字符串

爱吃自动机

```
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        int top;
        int val;
        int d;
        std::array<int, ALPHABET> next;
        Node() : len{}, link{}, next{}, top{}, val {-1}, d{} {}
    };

    std::vector<Node> t;

    AhoCorasick() {
        init();
    }

    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }

    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }

    int add(const std::vector<int> &a) {
        int p = 1;
        for (auto x : a) {
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        apply (t[p].val);
        return p;
    }

    int add(const std::string &a, char offset = 'a') {
        std::vector<int> b(a.size());
        for (int i = 0; i < a.size(); i++) {
            b[i] = a[i] - offset;
        }
        return add(b);
    }
```

```
void work() {
    std::queue<int> q;
    q.push(1);

    while (!q.empty()) {
        int x = q.front();
        q.pop();

        t[x].top = t[link(x)].val >= 0 ? link(x) : top(link(x));

        for (int i = 0; i < ALPHABET; i++) {
            if (t[x].next[i] == 0) {
                t[x].next[i] = t[t[x].link].next[i];
            } else {
                t[t[x].next[i]].link = t[t[x].link].next[i];
                t[t[t[x].link].next[i]].d += 1;
                q.push(t[x].next[i]);
            }
        }
    }
}

int next(int p, int x) {
    return t[p].next[x];
}

int next(int p, char c, char offset = 'a') {
    return next(p, c - 'a');
}

int link(int p) {
    return t[p].link;
}

int len(int p) {
    return t[p].len;
}

int& val(int p) {
    return t[p].val;
}

int top (int p) {
    return t[p].top;
}

int size() {
    return t.size();
}

int& d ( int p ) {
    return t[p].d;
}

void apply (auto& val) {
```

```
        val = 0 ;
    }
};
```

字符串哈希

```
std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());

bool isprime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
            return false;
    return true;
}
int findPrime(int n) {
    while (!isprime(n))
        n++;
    return n;
}

template<int N>
struct StringHash {
    static array<int, N> mod;
    static array<int, N> base;
    vector<array<int, N>> p, h;
    StringHash() = default;
    StringHash(const string& s) {
        int n = s.size();
        p.resize(n);
        h.resize(n);
        fill(p[0].begin(), p[0].end(), 1);
        for (int i = 0; i < n; i++)
        for (int j = 0; j < N; j++) {
            p[i][j] = 1ll * (i == 0 ? 1ll : p[i - 1][j]) * base[j] % mod[j];
            h[i][j] = (1ll * (i == 0 ? 0ll : h[i - 1][j]) * base[j] + s[i]) %
mod[j];
        }
    }
    array<int, N> query(int l, int r) {
        assert(r >= l - 1);
        array<int, N> ans{};
        if (l > r) return {0, 0};
        for (int i = 0; i < N; i++) {
            ans[i] = (h[r][i] - 1ll * (l == 0 ? 0ll : h[l - 1][i]) * (r - l + 1
== 0 ? 1ll : p[r - l][i]) % mod[i] + mod[i]) % mod[i];
        }
        return ans;
    }
};

constexpr int HN = 2;
template<>
array<int, 2> StringHash<HN>::mod =
    {findPrime(rng() % 900000000 + 100000000),
```

```
    findPrime(rng() % 900000000 + 100000000)};
template<>
array<int, 2> StringHash<HN>::base {13331, 131};
using Hashing = StringHash<HN>;
```