

Standard Code Library

FOREIGNERS

Jiangxi Normal University HeartFireY

September 1202

Contents

Section.5 字符串 2

 后缀自动机 2

 回文自动机 5

 manacher 5

 哈希 6

 后缀数组 8

 KMP 11

 Trie 11

 AC 自动机 11

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)

```
1 struct SAM{
2     int ch[N << 1][26], fa[N << 1], len[N << 1], vis[N << 1];
3     int last, tot;
4     SAM(): last(1), tot(1) {}
5     inline void extend(int x){ /* 单字符扩展
6         int p = last, np = last = ++tot;
7         len[np] = len[p] + 1, vis[np] = 1;
8         for(; p && !ch[p][x]; p = fa[p]) ch[p][x] = np;
9         if(!p) fa[np] = 1;
10        else{
11            int q = ch[p][x];
12            if(len[q] == len[p] + 1) fa[np] = q;
13            else {
14                int nq = ++tot;
15                for(int i = 0; i < 26; i++) ch[nq][i] = ch[q][i];
16                fa[nq] = fa[q], fa[np] = fa[q] = nq, len[nq] = len[p] + 1;
17                for(; ch[p][x] == q; p = fa[p]) ch[p][x] = nq;
18            }
19        }
20    }
21 }sam;
```

- 真·广义后缀自动机

```
1 int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
2 LL cnt[M][2];
3 void ins(int ch, int id) {
4     int p = last, np = 0, nq = 0, q = -1;
5     if (!t[p][ch]) {
6         np = sz++;
7         len[np] = len[p] + 1;
8         for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
9     }
10    if (!p) fa[np] = 1;
11    else {
12        q = t[p][ch];
13        if (len[p] + 1 == len[q]) fa[np] = q;
14        else {
15            nq = sz++; len[nq] = len[p] + 1;
16            memcpy(t[nq], t[q], sizeof t[0]);
17            fa[nq] = fa[q];
18            fa[np] = fa[q] = nq;
19            for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
20        }
21    }
22    last = np ? np : nq ? nq : q;
23    cnt[last][id] = 1;
24 }
```

- 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

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

```

11     fa[nq] = fa[q];
12     fa[q] = fa[np] = nq;
13     for (; p && t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
14 }
15 }
16
17 int up[M], c[256] = {2}, a[M];
18 void rsort2() {
19     FOR (i, 1, 256) c[i] = 0;
20     FOR (i, 2, sz) up[i] = s[one[i] + len[fa[i]]];
21     FOR (i, 2, sz) c[up[i]]++;
22     FOR (i, 1, 256) c[i] += c[i - 1];
23     FOR (i, 2, sz) a[--c[up[i]]] = i;
24     FOR (i, 2, sz) G[fa[a[i]]].push_back(a[i]);
25 }

```

- 广义后缀自动机建后缀树，必须反向插入

```

1  int t[M][26], len[M] = {0}, fa[M], sz = 2, last = 1;
2  char* one[M];
3  void ins(int ch, char* pp) {
4      int p = last, np = 0, nq = 0, q = -1;
5      if (!t[p][ch]) {
6          np = sz++; one[np] = pp;
7          len[np] = len[p] + 1;
8          for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
9      }
10     if (!p) fa[np] = 1;
11     else {
12         q = t[p][ch];
13         if (len[p] + 1 == len[q]) fa[np] = q;
14         else {
15             nq = sz++; len[nq] = len[p] + 1; one[nq] = one[q];
16             memcpy(t[nq], t[q], sizeof t[0]);
17             fa[nq] = fa[q];
18             fa[np] = fa[q] = nq;
19             for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
20         }
21     }
22     last = np ? np : nq ? nq : q;
23 }
24 int up[M], c[256] = {2}, aa[M];
25 vector<int> G[M];
26 void rsort() {
27     FOR (i, 1, 256) c[i] = 0;
28     FOR (i, 2, sz) up[i] = *(one[i] + len[fa[i]]);
29     FOR (i, 2, sz) c[up[i]]++;
30     FOR (i, 1, 256) c[i] += c[i - 1];
31     FOR (i, 2, sz) aa[--c[up[i]]] = i;
32     FOR (i, 2, sz) G[fa[aa[i]]].push_back(aa[i]);
33 }

```

- 匹配

```

1  int u = 1, l = 0;
2  FOR (i, 0, strlen(s)) {
3      int ch = s[i] - 'a';
4      while (u && !t[u][ch]) { u = fa[u]; l = len[u]; }
5      ++l; u = t[u][ch];
6      if (!u) u = 1;
7      if (l) // do something...
8  }

```

- 获取子串状态

```

1  int get_state(int l, int r) {
2      int u = rpos[r], s = r - l + 1;
3      FOR (i, SP - 1, -1) if (len[pa[u][i]] >= s) u = pa[u][i];
4      return u;
5  }

```

- 配合 LCT

```

1 namespace lct_sam {
2     extern struct P *const null;
3     const int M = N;
4     struct P {
5         P *fa, *ls, *rs;
6         int last;
7
8         bool has_fa() { return fa->ls == this || fa->rs == this; }
9         bool d() { return fa->ls == this; }
10        P*& c(bool x) { return x ? ls : rs; }
11        P* up() { return this; }
12        void down() {
13            if (ls != null) ls->last = last;
14            if (rs != null) rs->last = last;
15        }
16        void all_down() { if (has_fa()) fa->all_down(); down(); }
17    } *const null = new P{0, 0, 0, 0}, pool[M], *pit = pool;
18    P* G[N];
19    int t[M][26], len[M] = {-1}, fa[M], sz = 2, last = 1;
20
21    void rot(P* o) {
22        bool dd = o->d();
23        P *f = o->fa, *t = o->c(!dd);
24        if (f->has_fa()) f->fa->c(f->d()) = o; o->fa = f->fa;
25        if (t != null) t->fa = f; f->c(dd) = t;
26        o->c(!dd) = f->up(); f->fa = o;
27    }
28    void splay(P* o) {
29        o->all_down();
30        while (o->has_fa()) {
31            if (o->fa->has_fa())
32                rot(o->d() ^ o->fa->d() ? o : o->fa);
33            rot(o);
34        }
35        o->up();
36    }
37    void access(int last, P* u, P* v = null) {
38        if (u == null) { v->last = last; return; }
39        splay(u);
40        P *t = u;
41        while (t->ls != null) t = t->ls;
42        int L = len[fa[t - pool]] + 1, R = len[u - pool];
43
44        if (u->last) bit::add(u->last - R + 2, u->last - L + 2, 1);
45        else bit::add(1, 1, R - L + 1);
46        bit::add(last - R + 2, last - L + 2, -1);
47
48        u->rs = v;
49        access(last, u->up()->fa, u);
50    }
51    void insert(P* u, P* v, P* t) {
52        if (v != null) { splay(v); v->rs = null; }
53        splay(u);
54        u->fa = t; t->fa = v;
55    }
56
57    void ins(int ch, int pp) {
58        int p = last, np = last = sz++;
59        len[np] = len[p] + 1;
60        for (; p && !t[p][ch]; p = fa[p]) t[p][ch] = np;
61        if (!p) fa[np] = 1;
62        else {
63            int q = t[p][ch];
64            if (len[p] + 1 == len[q]) { fa[np] = q; G[np]->fa = G[q]; }
65            else {
66                int nq = sz++; len[nq] = len[p] + 1;
67                memcpy(t[nq], t[q], sizeof t[0]);
68                insert(G[q], G[fa[q]], G[nq]);
69                G[nq]->last = G[q]->last;
70                fa[nq] = fa[q];
71                fa[np] = fa[q] = nq;

```

```

72         G[np]->fa = G[nq];
73         for (; t[p][ch] == q; p = fa[p]) t[p][ch] = nq;
74     }
75 }
76 access(pp + 1, G[np]);
77 }
78
79 void init() {
80     ++pit;
81     FOR (i, 1, N) {
82         G[i] = pit++;
83         G[i]->ls = G[i]->rs = G[i]->fa = null;
84     }
85     G[1] = null;
86 }
87 }

```

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）

```

1  namespace pam {
2      int t[N][26], fa[N], len[N], rs[N], cnt[N], num[N];
3      int sz, n, last;
4      int _new(int l) {
5          len[sz] = l; cnt[sz] = num[sz] = 0;
6          return sz++;
7      }
8      void init() {
9          memset(t, 0, sz * sizeof t[0]);
10         rs[n = sz = 0] = -1;
11         last = _new(0);
12         fa[last] = _new(-1);
13     }
14     int get_fa(int x) {
15         while (rs[n - 1 - len[x]] != rs[n]) x = fa[x];
16         return x;
17     }
18     void ins(int ch) {
19         rs[++n] = ch;
20         int p = get_fa(last);
21         if (!t[p][ch]) {
22             int np = _new(len[p] + 2);
23             num[np] = num[fa[np]] = t[get_fa(fa[p])][ch] + 1;
24             t[p][ch] = np;
25         }
26         ++cnt[last = t[p][ch]];
27     }
28 }

```

manacher

```

1  int RL[N];
2  void manacher(int* a, int n) { // "abc" => "#a#b#a#"
3      int r = 0, p = 0;
4      FOR (i, 0, n) {
5          if (i < r) RL[i] = min(RL[2 * p - i], r - i);
6          else RL[i] = 1;
7          while (i - RL[i] >= 0 && i + RL[i] < n && a[i - RL[i]] == a[i + RL[i]])
8              RL[i]++;
9          if (RL[i] + i - 1 > r) { r = RL[i] + i - 1; p = i; }
10     }
11     FOR (i, 0, n) --RL[i];
12 }
13

```

哈希

内置了自动双哈希开关（小心 TLE）。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  #define ENABLE_DOUBLE_HASH
5
6  typedef long long LL;
7  typedef unsigned long long ULL;
8
9  const int x = 135;
10 const int N = 4e5 + 10;
11 const int p1 = 1e9 + 7, p2 = 1e9 + 9;
12 ULL xp1[N], xp2[N], xp[N];
13
14 void init_xp() {
15     xp1[0] = xp2[0] = xp[0] = 1;
16     for (int i = 1; i < N; ++i) {
17         xp1[i] = xp1[i - 1] * x % p1;
18         xp2[i] = xp2[i - 1] * x % p2;
19         xp[i] = xp[i - 1] * x;
20     }
21 }
22
23 struct String {
24     char s[N];
25     int length, subsize;
26     bool sorted;
27     ULL h[N], hl[N];
28
29     ULL hash() {
30         length = strlen(s);
31         ULL res1 = 0, res2 = 0;
32         h[length] = 0; // ATTENTION!
33         for (int j = length - 1; j >= 0; --j) {
34             #ifdef ENABLE_DOUBLE_HASH
35                 res1 = (res1 * x + s[j]) % p1;
36                 res2 = (res2 * x + s[j]) % p2;
37                 h[j] = (res1 << 32) | res2;
38             #else
39                 res1 = res1 * x + s[j];
40                 h[j] = res1;
41             #endif
42             // printf("%llu\n", h[j]);
43         }
44         return h[0];
45     }
46
47     // 获取子串哈希，左闭右开区间
48     ULL get_substring_hash(int left, int right) const {
49         int len = right - left;
50         #ifdef ENABLE_DOUBLE_HASH
51             // get hash of s[left...right-1]
52             unsigned int mask32 = ~(0u);
53             ULL left1 = h[left] >> 32, right1 = h[right] >> 32;
54             ULL left2 = h[left] & mask32, right2 = h[right] & mask32;
55             return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
56                 (((left2 - right2 * xp2[len] % p2 + p2) % p2));
57         #else
58             return h[left] - h[right] * xp[len];
59         #endif
60     }
61
62     void get_all_subs_hash(int sublen) {
63         subsize = length - sublen + 1;
64         for (int i = 0; i < subsize; ++i)
65             hl[i] = get_substring_hash(i, i + sublen);
66         sorted = 0;
67     }
68 }
```

```

69 void sort_substring_hash() {
70     sort(hl, hl + subsize);
71     sorted = 1;
72 }
73
74 bool match(ULL key) const {
75     if (!sorted) assert (0);
76     if (!subsize) return false;
77     return binary_search(hl, hl + subsize, key);
78 }
79
80 void init(const char *t) {
81     length = strlen(t);
82     strcpy(s, t);
83 }
84 };
85
86 int LCP(const String &a, const String &b, int ai, int bi) {
87     // Find LCP of a[ai...] and b[bi...]
88     int l = 0, r = min(a.length - ai, b.length - bi);
89     while (l < r) {
90         int mid = (l + r + 1) / 2;
91         if (a.get_substring_hash(ai, ai + mid) == b.get_substring_hash(bi, bi + mid))
92             l = mid;
93         else r = mid - 1;
94     }
95     return l;
96 }
97
98 int check(int ans) {
99     if (T.length < ans) return 1;
100    T.get_all_subs_hash(ans); T.sort_substring_hash();
101    for (int i = 0; i < S.length - ans + 1; ++i)
102        if (!T.match(S.get_substring_hash(i, i + ans)))
103            return 1;
104    return 0;
105 }
106
107 int main() {
108     init_xp(); // DON'T FORGET TO DO THIS!
109
110     for (int tt = 1; tt <= kases; ++tt) {
111         scanf("%d", &n); scanf("%s", str);
112         S.init(str);
113         S.hash(); T.hash();
114     }
115 }

```

二维哈希

```

1 struct Hash2D { // 1-index
2     static const LL px = 131, py = 233, MOD = 998244353;
3     static LL pwx[N], pwy[N];
4     int a[N][N];
5     LL hv[N][N];
6     static void init_xp() {
7         pwx[0] = pwy[0] = 1;
8         FOR (i, 1, N) {
9             pwx[i] = pwx[i - 1] * px % MOD;
10            pwy[i] = pwy[i - 1] * py % MOD;
11        }
12    }
13    void init_hash(int n, int m) {
14        FOR (i, 1, n + 1) {
15            LL s = 0;
16            FOR (j, 1, m + 1) {
17                s = (s * py + a[i][j]) % MOD;
18                hv[i][j] = (hv[i - 1][j] * px + s) % MOD;
19            }
20        }
21    }
22    LL h(int x, int y, int dx, int dy) {

```



```

23     --x; --y;
24     LL ret = hv[x + dx][y + dy] + hv[x][y] * pwx[dx] % MOD * pwy[dy]
25             - hv[x][y + dy] * pwx[dx] - hv[x + dx][y] * pwy[dy];
26     return (ret % MOD + MOD) % MOD;
27 }
28 } ha, hb;
29 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。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 2e5 + 10;
5  const int Nlog = 18;
6
7  struct SuffixArray {
8      const int L;
9      vector<vector<int>> > P;
10     vector<pair<pair<int, int>, int>> > M;
11     int s[N], sa[N], rank[N], height[N];
12     // s: raw string
13     // sa[i]=k: s[k...L-1] ranks i (0 based)
14     // rank[i]=k: the rank of s[i...L-1] is k (0 based)
15     // height[i] = lcp(sa[i-1], sa[i])
16
17     SuffixArray(const string &raw_s) : L(raw_s.length()), P(1, vector<int>(L, 0)), M(L) {
18         for (int i = 0; i < L; i++)
19             P[0][i] = this->s[i] = int(raw_s[i]);
20         for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
21             P.push_back(vector<int>(L, 0));
22             for (int i = 0; i < L; i++)
23                 M[i] = make_pair(make_pair(P[level - 1][i], i + skip < L ? P[level - 1][i + skip] : -1000), i);
24             sort(M.begin(), M.end());
25             for (int i = 0; i < L; i++)
26                 P[level][M[i].second] = (i > 0 && M[i].first == M[i - 1].first) ? P[level][M[i - 1].second] : i;
27         }
28         for (unsigned i = 0; i < P.back().size(); ++i) {
29             rank[i] = P.back()[i];
30             sa[rank[i]] = i;
31         }
32     }
33
34     // This is a traditional way to calculate LCP
35     void getHeight() {
36         memset(height, 0, sizeof height);
37         int k = 0;
38         for (int i = 0; i < L; ++i) {
39             if (rank[i] == 0) continue;
40             if (k) k--;
41             int j = sa[rank[i] - 1];
42             while (i + k < L && j + k < L && s[i + k] == s[j + k]) ++k;
43             height[rank[i]] = k;
44         }
45         rmq_init(height, L);
46     }
47
48     int f[N][Nlog];
49     inline int highbit(int x) {
50         return 31 - __builtin_clz(x);
51     }
52
53     int rmq_query(int x, int y) {
54         int p = highbit(y - x + 1);
55         return min(f[x][p], f[y - (1 << p) + 1][p]);
56     }
57
58     // arr has to be 0 based
59     void rmq_init(int *arr, int length) {

```

```

60     for (int x = 0; x <= highbit(length); ++x)
61         for (int i = 0; i <= length - (1 << x); ++i) {
62             if (!x) f[i][x] = arr[i];
63             else f[i][x] = min(f[i][x - 1], f[i + (1 << (x - 1))][x - 1]);
64         }
65     }
66
67     #ifdef NEW
68     // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
69     int LongestCommonPrefix(int i, int j) {
70         int len = 0;
71         if (i == j) return L - i;
72         for (int k = (int) P.size() - 1; k >= 0 && i < L && j < L; k--) {
73             if (P[k][i] == P[k][j]) {
74                 i += 1 << k;
75                 j += 1 << k;
76                 len += 1 << k;
77             }
78         }
79         return len;
80     }
81     #else
82     int LongestCommonPrefix(int i, int j) {
83         // getHeight() must be called first
84         if (i == j) return L - i;
85         if (i > j) swap(i, j);
86         return rmq_query(i + 1, j);
87     }
88     #endif
89
90     int checkNonOverlappingSubstring(int K) {
91         // check if there is two non-overlapping identical substring of length K
92         int minsa = 0, maxsa = 0;
93         for (int i = 0; i < L; ++i) {
94             if (height[i] < K) {
95                 minsa = sa[i]; maxsa = sa[i];
96             } else {
97                 minsa = min(minsa, sa[i]);
98                 maxsa = max(maxsa, sa[i]);
99                 if (maxsa - minsa >= K) return 1;
100             }
101         }
102         return 0;
103     }
104
105     int checkBelongToDifferentSubstring(int K, int split) {
106         int minsa = 0, maxsa = 0;
107         for (int i = 0; i < L; ++i) {
108             if (height[i] < K) {
109                 minsa = sa[i]; maxsa = sa[i];
110             } else {
111                 minsa = min(minsa, sa[i]);
112                 maxsa = max(maxsa, sa[i]);
113                 if (maxsa > split && minsa < split) return 1;
114             }
115         }
116         return 0;
117     }
118 } *S;
119
120
121 int main() {
122     string s, t;
123     cin >> s >> t;
124     int sp = s.length();
125     s += "*" + t;
126     S = new SuffixArray(s);
127     S->getHeight();
128     int left = 0, right = sp;
129     while (left < right) {
130         int mid = (left + right + 1) / 2;

```

```

131         if (S->checkBelongToDifferentSubstring(mid, sp))
132             left = mid;
133         else right = mid - 1;
134     }
135     printf("%d\n", left);
136 }

```

- 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

```

1 // rk [0..n-1] -> [1..n], sa/ht [1..n]
2 // s[i] > 0 && s[n] = 0
3 // b: normally as bucket
4 // c: normally as bucket1
5 // d: normally as bucket2
6 // f: normally as cntbuf
7
8 template<size_t size>
9 struct SuffixArray {
10     bool t[size << 1];
11     int b[size], c[size];
12     int sa[size], rk[size], ht[size];
13     inline bool isLMS(const int i, const bool *t) { return i > 0 && t[i] && !t[i - 1]; }
14     template<class T>
15     inline void inducedSort(T s, int *sa, const int n, const int M, const int bs,
16                             bool *t, int *b, int *f, int *p) {
17         fill(b, b + M, 0); fill(sa, sa + n, -1);
18         FOR (i, 0, n) b[s[i]]++;
19         f[0] = b[0];
20         FOR (i, 1, M) f[i] = f[i - 1] + b[i];
21         FORD (i, bs - 1, -1) sa[--f[s[p[i]]]] = p[i];
22         FOR (i, 1, M) f[i] = f[i - 1] + b[i - 1];
23         FOR (i, 0, n) if (sa[i] > 0 && !t[sa[i] - 1]) sa[f[s[sa[i] - 1]]++] = sa[i] - 1;
24         f[0] = b[0];
25         FOR (i, 1, M) f[i] = f[i - 1] + b[i];
26         FORD (i, n - 1, -1) if (sa[i] > 0 && t[sa[i] - 1]) sa[--f[s[sa[i] - 1]]] = sa[i] - 1;
27     }
28     template<class T>
29     inline void sais(T s, int *sa, int n, bool *t, int *b, int *c, int M) {
30         int i, j, bs = 0, cnt = 0, p = -1, x, *r = b + M;
31         t[n - 1] = 1;
32         FORD (i, n - 2, -1) t[i] = s[i] < s[i + 1] || (s[i] == s[i + 1] && t[i + 1]);
33         FOR (i, 1, n) if (t[i] && !t[i - 1]) c[bs++] = i;
34         inducedSort(s, sa, n, M, bs, t, b, r, c);
35         for (i = bs = 0; i < n; i++) if (isLMS(sa[i], t)) sa[bs++] = sa[i];
36         FOR (i, bs, n) sa[i] = -1;
37         FOR (i, 0, bs) {
38             x = sa[i];
39             for (j = 0; j < n; j++) {
40                 if (p == -1 || s[x + j] != s[p + j] || t[x + j] != t[p + j]) { cnt++, p = x; break; }
41                 else if (j > 0 && (isLMS(x + j, t) || isLMS(p + j, t))) break;
42             }
43             x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bs + x] = cnt - 1;
44         }
45         for (i = j = n - 1; i >= bs; i--) if (sa[i] >= 0) sa[j--] = sa[i];
46         int *s1 = sa + n - bs, *d = c + bs;
47         if (cnt < bs) sais(s1, sa, bs, t + n, b, c + bs, cnt);
48         else FOR (i, 0, bs) sa[s1[i]] = i;
49         FOR (i, 0, bs) d[i] = c[sa[i]];
50         inducedSort(s, sa, n, M, bs, t, b, r, d);
51     }
52     template<typename T>
53     inline void getHeight(T s, const int n, const int *sa) {
54         for (int i = 0, k = 0; i < n; i++) {
55             if (rk[i] == 0) k = 0;
56             else {
57                 if (k > 0) k--;
58                 int j = sa[rk[i] - 1];
59                 while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
60             }
61         }
62     }
63 }

```

```

61         ht[rk[i]] = k;
62     }
63 }
64 template<class T>
65 inline void init(T s, int n, int M) {
66     sais(s, sa, ++n, t, b, c, M);
67     for (int i = 1; i < n; i++) rk[sa[i]] = i;
68     getHeight(s, n, sa);
69 }
70 };
71
72 const int N = 2E5 + 100;
73 SuffixArray<N> sa;
74
75 int main() {
76     string s; cin >> s; int n = s.length();
77     sa.init(s, n, 128);
78     FOR (i, 1, n + 1) printf("%d%c", sa.sa[i] + 1, i == _i - 1 ? '\n' : ' ');
79     FOR (i, 2, n + 1) printf("%d%c", sa.ht[i], i == _i - 1 ? '\n' : ' ');
80 }

```

KMP

- 前缀函数（每一个前缀的最长 border）

```

1 void get_pi(int a[], char s[], int n) {
2     int j = a[0] = 0;
3     FOR (i, 1, n) {
4         while (j && s[i] != s[j]) j = a[j - 1];
5         a[i] = j += s[i] == s[j];
6     }
7 }

```

- Z 函数（每一个后缀和该字符串的 LCP 长度）

```

1 void get_z(int a[], char s[], int n) {
2     int l = 0, r = 0; a[0] = n;
3     FOR (i, 1, n) {
4         a[i] = i > r ? 0 : min(r - i + 1, a[i - l]);
5         while (i + a[i] < n && s[a[i]] == s[i + a[i]]) ++a[i];
6         if (i + a[i] - 1 > r) { l = i; r = i + a[i] - 1; }
7     }
8 }

```

Trie

```

1 namespace trie {
2     int t[N][26], sz, ed[N];
3     void init() { sz = 2; memset(ed, 0, sizeof ed); }
4     int _new() { memset(t[sz], 0, sizeof t[sz]); return sz++; }
5     void ins(char* s, int p) {
6         int u = 1;
7         FOR (i, 0, strlen(s)) {
8             int c = s[i] - 'a';
9             if (!t[u][c]) t[u][c] = _new();
10            u = t[u][c];
11        }
12        ed[u] = p;
13    }
14 }

```

AC 自动机

```

1 const int N = 1e6 + 100, M = 26;
2
3 int mp(char ch) { return ch - 'a'; }
4
5 struct ACA {
6     int ch[N][M], danger[N], fail[N];
7     int sz;

```

```

8 void init() {
9     sz = 1;
10    memset(ch[0], 0, sizeof ch[0]);
11    memset(danger, 0, sizeof danger);
12 }
13 void insert(const string &s, int m) {
14     int n = s.size(); int u = 0, c;
15     FOR (i, 0, n) {
16         c = mp(s[i]);
17         if (!ch[u][c]) {
18             memset(ch[sz], 0, sizeof ch[sz]);
19             danger[sz] = 0; ch[u][c] = sz++;
20         }
21         u = ch[u][c];
22     }
23     danger[u] |= 1 << m;
24 }
25 void build() {
26     queue<int> Q;
27     fail[0] = 0;
28     for (int c = 0, u; c < M; c++) {
29         u = ch[0][c];
30         if (u) { Q.push(u); fail[u] = 0; }
31     }
32     while (!Q.empty()) {
33         int r = Q.front(); Q.pop();
34         danger[r] |= danger[fail[r]];
35         for (int c = 0, u; c < M; c++) {
36             u = ch[r][c];
37             if (!u) {
38                 ch[r][c] = ch[fail[r]][c];
39                 continue;
40             }
41             fail[u] = ch[fail[r]][c];
42             Q.push(u);
43         }
44     }
45 }
46 } ac;
47
48 char s[N];
49
50 int main() {
51     int n; scanf("%d", &n);
52     ac.init();
53     while (n--) {
54         scanf("%s", s);
55         ac.insert(s, 0);
56     }
57     ac.build();
58
59     scanf("%s", s);
60     int u = 0; n = strlen(s);
61     FOR (i, 0, n) {
62         u = ac.ch[u][mp(s[i])];
63         if (ac.danger[u]) {
64             puts("YES");
65             return 0;
66         }
67     }
68     puts("NO");
69     return 0;
70 }

```