1.KMP

`f[i]`为`s`以`i`结束的能作为`s`前缀的最长串长

```
void getf(const string& s, vector<int>& f) {
    int n = s.size(); f.resize(n, 0);
    for (int i = 1, j = 0; i != n; ++i) {
        while (j && s[i] != s[j]) j = f[j - 1];
        if (s[i] == s[j]) ++j;
        f[i] = j;
    }
}
```

`l[i]`为`t`以`i`结束的能作为`s`前缀的的最长串长

```
void getl(const string& s, const string& t, const vector<int>& f, vector<int>&
l) {
    int n = s.size(), m = t.size(); l.resize(m, 0);
    for (int i = 0, j = 0; i != m; ++i) {
        while(j && t[i] != s[j]) j = f[j - 1];
        if (t[i] == s[j]) ++j;
        l[i] = j;
        if (j == n) j = f[j - 1];
    }
}
```

2.EXKMP

`z[i]`为`s`以`i`为起始的能作为`s`前缀的最长串长

```
void getz(const string& s, vector<int>& z) {
    int n = s.size(); z.resize(n, 0);
    for (int i = 1, l = 0, r = 0; i != n; ++i) {
        int j = z[i - l];
        if (i+j>r) for (j=max(0, r-i); i+j!=n&&s[ia+j]==s[j]; ++j);
        z[i] = j; if (i+j-1>r) l=i, r=i+j-1;
    }
    z[0] = n;
}
```

`e[i]`为`t`以`i`起始的能作为`s`前缀的最长串长

```
void gete(const string& s, const string& t, const vector<int>& z, vector<int>&
e) {
    int n = s.size(), m = t.size(); e.resize(m, 0);
    for (int i = 0, l = 0, r = 0; i != m; ++i) {
        int j = z[i - l];
        if (i+j>r) for (j=max(0, r-i); i+j!=m&&t[i+j]==s[j]; ++j);
        e[i] = j; if (i+j-1>r) l=i, r=i+j-1;
    }
}
```

3.Manacher

`r[i]`为`t`串以`i`为中心的回文半径：`t[i+r[i]]==t[i-r[i]]`

```
void getr(const string& s, vector<int>& r) {
    int n = s.size(), m = 2 * n + 1; r.resize(m, 0);
    string t(m, '#');
    for (int i = 0; i != n; ++i) t[2 * i + 1] = s[i];
    for (int i = 1, p = 0, w = 0; i != m; ++i) {
        int j = (i>p+w ? 0 : min(r[p-(i-p)], p+w-i));
        while(0<=i-j-1 && i+j+1<m && t[i-j-1]==t[i+j+1]) ++j;
        r[i] = j; if (i + j > p + w)  p = i, w = j;
    }
}
```

判断区间`[l,u)`是否为回文

```
bool ispal(const vector<int>& r, int l, int u) {
    return r[l + u] >= u - l;
}
```

4.哈希

```cpp
typedef long long ll;
typedef pair<ll, ll> pll;
const ll p1 = 1145140019, b1 = 893;
const ll p2 = 1919810021, b2 = 364;
ll hs1[N], hb1[N];
ll hs2[N], hb2[N];

char s[N]; int n;
void genh() {
    hb1[0] = hb2[0] = 1;
    for (int i = 1; i <= n; ++i) {
        hs1[i] = (hs1[i - 1] * b1 + s[i]) % p1;
        hb1[i] = hb1[i - 1] * b1 % p1;
        hs2[i] = (hs2[i - 1] * b2 + s[i]) % p2;
        hb2[i] = hb2[i - 1] * b2 % p2;
    }
}

pll geth(int l, int r) {
    ll v1 = (hs1[r] - hs1[l - 1] * hb1[r - l + 1]) % p1;
    ll v2 = (hs2[r] - hs2[l - 1] * hb2[r - l + 1]) % p2;
    if (v1 < 0) v1 += p1; if (v2 < 0) v2 += p2;
    return { v1, v2 };
}

pll concat(pll lh, int ln, pll rh, int rn) {
    pll r;
    r.first = (lh.first * hb1[rn] + rh.first) % p1;
    r.second = (lh.second * hb2[rn] + rh.second) % p2;
    return r;
}

int lcp(int u, int v) {
    if (s[u] != s[v]) return 0;
    int l = 1, r = n - max(u, v) + 1;
    while (l <= r) {
        int mid = (l + r) / 2;
        if (geth(u, u + mid - 1) == geth(v, v + mid - 1))
            l = mid + 1;
        else
            r = mid - 1;
    }
    return r;
}

int lcs(int u, int v) {
    if (s[u] != s[v]) return 0;
    int l = 1, r = min(u, v);
    while (l <= r) {
        int mid = (l + r) / 2;
        if (geth(u - mid + 1, u) == geth(v - mid + 1, v))
            l = mid + 1;
        else
            r = mid - 1;
    }
    return r;
}
```

5.自动机

1.AC自动机

```
int g[N][26], f[N], e[N], nc;

int gn() {
    int p = nc++; f[p] = e[p] = 0;
    memset(g[p], 0, sizeof(g[p]));
    return p;
}

void clr() { nc = 0; gn(); }

void ins(const string& s) {
    int p = 0;
    for (int i = 0; i != s.size(); ++i) {
        int o = s[i] - 'a';
        if (!g[p][o]) g[p][o] = gn();
        p = g[p][o];
    }
    e[p] = 1;
}

void build() {
    queue<int> q;
    for (int o = 0; o != 26; ++o)
        if (g[0][o]) q.push(g[0][o]);
    while(!q.empty()) {
        int u = q.front(); q.pop();
        if (e[f[u]]) e[u] = 1;
        for (int o = 0; o != 26; ++o) {
            int& v = g[u][o];
            if (!v) v = g[f[u]][o];
            else {
                f[v] = g[f[u]][o];
                q.push(v);
            }
        }
    }
}
```

2.后缀自动机

注：将注释去掉后即为（假的）广义SAM

```
char s[N]; int n;
int g[N][26], f[N], l[N], w[N], nc;

inline int gn(int len, int q = 0) {
    int p = nc++; l[p] = len;
    if (q) memcpy(g[p], g[q], sizeof(g[p])), f[p] = f[q];
    else memset(g[p], 0, sizeof(g[p])), f[p] = 0;
    return p;
}

void clr() { nc = 0; gn(0); f[0] = -1; }

int extend(int p, int o) {
    //  if (g[p][o] && l[p] + 1 == l[g[p][o]]) return g[p][o];
    int np = gn(l[p] + 1);
    for (; ~p && !g[p][o]; p = f[p])
        g[p][o] = np;
    if (!~p) f[np] = 0;
    else {
        int q = g[p][o];
        if (l[q] == l[p] + 1) f[np] = q;
        else {
            int nq = gn(l[p] + 1, q);
            f[q] = f[np] = nq;
            for (; ~p && g[p][o] == q; p = f[p])
                g[p][o] = nq;
        }
    }
    return np;
}

int c[N], pos[N];
void sort_sam() {
    fill(c, c + n + 1, 0);
    for (int p = 0; p != nc; ++p) c[l[p]]++;
    for (int i = 1; i <= n; ++i) c[i] += c[i - 1];
    for (int p = 0; p != nc; ++p) pos[--c[l[p]]] = p;
}
```

```
\\广义后缀自动机
struct Suffix_Automaton{
    int O,link[N],maxlen[N],trans[N][26];
    //link[i]: 后缀链接
    //trans[i]: 状态转移数组
    Suffix_Automaton(){O=1;}//根初始化为1
    inline int insert(Re ch,Re last){
        if(trans[last][ch]){
            Re p=last,x=trans[p][ch];
            if(maxlen[p]+1==maxlen[x])return x;//即最初的特判1
            else{
                Re y=++O;maxlen[y]=maxlen[p]+1;
                for(Re i=0;i<26;++i)trans[y][i]=trans[x][i];
                while(p&&trans[p][ch]==x)trans[p][ch]=y,p=link[p];
                link[y]=link[x],link[x]=y;
                return y;//即最初的特判2
            }
        }
        Re z=++O,p=last;maxlen[z]=maxlen[last]+1;
        while(p&&!trans[p][ch])trans[p][ch]=z,p=link[p];
        if(!p)link[z]=1;
        else{
            Re x=trans[p][ch];
            if(maxlen[p]+1==maxlen[x])link[z]=x;
            else{
                Re y=++O;maxlen[y]=maxlen[p]+1;
                for(Re i=0;i<26;++i)trans[y][i]=trans[x][i];
                while(p&&trans[p][ch]==x)trans[p][ch]=y,p=link[p];
                link[y]=link[x],link[z]=link[x]=y;
            }
        }
        return z;
    }
    inline void sakura(){
        LL ans=0;
        for(Re i=2;i<=O;++i)ans+=maxlen[i]-maxlen[link[i]];
        printf("%lld\n",ans);
    }
}SAM;
int main(){
//    freopen("123.txt","r",stdin);
    in(n);
    for(Re i=1;i<=n;++i){
        scanf("%s",ch+1);Re last=1;
        for(Re j=1;ch[j];++j)last=SAM.insert(ch[j]-'a',last);
    }
    SAM.sakura();
}
```

3.回文自动机

```
char s[N];
int g[N][26], f[N], l[N], nc;

int gn(int len) {
    int p = nc++; l[p] = len; f[p] = 0;
    memset(g[p], 0, sizeof(g[p]));
    return p;
}

int gf(int p, int i) {
    while (s[i] != s[i - l[p] - 1])
        p = f[p];
    return p;
}

void clr() {
    nc = 0; gn(0); gn(-1);
    f[0] = 1; f[1] = -1;
}

int extend(int p, int i) {
    int o = s[i] - 'a'; p = gf(p, i);
    if (!g[p][o]) {
        int q = gn(l[p] + 2); g[p][o] = q;
        if (p != 1) f[q] = g[gf(f[p], i)][o];
    }
    p = g[p][o];
    return p;
}
```

3.2双端插入

```
namespace pam {

char s[N<<1]; int tl, tr;
int g[N][26], f[N], l[N], nc;
int pl, pr;

int gn(int len) {
    int q = nc++; l[q] = len; f[q] = 0;
    memset(g[q], 0, sizeof(g[q]));
    return q;
}

void clr() {
    fill(s + tl, s + tr + 1, 0);
    nc = 0; gn(0); gn(-1); f[0] = 1;
    tr = N; tl = tr + 1;
    pl = pr = 0;
}

int gfl(int p, int i) {
    while (s[i] != s[i + l[p] + 1]) p = f[p];
    return p;
}

int gfr(int p, int i) {
    while (s[i] != s[i - l[p] - 1]) p = f[p];
    return p;
}

int extl(int o) {
    s[--tl] = o + 'a'; int p = gfl(pl, tl);
    if (!g[p][o]) {
        int q = gn(l[p] + 2); g[p][o] = q;
        if (p != 1) f[q] = g[gfl(f[p], tl)][o];
    }
    pl = g[p][o];
    if (tl + l[pl] - 1 == tr) pr = pl;
    return h[pl];
}

int extr(int o) {
    s[++tr] = o + 'a'; int p = gfr(pr, tr);
    if (!g[p][o]) {
        int q = gn(l[p] + 2); g[p][o] = q;
        if (p != 1) f[q] = g[gfr(f[p], tr)][o];
    }
    pr = g[p][o];
    if (tr - l[pr] + 1 == tl) pl = pr;
    return h[pr];
}

}
```

6.后缀数组

```cpp
//倍增
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <iostream>

using namespace std;

const int N = 1000010;

char s[N];
int n, sa[N], rk[N], oldrk[N << 1], id[N], px[N], cnt[N];
// px[i] = rk[id[i]]（用于排序的数组所以叫 px）

bool cmp(int x, int y, int w) {
  return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
}

int main() {
  int i, m = 300, p, w;

  scanf("%s", s + 1);
  n = strlen(s + 1);
  for (i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
  for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
  for (i = n; i >= 1; --i) sa[cnt[rk[i]]--] = i;

  for (w = 1;; w <<= 1, m = p) {  // m=p 就是优化计数排序值域
    for (p = 0, i = n; i > n - w; --i) id[++p] = i;
    for (i = 1; i <= n; ++i)
      if (sa[i] > w) id[++p] = sa[i] - w;
    memset(cnt, 0, sizeof(cnt));
    for (i = 1; i <= n; ++i) ++cnt[px[i] = rk[id[i]]];
    for (i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
    for (i = n; i >= 1; --i) sa[cnt[px[i]]--] = id[i];
    memcpy(oldrk, rk, sizeof(rk));
    for (p = 0, i = 1; i <= n; ++i)
      rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
    if (p == n) {
      for (int i = 1; i <= n; ++i) sa[rk[i]] = i;
      break;
    }
  }

  for (i = 1; i <= n; ++i) printf("%d ", sa[i]);

  return 0;
}
```

```cpp
//SAIS// 后缀类型
#define L_TYPE 0
#define S_TYPE 1
```

```
// 判断一个字符是否为LMS字符
inline bool is_lms_char(int *type, int x) {
    return x > 0 && type[x] == S_TYPE && type[x - 1] == L_TYPE;
}

// 判断两个LMS子串是否相同
inline bool equal_substring(int *S, int x, int y, int *type) {
    do {
        if (S[x] != S[y])
            return false;
        x++, y++;
    } while (!is_lms_char(type, x) && !is_lms_char(type, y));

    return S[x] == S[y];
}

// 诱导排序(从*型诱导到L型、从L型诱导到S型)
// 调用之前应将*型按要求放入SA中
inline void induced_sort(int *S, int *SA, int *type, int *bucket, int *lbucket,
                         int *sbucket, int n, int SIGMA) {
    for (int i = 0; i <= n; i++)
        if (SA[i] > 0 && type[SA[i] - 1] == L_TYPE)
            SA[lbucket[S[SA[i] - 1]]++] = SA[i] - 1;
    for (int i = 1; i <= SIGMA; i++)  // Reset S-type bucket
        sbucket[i] = bucket[i] - 1;
    for (int i = n; i >= 0; i--)
        if (SA[i] > 0 && type[SA[i] - 1] == S_TYPE)
            SA[sbucket[S[SA[i] - 1]]--] = SA[i] - 1;
}

// SA-IS主体
// S是输入字符串，length是字符串的长度, SIGMA是字符集的大小
static int *SAIS(int *S, int length, int SIGMA) {
    int n = length - 1;
    int *type = new int[n + 1];  // 后缀类型
    int *position = new int[n + 1];  // 记录LMS子串的起始位置
    int *name = new int[n + 1];  // 记录每个LMS子串的新名称
    int *SA = new int[n + 1];  // SA数组
    int *bucket = new int[SIGMA + 1];  // 每个字符的桶
    int *lbucket = new int[SIGMA + 1];  // 每个字符的L型桶的起始位置
    int *sbucket = new int[SIGMA + 1];  // 每个字符的S型桶的起始位置

    // 初始化每个桶
    memset(bucket, 0, sizeof(int) * (SIGMA + 1));
    for (int i = 0; i <= n; i++)
        bucket[S[i]]++;
    lbucket[0] = sbucket[0] = 0;
    for (int i = 1; i <= SIGMA; i++) {
        bucket[i] += bucket[i - 1];
        lbucket[i] = bucket[i - 1];
        sbucket[i] = bucket[i] - 1;
    }

    // 确定后缀类型(利用引理2.1)
    type[n] = S_TYPE;
    for (int i = n - 1; i >= 0; i--) {
        if (S[i] < S[i + 1])
```

```
            type[i] = S_TYPE;
        else if (S[i] > S[i + 1])
            type[i] = L_TYPE;
        else
            type[i] = type[i + 1];
    }

    // 寻找每个LMS子串
    int cnt = 0;
    for (int i = 1; i <= n; i++)
        if (type[i] == S_TYPE && type[i - 1] == L_TYPE)
            position[cnt++] = i;

    // 对LMS子串进行排序
    fill(SA, SA + n + 1, -1);
    for (int i = 0; i < cnt; i++)
        SA[sbucket[S[position[i]]]--] = position[i];
    induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);

    // 为每个LMS子串命名
    fill(name, name + n + 1, -1);
    int lastx = -1, namecnt = 1;  // 上一次处理的LMS子串与名称的计数
    bool flag = false;  // 这里顺便记录是否有重复的字符
    for (int i = 1; i <= n; i++) {
        int x = SA[i];

        if (is_lms_char(type, x)) {
            if (lastx >= 0 && !equal_substring(S, x, lastx, type))
                namecnt++;
            // 因为只有相同的LMS子串才会有同样的名称
            if (lastx >= 0 && namecnt == name[lastx])
                flag = true;

            name[x] = namecnt;
            lastx = x;
        }
    }  // for
    name[n] = 0;

    // 生成S1
    int *S1 = new int[cnt];
    int pos = 0;
    for (int i = 0; i <= n; i++)
        if (name[i] >= 0)
            S1[pos++] = name[i];

    int *SA1;
    if (!flag) {
        // 直接计算SA1
        SA1 = new int[cnt + 1];

        for (int i = 0; i < cnt; i++)
            SA1[S1[i]] = i;
    } else
        SA1 = SAIS(S1, cnt, namecnt);  // 递归计算SA1

    // 从SA1诱导到SA
    lbucket[0] = sbucket[0] = 0;
    for (int i = 1; i <= SIGMA; i++) {
        lbucket[i] = bucket[i - 1];
        sbucket[i] = bucket[i] - 1;
    }
    fill(SA, SA + n + 1, -1);
    for (int i = cnt - 1; i >= 0; i--)  // 这里是逆序扫描SA1，因为SA中S型桶是倒序的
        SA[sbucket[S[position[SA1[i]]]]--] = position[SA1[i]];
    induced_sort(S, SA, type, bucket, lbucket, sbucket, n, SIGMA);

    // 后缀数组计算完毕
    return SA;
}
```

求height

```
for (i = 1, k = 0; i <= n; ++i) {
  if (k) --k;
  while (s[i + k] == s[sa[rk[i] - 1] + k]) ++k;
  ht[rk[i]] = k;  // height太长了缩写为ht
}
```

7.后缀平衡树

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

const int N = 8e5 + 5;
const double INF = 1e18;

void decode(char* s, int len, int mask) {
  for (int i = 0; i < len; ++i) {
    mask = (mask * 131 + i) % len;
    swap(s[i], s[mask]);
  }
}

int q, n, na;
char a[N], t[N];

// SuffixBST(SGT Ver)

// 顺序加入，查询时将询问串翻转
// 以i结束的前缀，对应节点的编号为i
// 注意：不能写懒惰删除，否则可能会破坏树的结构
const double alpha = 0.75;
int root;
int sz[N], L[N], R[N];
double tag[N];
int buffer_size, buffer[N];

bool cmp(int x, int y) {
  if (t[x] != t[y]) return t[x] < t[y];
  return tag[x - 1] < tag[y - 1];
}

void init() { root = 0; }

void new_node(int& rt, int p, double lv, double rv) {
  rt = p;
  sz[rt] = 1;
  tag[rt] = (lv + rv) / 2;
  L[rt] = R[rt] = 0;
}

void push_up(int x) {
  if (!x) return;
  sz[x] = sz[L[x]] + 1 + sz[R[x]];
}

bool balance(int rt) { return alpha * sz[rt] > max(sz[L[rt]], sz[R[rt]]); }

void flatten(int rt) {
  if (!rt) return;
  flatten(L[rt]);
  buffer[++buffer_size] = rt;
  flatten(R[rt]);
}

void build(int& rt, int l, int r, double lv, double rv) {
  if (l > r) {
    rt = 0;
    return;
  }
  int mid = (l + r) >> 1;
  double mv = (lv + rv) / 2;

  rt = buffer[mid];
  tag[rt] = mv;
  build(L[rt], l, mid - 1, lv, mv);
  build(R[rt], mid + 1, r, mv, rv);
  push_up(rt);
}

void rebuild(int& rt, double lv, double rv) {
  buffer_size = 0;
  flatten(rt);
  build(rt, 1, buffer_size, lv, rv);
}

void insert(int& rt, int p, double lv, double rv) {
  if (!rt) {
    new_node(rt, p, lv, rv);
    return;
  }

  if (cmp(p, rt))
    insert(L[rt], p, lv, tag[rt]);
  else
    insert(R[rt], p, tag[rt], rv);

  push_up(rt);
  if (!balance(rt)) rebuild(rt, lv, rv);
}

void remove(int& rt, int p, double lv, double rv) {
  if (!rt) return;

  if (rt == p) {
    if (!L[rt] || !R[rt]) {
      rt = (L[rt] | R[rt]);
    } else {
      // 找到rt的前驱来替换rt
      int nrt = L[rt], fa = rt;
      while (R[nrt]) {
        fa = nrt;
        sz[fa]--;
        nrt = R[nrt];
      }
      if (fa == rt) {
        R[nrt] = R[rt];
      } else {
        L[nrt] = L[rt];
        R[nrt] = R[rt];
        R[fa] = 0;
      }
      rt = nrt;
      tag[rt] = (lv + rv) / 2;
    }
  } else {
    double mv = (lv + rv) / 2;
    if (cmp(p, rt))
      remove(L[rt], p, lv, mv);
    else
      remove(R[rt], p, mv, rv);
  }

  push_up(rt);
  if (!balance(rt)) rebuild(rt, lv, rv);
}

bool cmp1(char* s, int len, int p) {
  for (int i = 1; i <= len; ++i, --p) {
    if (s[i] < t[p]) return true;
    if (s[i] > t[p]) return false;
  }
}

int query(int rt, char* s, int len) {
  if (!rt) return 0;
  if (cmp1(s, len, rt))
    return query(L[rt], s, len);
  else
    return sz[L[rt]] + 1 + query(R[rt], s, len);
}

void solve() {
  n = 0;
  scanf("%d", &q);
  init();

  scanf("%s", a + 1);
  na = strlen(a + 1);
  for (int i = 1; i <= na; ++i) {
    t[++n] = a[i];
    insert(root, n, 0, INF);
  }

  int mask = 0;
  char op[10];
  for (int i = 1; i <= q; ++i) {
    scanf("%s", op);

    //三种情况分别处理

    if (op[0] == 'A') {  // ADD
      scanf("%s", a + 1);
      na = strlen(a + 1);
      decode(a + 1, na, mask);

      for (int i = 1; i <= na; ++i) {
        t[++n] = a[i];
        insert(root, n, 0, INF);
      }
    } else if (op[0] == 'D') {  // DEL
      int x;
      scanf("%d", &x);
      while (x) {
        remove(root, n, 0, INF);
        --n;
        --x;
      }
    } else if (op[0] == 'Q') {  // QUERY
      scanf("%s", a + 1);
      na = strlen(a + 1);
      decode(a + 1, na, mask);

      reverse(a + 1, a + 1 + na);

      a[na + 1] = 'Z' + 1;
      a[na + 2] = 0;
      int ans = query(root, a, na + 1);

      --a[na];
      ans -= query(root, a, na + 1);

      printf("%d\n", ans);
      mask ^= ans;
    }
  }
}

int main() {
  solve();
  return 0;
}
```

8.lyndon分解

```cpp
// C++ Version
// duval_algorithm
vector<string> duval(string const& s) {
  int n = s.size(), i = 0;
  vector<string> factorization;
  while (i < n) {
    int j = i + 1, k = i;
    while (j < n && s[k] <= s[j]) {
      if (s[k] < s[j])
        k = i;
      else
        k++;
      j++;
    }
    while (i <= k) {
      factorization.push_back(s.substr(i, j - k));
      i += j - k;
    }
  }
  return factorization;
}
```