字符串

KMP

```
void solve(int CaseT) {
    string s, p;
    cin >> s >> p;
    int n = SZ(s), m = SZ(p);
    s = "#" + s;
    p = "#" + p;
    // calc the next[]
    vec<int> nxt(m + 1, 0);
    nxt[1] = 0;
    for (int i = 2, j = 0; i <= m; i++) {
        while (j && p[i] != p[j + 1])
            j = nxt[j];
        if (p[j + 1] == p[i])
            j++;
        nxt[i] = j;
    }
    // match operation
    vec<int> pos;
    for (int i = 1, j = 0; i <= n; i++) {
        while (j == m || (j && s[i] != p[j + 1]))
            j = nxt[j];
        if (p[j + 1] == s[i])
            j++;
        if (j == m) {
            pos.pb(i - m + 1);
            j = nxt[j];
        }
    }
}
```

Z_Algorithm(exkmp)

```
// Z algorithm 最长公共前后缀
// Z[i] 表示字符串S和S[i]...S[n]的最长公共前缀

vector<int> Z_algorithm(string s) {
    int n = SZ(s);
    vec<int> Z(n + 1, 0);
    Z[1] = 0;
    int l = 1, r = 0;
    for (int i = 2; i < n; i++) {
        if (i > r) {
            Z[i] = 0;
```

```
        } else {
            int k = i - l + 1;
            Z[i] = min(Z[k], r - i + 1);
        }

        while (i + Z[i] <= n && s[Z[i] + 1] == s[Z[i] + i]) Z[i]++;
        if (i + Z[i] - 1 > r)
            l = i, r = i + Z[i] - 1;
    }
    return Z;
}

void solve(int CaseT) {
    string s, p;
    cin >> s >> p;
    int n = SZ(s), m = SZ(p);
    string res = "#" + p + "#" + s;
    auto Z = Z_algorithm(res);

    vec<int> pos;
    for (int i = m + 1; i < SZ(res); i++) {
        if (Z[i] == m) {
            pos.pb(i - m - 1);
        }
    }
}
```

Manacher

```
vector<int> manacher(string s) {
    int n = SZ(s), m = 0;
    string t = "$";
    for (int i = 0; i < n; i++) {
        t.pb(s[i]);
        t.pb('$');
    }

    m = SZ(t);
    vec<int> p(m + 1, 0);

    int M = 0, R = 0;
    for (int i = 1; i <= m; i++) {
        if (i > R) {
            p[i] = 1;
        } else {
            p[i] = min(p[2 * M - i], R - i + 1);
        }
        while (p[i] + i <= m && i - p[i] >= 0 && t[i + p[i]] == t[i - p[i]])
            ++p[i];
        if (i + p[i] - 1 > R)
            M = i, R = i + p[i] - 1;
    }
    return p;
```

```
}

void solve(int CaseT) {
    string s;
    cin >> s;
    auto p = manacher(s);
    int ans = 0;
    for (int i = 1; i < SZ(p); i++)
        ans = max(ans, p[i]);
    cout << ans - 1 << '\n';
}
```

ACAM

```
const int N = 1000010, M = 26;
struct ACAM {
    int nxt[N][M], fail[N], cnt[N], id[N];
    int arr[N], t;
    int root, idx;
    ACAM() {
        clear();
    }
    void clear() {
        // memset(nxt[0], 0, sizeof nxt[0]);
        memset(cnt, 0, sizeof cnt);
        // root = idx = 0;
    }
    int newnode() {
        idx ++;
        memset(nxt[idx], 0, sizeof nxt[idx]);
        return idx;
    }
    int insert(string s, int tag) {
        int len = s.size();
        int now = root;
        for (int i = 0; i < len; i++) {
            int u = s[i] - 'a';
            if (!nxt[now][u]) nxt[now][u] = newnode();
            now = nxt[now][u];
        }
        return id[tag] = now;
    }
    void build() {
        fail[root] = root;
        queue<int> q;
        for (int i = 0; i < M; i ++) if (nxt[root][i]) q.push(nxt[root][i]);
        while(q.size()) {
            int u = q.front(); q.pop();
            arr[t ++] = u;
            for (int i = 0; i < M; i++) {
                if (!nxt[u][i]) {
                    nxt[u][i] = nxt[fail[u]][i];
                    // 虚空儿子，给不存在的儿子造一个fail
                }
                else {
```

```
                    int tmp = nxt[u][i];
                    fail[tmp] = nxt[fail[u]][i];
                    q.push(tmp);
                }
            }
        }
    }
    void solve(string s, int m) {
        int now = root;
        for (int i = 0; i < s.size(); i++) {
            if(s[i] == '0') now = root;
            else now = nxt[now][s[i] - 'a'];
            cnt[now]++;
        }
        for (int i = t - 1; i > 0; i--) {
            cnt[fail[arr[i]]] += cnt[arr[i]];
        }
        for (int i = 0; i < m; i ++) {
            cout << cnt[id[i]] << '\n';
        }
        return ;
    }
} sol; // 注意不能定义在局部变量里面;
int n, m;
string s, x;
void solve() {
    cin >> n;
    for (int i = 0; i < n; i ++) {
        cin >> x;
        sol.insert(x, i);
        s = s + x;
        s.push_back('0');
    }
    sol.build();
    sol.solve(s, n);
}
```

PAM

```
const int N = 500010;
char str[N];
int len[N], n, m, fail[N], son[N][26], idx = 1;
int ans[N];
int get_fail(int x, int i) {
    // 跳到 奇根 或者找到了  插入str[i]的位置
    while (i - len[x] - 1 < 1 || str[i] != str[i - len[x] - 1]) {
        x = fail[x];
    }
    return x;
}

void insert() {
    int cur = 1; // 初始在奇根
    len[1] = -1;
    len[0] = 0;
```

```
    fail[0] = 1;
    fail[1] = 0;
    int n = strlen(str + 1);
    int last = 0;
    for (int i = 1; i <= n; i++) {
        str[i] = (str[i] - 97 + last) % 26 + 97;
        int u = get_fail(cur, i);
        // 确定str[i]插入在u的子树中
        if (!son[u][str[i] - 'a']) {
            // 没有这个儿子
            // 先计算fail, 再建儿子
            fail[++idx] = son[get_fail(fail[u], i)][str[i] - 'a'];
            son[u][str[i] - 'a'] = idx;
            len[idx] = len[u] + 2;
            ans[idx] = ans[fail[idx]] + 1;
        }
        cur = son[u][str[i] - 'a'];
        last = ans[cur];
        printf("%d ", last);
    }
    return ;
}

int main() {
    scanf("%s", str + 1);
    insert();
  return 0;
}
```