[sz] = 0;
        return sz++;
    }
    void init() {
        memset(t, 0, sz * sizeof t[0]);
        rs[n = sz = 0] = -1;
        last = _new(0);
        fa[last] = _new(-1);
    }
    int get_fa(int x) {
        while (rs[n - 1 - len[x]] != rs[n]) x = fa[x];
        return x;
    }
    void ins(int ch) {
        rs[++n] = ch;
        int p = get_fa(last);
        if (!t[p][ch]) {
            int np = _new(len[p] + 2);
            num[np] = num[fa[np] = t[get_fa(fa[p])][ch]] + 1;
            t[p][ch] = np;
        }
        ++cnt[last = t[p][ch]];
    }
}
```

Manacher

$P[i]$ 的计算方法：假设 j 是 i 关于 C 的镜像点，$P[j]$ 已经求解完毕。

- 若 $i \geq R$，由于 R 右侧的字符都没有检查过，因此只能初始化 $P[i] = 1$ 然后暴力中心扩展
- 若 $i < R$，分两种情况：
    1. j 的回文串被 C 的回文串包含，即 j 的回文串左端点比 C 回文串的左端点大，按照镜像原理，镜像 i 的回文不会超过 C 的右端点 R，因此根据 $(i+j)/2 = C$ 得 $j = 2C - i$，故 $P[i] = P[j] = P[2C - i]$。然后继续用暴力中心扩展法完成

$P[i]$ 的计算。

2. j 的回文串不被 C 的回文串包含，即 j 的回文串左端点比 C 回文串的左端点小。此时 i 回文串的右端点比 R 大，但是由于 R 右边的字符还没有检查过，只能先让 $P[i]$ 被限制在 R 之内，有 $P[i] = w = R - i = C + P[i] - i$，然后继续用暴力中心扩展法完成 $P[i]$ 的计算。

```cpp
int n, p[N << 1];       // p[i]: 以 s[i] 为中心的回文串半径
char a[N], s[N << 1];   // a 为原始串, s 为修改后的串

void change() {
    n = strlen(a);
    int k = 0; s[k++] = '$', s[k++] = '#';
    for(int i = 0; i < n; i++) s[k++] = a[i], s[k++] = '#';
    s[k++] = '&', n = k;
}

void manacher() {
    int R = 0, C = 0;
    for(int i = 1; i < n; i++) {
        if(i < R) p[i] = min(p[(C << 1) - i], p[C] + C - i);    // 1. 合并处理两种情况
        else p[i] = 1;                                          //
        while(s[i + p[i]] == s[i - p[i]]) p[i]++;               // 2. 暴力中心扩展
        if(p[i] + i > R) R = p[i] + i, C = i;                   // 3. 更新最大的 R
    }
}

inline void solve() {
    cin >> a;
    change(), manacher();
    int ans = 1;
    for(int i = 0; i < n; i++) ans = max(ans, p[i]);
    cout << ans - 1 << endl;
}
```

哈希

内置了自动双哈希开关（小心 TLE）。

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

#define ENABLE_DOUBLE_HASH

typedef long long LL;
typedef unsigned long long ULL;

const int x = 135;
const int N = 4e5 + 10;
const int p1 = 1e9 + 7, p2 = 1e9 + 9;
ULL xp1[N], xp2[N], xp[N];

void init_xp() {
    xp1[0] = xp2[0] = xp[0] = 1;
    for (int i = 1; i < N; ++i) {
        xp1[i] = xp1[i - 1] * x % p1;
        xp2[i] = xp2[i - 1] * x % p2;
        xp[i] = xp[i - 1] * x;
    }
}

struct String {
    char s[N];
    int length, subsize;
    bool sorted;
    ULL h[N], hl[N];

    ULL hash() {
        length = strlen(s);
        ULL res1 = 0, res2 = 0;
        h[length] = 0;  // ATTENTION!
        for (int j = length - 1; j >= 0; --j) {
```

```
        #ifdef ENABLE_DOUBLE_HASH
            res1 = (res1 * x + s[j]) % p1;
            res2 = (res2 * x + s[j]) % p2;
            h[j] = (res1 << 32) | res2;
        #else
            res1 = res1 * x + s[j];
            h[j] = res1;
        #endif
            // printf("%llu\n", h[j]);
        }
        return h[0];
    }

    // 获取子串哈希，左闭右开区间
    ULL get_substring_hash(int left, int right) const {
        int len = right - left;
    #ifdef ENABLE_DOUBLE_HASH
        // get hash of s[left...right-1]
        unsigned int mask32 = ~(0u);
        ULL left1 = h[left] >> 32, right1 = h[right] >> 32;
        ULL left2 = h[left] & mask32, right2 = h[right] & mask32;
        return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
               (((left2 - right2 * xp2[len] % p2 + p2) % p2));
    #else
        return h[left] - h[right] * xp[len];
    #endif
    }

    void get_all_subs_hash(int sublen) {
        subsize = length - sublen + 1;
        for (int i = 0; i < subsize; ++i)
            hl[i] = get_substring_hash(i, i + sublen);
        sorted = 0;
    }

    void sort_substring_hash() {
        sort(hl, hl + subsize);
        sorted = 1;
    }

    bool match(ULL key) const {
        if (!sorted) assert (0);
        if (!subsize) return false;
        return binary_search(hl, hl + subsize, key);
    }

    void init(const char *t) {
        length = strlen(t);
        strcpy(s, t);
    }
};

int LCP(const String &a, const String &b, int ai, int bi) {
    // Find LCP of a[ai...] and b[bi...]
    int l = 0, r = min(a.length - ai, b.length - bi);
    while (l < r) {
        int mid = (l + r + 1) / 2;
        if (a.get_substring_hash(ai, ai + mid) == b.get_substring_hash(bi, bi + mid))
            l = mid;
        else r = mid - 1;
    }
    return l;
}

int check(int ans) {
    if (T.length < ans) return 1;
    T.get_all_subs_hash(ans); T.sort_substring_hash();
    for (int i = 0; i < S.length - ans + 1; ++i)
        if (!T.match(S.get_substring_hash(i, i + ans)))
            return 1;
    return 0;
```

```
}

int main() {
    init_xp();  // DON'T FORGET TO DO THIS!

    for (int tt = 1; tt <= kases; ++tt) {
        scanf("%d", &n); scanf("%s", str);
        S.init(str);
        S.hash(); T.hash();
    }
}
```

二维哈希

```
struct Hash2D { // 1-index
    static const LL px = 131, py = 233, MOD = 998244353;
    static LL pwx[N], pwy[N];
    int a[N][N];
    LL hv[N][N];
    static void init_xp() {
        pwx[0] = pwy[0] = 1;
        FOR (i, 1, N) {
            pwx[i] = pwx[i - 1] * px % MOD;
            pwy[i] = pwy[i - 1] * py % MOD;
        }
    }
    void init_hash(int n, int m) {
        FOR (i, 1, n + 1) {
            LL s = 0;
            FOR (j, 1, m + 1) {
                s = (s * py + a[i][j]) % MOD;
                hv[i][j] = (hv[i - 1][j] * px + s) % MOD;
            }
        }
    }
    LL h(int x, int y, int dx, int dy) {
        --x; --y;
        LL ret = hv[x + dx][y + dy] + hv[x][y] * pwx[dx] % MOD * pwy[dy]
                 - hv[x][y + dy] * pwx[dx] - hv[x + dx][y] * pwy[dy];
        return (ret % MOD + MOD) % MOD;
    }
} ha, hb;
LL Hash2D::pwx[N], Hash2D::pwy[N];
```

后缀数组

构造时间：$O(L\log L)$；查询时间 $O(\log L)$。**suffix** 数组是排好序的后缀下标，**suffix** 的反数组是后缀数组。

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

const int N = 2e5 + 10;
const int Nlog = 18;

struct SuffixArray {
    const int L;
    vector<vector<int> > P;
    vector<pair<pair<int, int>, int> > M;
    int s[N], sa[N], rank[N], height[N];
    // s: raw string
    // sa[i]=k: s[k...L-1] ranks i (0 based)
    // rank[i]=k: the rank of s[i...L-1] is k (0 based)
    // height[i] = lcp(sa[i-1], sa[i])

    SuffixArray(const string &raw_s) : L(raw_s.length()), P(1, vector<int>(L, 0)), M(L) {
        for (int i = 0; i < L; i++)
            P[0][i] = this->s[i] = int(raw_s[i]);
        for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
            P.push_back(vector<int>(L, 0));
            for (int i = 0; i < L; i++)
                M[i] = make_pair(make_pair(P[level - 1][i], i + skip < L ? P[level - 1][i + skip] : -1000), i);
            sort(M.begin(), M.end());
            for (int i = 0; i < L; i++)
                P[level][M[i].second] = (i > 0 && M[i].first == M[i - 1].first) ? P[level][M[i - 1].second] : i;
        }
        for (unsigned i = 0; i < P.back().size(); ++i) {
            rank[i] = P.back()[i];
            sa[rank[i]] = i;
        }
    }

    // This is a traditional way to calculate LCP
    void getHeight() {
        memset(height, 0, sizeof height);
        int k = 0;
        for (int i = 0; i < L; ++i) {
            if (rank[i] == 0) continue;
            if (k) k--;
            int j = sa[rank[i] - 1];
            while (i + k < L && j + k < L && s[i + k] == s[j + k]) ++k;
            height[rank[i]] = k;
        }
        rmq_init(height, L);
    }

    int f[N][Nlog];
    inline int highbit(int x) {
        return 31 - __builtin_clz(x);
    }

    int rmq_query(int x, int y) {
        int p = highbit(y - x + 1);
        return min(f[x][p], f[y - (1 << p) + 1][p]);
    }

    // arr has to be 0 based
    void rmq_init(int *arr, int length) {
        for (int x = 0; x <= highbit(length); ++x)
            for (int i = 0; i <= length - (1 << x); ++i) {
                if (!x) f[i][x] = arr[i];
                else f[i][x] = min(f[i][x - 1], f[i + (1 << (x - 1))][x - 1]);
            }
    }

    #ifdef NEW
    // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
    int LongestCommonPrefix(int i, int j) {
        int len = 0;
        if (i == j) return L - i;
        for (int k = (int) P.size() - 1; k >= 0 && i < L && j < L; k--) {
            if (P[k][i] == P[k][j]) {
                i += 1 << k;
                j += 1 << k;
                len += 1 << k;
            }
        }
        return len;
    }
    #else
    int LongestCommonPrefix(int i, int j) {
        // getHeight() must be called first
        if (i == j) return L - i;
        if (i > j) swap(i, j);
        return rmq_query(i + 1, j);
    }
    #endif

    int checkNonOverlappingSubstring(int K) {
        // check if there is two non-overlapping identical substring of length K
        int minsa = 0, maxsa = 0;
        for (int i = 0; i < L; ++i) {
            if (height[i] < K) {
                minsa = sa[i]; maxsa = sa[i];
            } else {
                minsa = min(minsa, sa[i]);
                maxsa = max(maxsa, sa[i]);
                if (maxsa - minsa >= K) return 1;
            }
        }
        return 0;
    }

    int checkBelongToDifferentSubstring(int K, int split) {
        int minsa = 0, maxsa = 0;
        for (int i = 0; i < L; ++i) {
            if (height[i] < K) {
                minsa = sa[i]; maxsa = sa[i];
            } else {
                minsa = min(minsa, sa[i]);
                maxsa = max(maxsa, sa[i]);
                if (maxsa > split && minsa < split) return 1;
            }
        }
        return 0;
    }

} *S;

int main() {
    string s, t;
    cin >> s >> t;
    int sp = s.length();
    s += "*" + t;
    S = new SuffixArray(s);
    S->getHeight();
    int left = 0, right = sp;
    while (left < right) {
        int mid = (left + right + 1) / 2;
        if (S->checkBelongToDifferentSubstring(mid, sp))
            left = mid;
        else right = mid - 1;
    }
    printf("%d\n", left);
}
```

- SA-IS
- 仅在后缀自动机被卡内存或者卡常且需要 O(1) LCA 的情况下使用（比赛中敲这个我觉得不行）
- UOJ 35

```
// rk [0..n-1] -> [1..n], sa/ht [1..n]
// s[i] > 0 && s[n] = 0
// b: normally as bucket
// c: normally as bucket1
// d: normally as bucket2
// f: normally as cntbuf

template<size_t size>
struct SuffixArray {
    bool t[size << 1];
    int b[size], c[size];
    int sa[size], rk[size], ht[size];
    inline bool isLMS(const int i, const bool *t) { return i > 0 && t[i] && !t[i - 1]; }
    template<class T>
    inline void inducedSort(T s, int *sa, const int n, const int M, const int bs,
                            bool *t, int *b, int *f, int *p) {
        fill(b, b + M, 0); fill(sa, sa + n, -1);
        FOR (i, 0, n) b[s[i]]++;
        f[0] = b[0];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i];
        FORD (i, bs - 1, -1) sa[--f[s[p[i]]]] = p[i];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i - 1];
        FOR (i, 0, n) if (sa[i] > 0 && !t[sa[i] - 1]) sa[f[s[sa[i] - 1]]++] = sa[i] - 1;
        f[0] = b[0];
        FOR (i, 1, M) f[i] = f[i - 1] + b[i];
        FORD (i, n - 1, -1) if (sa[i] > 0 && t[sa[i] - 1]) sa[--f[s[sa[i] - 1]]] = sa[i] - 1;
    }
    template<class T>
    inline void sais(T s, int *sa, int n, bool *t, int *b, int *c, int M) {
        int i, j, bs = 0, cnt = 0, p = -1, x, *r = b + M;
        t[n - 1] = 1;
        FORD (i, n - 2, -1) t[i] = s[i] < s[i + 1] || (s[i] == s[i + 1] && t[i + 1]);
        FOR (i, 1, n) if (t[i] && !t[i - 1]) c[bs++] = i;
        inducedSort(s, sa, n, M, bs, t, b, r, c);
        for (i = bs = 0; i < n; i++) if (isLMS(sa[i], t)) sa[bs++] = sa[i];
        FOR (i, bs, n) sa[i] = -1;
        FOR (i, 0, bs) {
            x = sa[i];
            for (j = 0; j < n; j++) {
                if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) { cnt++, p = x; break; }
                else if (j > 0 && (isLMS(x + j, t) || isLMS(p + j, t))) break;
            }
            x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bs + x] = cnt - 1;
        }
        for (i = j = n - 1; i >= bs; i--) if (sa[i] >= 0) sa[j--] = sa[i];
        int *s1 = sa + n - bs, *d = c + bs;
        if (cnt < bs) sais(s1, sa, bs, t + n, b, c + bs, cnt);
        else FOR (i, 0, bs) sa[s1[i]] = i;
        FOR (i, 0, bs) d[i] = c[sa[i]];
        inducedSort(s, sa, n, M, bs, t, b, r, d);
    }
    template<typename T>
    inline void getHeight(T s, const int n, const int *sa) {
        for (int i = 0, k = 0; i < n; i++) {
            if (rk[i] == 0) k = 0;
            else {
                if (k > 0) k--;
                int j = sa[rk[i] - 1];
                while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
            }
            ht[rk[i]] = k;
        }
    }
    template<class T>
    inline void init(T s, int n, int M) {
        sais(s, sa, ++n, t, b, c, M);
        for (int i = 1; i < n; i++) rk[sa[i]] = i;
        getHeight(s, n, sa);
    }
};

const int N = 2E5 + 100;
SuffixArray<N> sa;

int main() {
    string s; cin >> s; int n = s.length();
    sa.init(s, n, 128);
    FOR (i, 1, n + 1) printf("%d%c", sa.sa[i] + 1, i == _i - 1 ? '\n' : ' ');
    FOR (i, 2, n + 1) printf("%d%c", sa.ht[i], i == _i - 1 ? '\n' : ' ');
}
```

KMP

- 前缀函数（每一个前缀的最长 border）

```
void get_pi(int a[], char s[], int n) {
    int j = a[0] = 0;
    FOR (i, 1, n) {
        while (j && s[i] != s[j]) j = a[j - 1];
        a[i] = j += s[i] == s[j];
    }
}
```

- Z 函数（每一个后缀和该字符串的 LCP 长度）

```
void get_z(int a[], char s[], int n) {
    int l = 0, r = 0; a[0] = n;
    FOR (i, 1, n) {
        a[i] = i > r ? 0 : min(r - i + 1, a[i - l]);
        while (i + a[i] < n && s[a[i]] == s[i + a[i]]) ++a[i];
        if (i + a[i] - 1 > r) { l = i; r = i + a[i] - 1; }
    }
}
```

Trie

```
namespace trie {
    int t[N][26], sz, ed[N];
    void init() { sz = 2; memset(ed, 0, sizeof ed); }
    int _new() { memset(t[sz], 0, sizeof t[sz]); return sz++; }
    void ins(char* s, int p) {
        int u = 1;
        FOR (i, 0, strlen(s)) {
            int c = s[i] - 'a';
            if (!t[u][c]) t[u][c] = _new();
            u = t[u][c];
        }
        ed[u] = p;
    }
}
```

AC 自动机

```
const int N = 1e6 + 100, M = 26;

int mp(char ch) { return ch - 'a'; }

struct ACA {
    int ch[N][M], danger[N], fail[N];
    int sz;
    void init() {
        sz = 1;
        memset(ch[0], 0, sizeof ch[0]);
        memset(danger, 0, sizeof danger);
    }
    void insert(const string &s, int m) {
        int n = s.size(); int u = 0, c;
        FOR (i, 0, n) {
            c = mp(s[i]);
            if (!ch[u][c]) {
                memset(ch[sz], 0, sizeof ch[sz]);
                danger[sz] = 0; ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        danger[u] |= 1 << m;
    }
    void build() {
        queue<int> Q;
        fail[0] = 0;
        for (int c = 0, u; c < M; c++) {
            u = ch[0][c];
            if (u) { Q.push(u); fail[u] = 0; }
        }
        while (!Q.empty()) {
            int r = Q.front(); Q.pop();
            danger[r] |= danger[fail[r]];
            for (int c = 0, u; c < M; c++) {
                u = ch[r][c];
                if (!u) {
                    ch[r][c] = ch[fail[r]][c];
                    continue;
                }
                fail[u] = ch[fail[r]][c];
                Q.push(u);
            }
        }
    }
} ac;

char s[N];

int main() {
    int n; scanf("%d", &n);
    ac.init();
    while (n--) {
        scanf("%s", s);
        ac.insert(s, 0);
    }
    ac.build();

    scanf("%s", s);
    int u = 0; n = strlen(s);
    FOR (i, 0, n) {
        u = ac.ch[u][mp(s[i])];
        if (ac.danger[u]) {
            puts("YES");
            return 0;
        }
    }
    puts("NO");
    return 0;
}
```