ACM TEMPLATE

Fibonacci's Rabbit

Last build at May 10, 2019

Contents

1 数论 **2**
1.1 素数筛 2
1.2 唯一分解定理 2
1.3 欧拉函数 3
1.4 扩展欧几里得 3
1.5 逆元 3
1.6 快速幂取模 4
1.7 大整数取模 4
1.8 中国剩余定理 4

2 数学 **5**
2.1 矩阵 5
2.2 杜教筛 6
2.3 快速傅里叶变换 7
2.4 快速数论变换 8

3 字符串 **9**
3.1 字符串最小最大表示 9
3.2 kmp 算法 10
3.3 z 函数 10
3.4 manacher 11
3.5 字典树 11
3.6 ac 自动机 12
3.7 后缀数组 13
3.7.1 后缀数组-倍增法 13
3.7.2 后缀数组-dc3 15
3.7.3 后缀数组-快排 16
3.8 字符串分割 17
3.8.1 按字符分割 17
3.8.2 按字符串分割 17
3.8.3 按字符分割 (STL) 17

1 数论

1.1 素数筛

```
bool vis[maxn];
int primes[maxn];
int primes_len;

void sieve(int n) {
    int m = (int)sqrt(n + 0.5);
    memset(vis, 0, sizeof(vis));
    for (int i = 2; i <= m; i++)
        if (!vis[i]) {
            for (int j = i * i; j <= n; j += i) vis[j] = true;
        }
}

int gen_primes(int n) {
    sieve(n);
    int c = 0;
    for (int i = 2; i <= n; i++)
        if (!vis[i]) {
            primes[c++] = i;
        }
    return c;
}
```

1.2 唯一分解定理

```
const int maxn = 100;

// 求因子个数
int cnt(int n) {
    int s = 1;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            int a = 0;
            while (n % i == 0) {
                n /= i;
                a++;
            }
            s = s * (a + 1);
        }
    }
    if (n > 1) s = s * 2;
    return s;
}

// 求因子的和
int sum(int n) {
    int s = 1;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            int a = 1;
            while (n % i == 0) {
                n /= i;
                a *= i;
            }
            s = s * (a * i - 1) / (i - 1);
        }
    }
    if (n > 1) s = s * (1 + n);
    return s;
}

const int MOD = 1e9 + 7;

// 同时求cnt和sum
// sum取模
void solve(int n, ll& sum, ll& cnt) {
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            ll a = 1;
            ll t = 0;
            while (n % i == 0) {
                n /= i;
                a = a * i % MOD;
                t++;
            }
            cnt *= t + 1;
            sum = sum * ((a * i - 1) / (i - 1) % MOD) % MOD;
        }
    }
    if (n > 1) {
        sum = sum * (1 + n) % MOD;
        cnt *= 2;
    }
}

int primes[maxn];
int primes_len;

// 打素数表，只遍历素数
ll cnt(ll n) {
    ll s = 1;
    for (int i = 0; i < primes_len && primes[i] * primes[i] <= n; ++i) {
        if (n % primes[i] == 0) {
            ll a = 0;
            while (n % primes[i] == 0) {
                n /= primes[i];
                a++;
            }
            s = s * (a + 1);
        }
    }
    if (n > 1) s = s * 2;
    return s;
}
```

1.3 欧拉函数

```
int euler_phi(int n) {
    int m = (int)sqrt(n + 0.5);
    int ans = n;
    for (int i = 2; i <= m; i++)
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    if (n > 1) ans = ans / n * (n - 1);
}

int phi[maxn];

// 求1~n的欧拉函数值
void phi_table(int n) {
    for (int i = 2; i <= n; i++) phi[i] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        if (!phi[i]) {
            for (int j = i; j <= n; j += i) {
                if (!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
        }
}
```

1.4 扩展欧几里得

```
void ex_gcd(ll a, ll b, ll& d, ll& x, ll& y) {
    if (!b) {
        d = a;
        x = 1;
        y = 0;
    } else {
        ex_gcd(b, a % b, d, y, x);
        y -= x * (a / b);
    }
}
```

1.5 逆元

```
// 计算模n下a的逆。如果不存在逆，返回-1
ll inv(ll a, ll n) {
    ll d, x, y;
    ex_gcd(a, n, d, x, y);
    return d == 1 ? (x + n) % n : -1;
}
```

1.6 快速幂取模

```
ll fast_pow_mod(ll a, ll p, ll n) {
    if (p == 0) return 1;
    a = a % n;
    ll ans = pow_mod(a, p / 2, n) % n;
    ans = (ans * ans) % n;
    if (p % 2 == 1) ans = (ans * a) % n;
    return ans;
}

ll faster_pow_mod(ll a, ll b, ll c) {
    ll ans = 1;
    a = a % c;
    while (b != 0) {
        if (b & 1) ans = (ans * a) % c;
        b >>= 1;
        a = (a * a) % c;
    }
    return ans;
}
```

1.7 大整数取模

```
// b>0
ll big_mod(const char* a, ll b) {
    int st = 0;
    if (a[0] == '-') st = 1;
    int len = strlen(a);
    if (b < 0) b = -b;
    long long ans = 0;
    for (int i = st; i < len; i++) {
        ans = (ans * 10 + a[i] - '0') % b;
    }
    return ans;
}
```

1.8 中国剩余定理

```
// n个方程：x=a[i](mod m[i]) (0<=i<n)
ll china(int n, ll* a, ll* m) {
    ll M = 1, d, y, x = 0;
    for (int i = 0; i < n; i++) M *= m[i];
    for (int i = 0; i < n; i++) {
        ll w = M / m[i];
        ex_gcd(m[i], w, d, d, y);
        x = (x + y * w * a[i]) % M;
    }
    return (x + M) % M;
}

// unused
long long ex_crt(long long a[], long long n[], int num) {
    long long n1 = n[0], a1 = a[0], n2, a2, k1, k2, x0, gcd, c;
    for (int i = 1; i < num; i++) {
        n2 = n[i], a2 = a[i];
        c = a2 - a1;
        gcd = ex_gcd(n1, n2, k1, k2);  //解得：n1*k1+n2*k2=gcd(n1,n2)
        if (c % gcd) {
            flag = 1;
            return 0;  //无解
        }
        x0 = c / gcd * k1;  // n1*x0+n2*(c/gcd*k2)=c  PS:k1/gcd*c错误！
        t = n2 / gcd;
        x0 = (x0 % t + t) % t;  //求n1*x0+n2*y=c的x0的最小解
        a1 += n1 * x0;
        n1 = n2 / gcd * n1;
    }
    return a1;
}
```

2 数学

2.1 矩阵

```
const int maxn=100;
const int MOD=1e9+7;

struct Matrix {
    double a[3][3];
    Matrix inverse() {    //求三阶矩阵的行列式和逆矩阵
        double det = a[0][0] * a[1][1] * a[2][2] + a[0][1] * a[1][2] * a[2][0] + a[0][2] * a[1][0] * a[2][1] - a[0][2] * a[1][1] * a[2][0] -
                     a[0][1] * a[1][0] * a[2][2] - a[0][0] * a[1][2] * a[2][1];
        Matrix ret;
        ret.a[0][0] = a[1][1] * a[2][2] - a[1][2] * a[2][1];
        ret.a[1][0] = (a[1][0] * a[2][2] - a[1][2] * a[2][0]) * (-1);
        ret.a[2][0] = a[1][0] * a[2][1] - a[1][1] * a[2][0];
        ret.a[0][1] = (a[0][1] * a[2][2] - a[0][2] * a[2][1]) * (-1);
        ret.a[0][2] = a[0][1] * a[1][2] - a[0][2] * a[1][1];
        ret.a[1][1] = a[0][0] * a[2][2] - a[0][2] * a[2][0];
        ret.a[2][1] = (a[0][0] * a[2][1] - a[0][1] * a[2][0]) * (-1);
        ret.a[1][2] = (a[0][0] * a[1][2] - a[0][2] * a[1][0]) * (-1);
        ret.a[2][2] = a[0][0] * a[1][1] - a[0][1] * a[1][0];
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++) { ret.a[i][j] /= det; }
        }
        return ret;
    }
};

struct Matrix {
    ll a[maxn][maxn];
};

//若矩阵太大，返回值写在参数里
//中间结果用全局变量保存,最好不要重复使用
Matrix mul(const Matrix& l, const Matrix& r, int len) {
    Matrix c;
    for (int i = 0; i < len; i++) {
        for (int j = 0; j < len; j++) {
            c.a[i][j] = 0;
            for (int k = 0; k < len; k++) {
                c.a[i][j] = (c.a[i][j] + (l.a[i][k] * r.a[k][j]) % MOD) % MOD;
            }
        }
    }
    return c;
}

Matrix pow_mod(Matrix x, ll n, int len) {
    Matrix ans;
    memset(ans.a, 0, sizeof(ans.a));
    for (int i = 0; i < len; i++) ans.a[i][i] = 1;
    while (n) {
        if (n & 1) ans = mul(ans, x, len);
        x = mul(x, x, len);
        n >>= 1;
    }
    return ans;
}

Matrix add(const Matrix& l, const Matrix& r, int len) {
    Matrix c;
    for (int i = 0; i < len; i++) {
        for (int j = 0; j < len; j++) {
            c.a[i][j] = l.a[i][j] + r.a[i][j];
            c.a[i][j] %= MOD;
        }
    }
    return c;
}

//倍增法求解a^1 + a^2 + ... + a^n
Matrix ad(const Matrix& x, int p) {
    if (p == 1) return x;
    Matrix tmp = ad(x, p / 2);
    Matrix sum = add(tmp, mul(tmp, pow_mod(x, p / 2, N), N), N);
    if (p & 1) sum = add(sum, pow_mod(x, p, N), N);
    return sum;
}
```

2.2 杜教筛

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

#define rep(i, a, n) for (long long i = a; i < n; i++)
#define per(i, a, n) for (long long i = n - 1; i >= a; i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(), (x).end()
#define fi first
#define se second
#define SZ(x) ((long long)(x).size())
typedef vector<long long> VI;
typedef long long ll;
typedef pair<long long, long long> PII;
const ll mod = 1e9 + 7;

ll powmod(ll a, ll b) {
    ll res = 1;
    a %= mod;
    assert(b >= 0);
    for (; b; b >>= 1) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
    }
    return res;
}
// head

long long _, n;
namespace linear_seq {
    const long long N = 10010;
    ll res[N], base[N], _c[N], _md[N];

    vector<long long> Md;
    void mul(ll *a, ll *b, long long k) {
        rep(i, 0, k + k) _c[i] = 0;
        rep(i, 0, k) if (a[i]) rep(j, 0, k) _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
        for (long long i = k + k - 1; i >= k; i--)
            if (_c[i]) rep(j, 0, SZ(Md)) _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]]) %
                mod;
        rep(i, 0, k) a[i] = _c[i];
    }
    long long solve(ll n, VI a, VI b) {  // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
                                         //        printf(”%d\n”,SZ(b));
        ll ans = 0, pnt = 0;
        long long k = SZ(a);
        assert(SZ(a) == SZ(b));
        rep(i, 0, k) _md[k - 1 - i] = -a[i];
        _md[k] = 1;
        Md.clear();
        rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
        rep(i, 0, k) res[i] = base[i] = 0;
        res[0] = 1;
        while ((1ll << pnt) <= n) pnt++;
        for (long long p = pnt; p >= 0; p--) {
            mul(res, res, k);
            if ((n >> p) & 1) {
                for (long long i = k - 1; i >= 0; i--) res[i + 1] = res[i];
                res[0] = 0;
                rep(j, 0, SZ(Md)) res[Md[j]] = (res[Md[j]] - res[k] * _md[Md[j]]) % mod;
            }
        }
        rep(i, 0, k) ans = (ans + res[i] * b[i]) % mod;
        if (ans < 0) ans += mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1, 1), B(1, 1);
        long long L = 0, m = 1, b = 1;
        rep(n, 0, SZ(s)) {
            ll d = 0;
            rep(i, 0, L + 1) d = (d + (ll)C[i] * s[n - i]) % mod;
            if (d == 0)
                ++m;
            else if (2 * L <= n) {
                VI T = C;
                ll c = mod - d * powmod(b, mod - 2) % mod;
                while (SZ(C) < SZ(B) + m) C.pb(0);
                rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
                L = n + 1 - L;
                B = T;
                b = d;
                m = 1;
            } else {
                ll c = mod - d * powmod(b, mod - 2) % mod;
                while (SZ(C) < SZ(B) + m) C.pb(0);
                rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
                ++m;
            }
        }
        return C;
    }
    long long gao(VI a, ll n) {
        VI c = BM(a);
        c.erase(c.begin());
        rep(i, 0, SZ(c)) c[i] = (mod - c[i]) % mod;
        return solve(n, c, VI(a.begin(), a.begin() + SZ(c)));
    }
};  // namespace linear_seq

int main() {
    while (~scanf("%I64d", &n)) {
        printf("%I64d\n", linear_seq::gao(VI{1, 5, 11, 36, 95, 281, 781, 2245, 6336, 18061, 51205}, n - 1));
    }
}
```

2.3 快速傅里叶变换

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <iostream>
using namespace std;

const double PI = acos(-1.0);

struct Complex {
    double r, i;
    Complex(double _r = 0.0, double _i = 0.0) {
        r = _r;
        i = _i;
    }
    Complex operator+(const Complex &b) { return Complex(r + b.r, i + b.i); }
    Complex operator-(const Complex &b) { return Complex(r - b.r, i - b.i); }
    Complex operator*(const Complex &b) {
        return Complex(r * b.r - i * b.i, r * b.i + i * b.r);
    }
};
/*
 * 进行FFT和IFFT前的反转变换。
 * 位置i和 （i二进制反转后位置）互换
 * len必须去2的幂
 */
void change(Complex y[], int len) {
    int i, j, k;
    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        //交换互为小标反转的元素，i<j保证交换一次
        // i做正常的+1，j左反转类型的+1,始终保持i和j是反转的
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
        if (j < k) j += k;
    }
}
/*
 * 做FFT
 * len必须为2^k形式，
 * on==1时是DFT，on==–1时是IDFT
 */
void fft(Complex y[], int len, int on) {
    change(y, len);
    for (int h = 2; h <= len; h <<= 1) {
        Complex wn(cos(–on * 2 * PI / h), sin(–on * 2 * PI / h));
        for (int j = 0; j < len; j += h) {
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++) {
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t;
                y[k + h / 2] = u – t;
                w = w * wn;
            }
        }
    }
    if (on == –1)
        for (int i = 0; i < len; i++) y[i].r /= len;
}

const int MAXN = 200010;
Complex x1[MAXN], x2[MAXN];
char str1[MAXN / 2], str2[MAXN / 2];
int sum[MAXN];

int main() {
    while (scanf(”%s%s”, str1, str2) == 2) {
        int len1 = strlen(str1);
        int len2 = strlen(str2);
        int len = 1;
        while (len < len1 * 2 || len < len2 * 2) len <<= 1;
        for (int i = 0; i < len1; i++)
            x1[i] = Complex(str1[len1 – 1 – i] – ’0’, 0);
        for (int i = len1; i < len; i++) x1[i] = Complex(0, 0);
        for (int i = 0; i < len2; i++)
            x2[i] = Complex(str2[len2 – 1 – i] – ’0’, 0);
        for (int i = len2; i < len; i++) x2[i] = Complex(0, 0);
        //求DFT
        fft(x1, len, 1);
        fft(x2, len, 1);
        for (int i = 0; i < len; i++) x1[i] = x1[i] * x2[i];
        fft(x1, len, –1);
        for (int i = 0; i < len; i++) sum[i] = (int)(x1[i].r + 0.5);
        for (int i = 0; i < len; i++) {
            sum[i + 1] += sum[i] / 10;
            sum[i] %= 10;
        }
        len = len1 + len2 – 1;
        while (sum[len] <= 0 && len > 0) len––;
        for (int i = len; i >= 0; i––) printf(”%c”, sum[i] + ’0’);
        printf(”\n”);
    }
    return 0;
}
```

2.4 快速数论变换

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

inline int read() {
    int x = 0, f = 1;
    char ch = getchar();
    while (ch < ’0’ || ch > ’9’) {
        if (ch == ’–’) f = –1;
        ch = getchar();
    }
    while (ch <= ’9’ && ch >= ’0’) {
        x = 10 * x + ch – ’0’;
        ch = getchar();
    }
    return x * f;
}

void print(int x) {
    if (x < 0) putchar(’–’), x = –x;
    if (x >= 10) print(x / 10);
    putchar(x % 10 + ’0’);
}

const int N = 300100, P = 998244353;

inline int qpow(int x, int y) {
    int res(1);
    while (y) {
        if (y & 1) res = 1ll * res * x % P;
        x = 1ll * x * x % P;
        y >>= 1;
    }
    return res;
}

int r[N];

void ntt(int *x, int lim, int opt) {
    int i, j, k, m, gn, g, tmp;
    for (i = 0; i < lim; ++i)
        if (r[i] < i) swap(x[i], x[r[i]]);
    for (m = 2; m <= lim; m <<= 1) {
        k = m >> 1;
        gn = qpow(3, (P – 1) / m);
        for (i = 0; i < lim; i += m) {
            g = 1;
            for (j = 0; j < k; ++j, g = 1ll * g * gn % P) {
                tmp = 1ll * x[i + j + k] * g % P;
                x[i + j + k] = (x[i + j] – tmp + P) % P;
                x[i + j] = (x[i + j] + tmp) % P;
            }
        }
    }
    if (opt == –1) {
        reverse(x + 1, x + lim);
        int inv = qpow(lim, P – 2);
        for (i = 0; i < lim; ++i) x[i] = 1ll * x[i] * inv % P;
    }
}

int A[N], B[N], C[N];

char a[N], b[N];

int main() {
    while (~scanf(”%s%s”, a, b)) {
        memset(A, 0, sizeof(A));
        memset(B, 0, sizeof(B));
        int i, lim(1), n;
        n = strlen(a);
        for (i = 0; i < n; ++i) A[i] = a[n – i – 1] – ’0’;
        while (lim < (n << 1)) lim <<= 1;
        n = strlen(b);
        for (i = 0; i < n; ++i) B[i] = b[n – i – 1] – ’0’;
        while (lim < (n << 1)) lim <<= 1;
        for (i = 0; i < lim; ++i) r[i] = (i & 1) * (lim >> 1) + (r[i >> 1] >> 1);
        ntt(A, lim, 1);
        ntt(B, lim, 1);
        for (i = 0; i < lim; ++i) C[i] = 1ll * A[i] * B[i] % P;
        ntt(C, lim, –1);
        int len(0);
        for (i = 0; i < lim; ++i) {
            if (C[i] >= 10) len = i + 1, C[i + 1] += C[i] / 10, C[i] %= 10;
            if (C[i]) len = max(len, i);
        }
        while (C[len] >= 10) C[len + 1] += C[len] / 10, C[len] %= 10, len++;
        for (i = len; ~i; ––i) putchar(C[i] + ’0’);
        putchar(’\n’);
    }
    return 0;
}
```

3 字符串

3.1 字符串最小最大表示

```
#include <algorithm>
using namespace std;
```

```

// T = sec[k..n-1]+sec[0..k-1]
// k为返回值,n为sec的大小,T为sec的最小表示法
int get_min(const char* sec, int n) {
    int k = 0, i = 0, j = 1;
    while (k < n && i < n && j < n) {
        if (sec[(i + k) % n] == sec[(j + k) % n]) {
            k++;
        } else {
            sec[(i + k) % n] > sec[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
            if (i == j) i++;
            k = 0;
        }
    }
    i = min(i, j);
    return i;
}

int get_max(const char* sec, int n) {
    int k = 0, i = 0, j = 1;
    while (k < n && i < n && j < n) {
        if (sec[(i + k) % n] == sec[(j + k) % n]) {
            k++;
        } else {
            sec[(i + k) % n] < sec[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
            if (i == j) i++;
            k = 0;
        }
    }
    i = min(i, j);
    return i;
}
```

3.2 kmp算法

以i结尾的最小循环节: $i - f[i]$

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 10000 + 5;

int f[maxn];

void get_next(const char *P, int n) {
    f[0] = 0;
    f[1] = 0;  // 递推边界初值
    for (int i = 1; i < n; i++) {
        int j = f[i];
        while (j && P[i] != P[j]) j = f[j];
        f[i + 1] = (P[i] == P[j] ? j + 1 : 0);
    }
}
```

3.3 z函数

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 1000000 + 5;

int z[maxn];

// s 为待匹配的字符串指针
// n 为字符串长度
// z[i]是s和s+i的最大公共前缀长度。
void z_function(const char* s, int n) {
    fill_n(z, n, 0);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
}
```

3.4 manacher

回文匹配算法 (可用后缀数组代替, 但是比后缀数组简洁得多)

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 1000000;
int d1[maxn], d2[maxn];

// s 为字符串,也可以是const string&
// n 是字符串长度,即为s.length()
// d1为奇数回文长度(算上起点),总长度为d1[.]*2-1
// d2为偶数回文长度(算上起点),总长度为d2[.]*2
void Manacher(const char* s, int n) {
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if (i + k > r) {
            l = i - k;
            r = i + k;
        }
    }

    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if (i + k > r) {
            l = i - k - 1;
            r = i + k;
        }
    }
}

// 判断[l,r)是否回文
bool is_palindrome(int l, int r) {
    if (l == r) return true;
    if ((r - l) & 1) {
        return d1[l + (r - l) / 2] >= (r - l + 1) / 2;
    } else {
        return d2[l + (r - l) / 2] >= (r - l) / 2;
    }
}
```

3.5 字典树

```
#include <cstring>
#include <vector>
using namespace std;

const int wordnum = 100;
const int wordlen = 4000;
const int maxnode = wordnum * wordlen + 10;
const int sigma_size = 26;

// 字母表为全体小写字母的Trie
struct Trie {
    int ch[maxnode][sigma_size];
    int val[maxnode];
    int sz;  // 结点总数
    void clear() {
        sz = 1;
        memset(ch[0], 0, sizeof(ch[0]));
    }                                    // 初始时只有一个根结点
    int idx(char c) { return c - 'a'; }  // 字符c的编号

    // 插入字符串s，附加信息为v。注意v必须非0，因为0代表“本结点不是单词结点”
    void insert(const char *s, int v) {
        int u = 0, n = strlen(s);
        for (int i = 0; i < n; i++) {
            int c = idx(s[i]);
            if (!ch[u][c]) {  // 结点不存在
```

```
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;      // 中间结点的附加信息为0
                ch[u][c] = sz++;  // 新建结点
            }
            u = ch[u][c];  // 往下走
        }
        val[u] = v;  // 字符串的最后一个字符的附加信息为v
    }
};
```

3.6 ac 自动机

```
#include <cstring>
#include <queue>

using namespace std;

const int SIGMA_SIZE = 128;
const int WORD_SIZE = 55;
const int WORD_NUM = 1005;
const int MAXNODE = WORD_SIZE * WORD_NUM + 10;

struct AhoCorasickAutomata {
    int ch[MAXNODE][SIGMA_SIZE];
    int f[MAXNODE];     // fail函数
    int val[MAXNODE];   // 每个字符串的结尾结点都有一个非0的val
    int last[MAXNODE];  // 输出链表的下一个结点
    bool vis[MAXNODE];
    int cnt[WORD_NUM];
    int sz;

    void init() {
        sz = 1;
        memset(ch[0], 0, sizeof(ch[0]));
        memset(vis, 0, sizeof(vis));
        memset(cnt, 0, sizeof(cnt));
    }

    // 字符c的编号
    int idx(char c) const { return c; }

    // 插入字符串。v必须非0
    void insert(char* s, int v) {
        int u = 0, n = strlen(s);
        for (int i = 0; i < n; i++) {
            int c = idx(s[i]);
            if (!ch[u][c]) {
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        val[u] = v;
    }

    // 递归打印以结点j结尾的所有字符串
    void print(int j) {
        int ret = 0;
        if (j) {
            cnt[val[j]]++;
            print(last[j]);
        }
    }

    // 在T中找模板
    void find(const char* T) {
        int n = strlen(T);
        int j = 0;  // 当前结点编号，初始为根结点
        for (int i = 0; i < n; i++) {  // 文本串当前指针
            int c = idx(T[i]);
            while (j && !ch[j][c]) j = f[j];  // 顺着细边走，直到可以匹配
            j = ch[j][c];
            if (val[j])
                print(j);
            else if (last[j])
                print(last[j]);  // 找到了！
        }
    }

    // 计算fail函数
    void getFail() {
        queue<int> q;
        f[0] = 0;
        // 初始化队列
        for (int c = 0; c < SIGMA_SIZE; c++) {
            int u = ch[0][c];
            if (u) {
                f[u] = 0;
                q.push(u);
                last[u] = 0;
            }
        }
        // 按BFS顺序计算fail
        while (!q.empty()) {
            int r = q.front();
            q.pop();
            for (int c = 0; c < SIGMA_SIZE; c++) {
                int u = ch[r][c];
                if (!u) continue;
                q.push(u);
                int v = f[r];
                while (v && !ch[v][c]) v = f[v];
                f[u] = ch[v][c];
                last[u] = val[f[u]] ? f[u] : last[f[u]];
            }
        }
    }
};  // namespace AhoCorasickAutomata

AhoCorasickAutomata ac;
```

3.7 后缀数组

全字符串找循环节 要有长度为 i 的循环节，就要满足以下条件：

$$\begin{gathered} rank[0]{-}rank[i]=1 \\ height[rank[0]]=len{-}i \\ len\%i==0 \end{gathered}$$

找字串循环节最大重复次数 枚举长度 len，枚举起点 j，求 $lcp(j,j+len)$

$$\begin{gathered} ans=lcp/len+1 \\ k=j-(len-ans\%len) \\ if(k>0\&\&lcp(k,k+len)>=len)\{ans++;\} \end{gathered}$$

求 ans 最大值

复杂度

* 后缀数组倍增法（时间 O(nlongn)，空间 O(4n)）

* 后缀数组 dc3 法（时间 O(n)，空间 O(10n)）

* 后置数组快排（适用于最大值很大的情况，除了 sa 数组，其他暂未测试）

3.7.1 后缀数组-倍增法

```
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

namespace SuffixArray {
    using std::printf;

    const int maxn = 1e7 + 5;  // max(字符串长度，最大字符值加1)

    int s[maxn];                     // 原始字符数组（最后一个字符应必须是0，而前面的字符必须非0）
    int sa[maxn];                    // 后缀数组
    int rank[maxn];                  // 名次数组．rank[0]一定是n-1，即最后一个字符
    int height[maxn];                // height数组
    int t[maxn], t2[maxn], c[maxn];  // 辅助数组
    int n;                           // 字符个数（包括最后一个0字符）

    void init() { n = 0; }

    // m为最大字符值加1。调用之前需设置好s和n
    void build_sa(int m) {
        int i, *x = t, *y = t2;
        for (i = 0; i < m; i++) c[i] = 0;
        for (i = 0; i < n; i++) c[x[i] = s[i]]++;
        for (i = 1; i < m; i++) c[i] += c[i - 1];
        for (i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
        for (int k = 1; k <= n; k <<= 1) {
            int p = 0;
            for (i = n - k; i < n; i++) y[p++] = i;
            for (i = 0; i < n; i++)
                if (sa[i] >= k) y[p++] = sa[i] - k;
            for (i = 0; i < m; i++) c[i] = 0;
            for (i = 0; i < n; i++) c[x[y[i]]]++;
            for (i = 0; i < m; i++) c[i] += c[i - 1];
            for (i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
            swap(x, y);
            p = 1;
            x[sa[0]] = 0;
            for (i = 1; i < n; i++) x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1 : p++;
            if (p >= n) break;
            m = p;
        }
    }

    void build_height() {
        int i, k = 0;
        for (i = 0; i < n; i++) rank[sa[i]] = i;
        for (i = 0; i < n; i++) {
            if (k) k--;
            int j = sa[rank[i] - 1];
            while (s[i + k] == s[j + k]) k++;
            height[rank[i]] = k;
        }
    }
}  // namespace SuffixArray

// 编号辅助
namespace SuffixArray {
    int idx[maxn];

    // 给字符串加上一个字符，属于字符串i
    void add(int ch, int i) {
        idx[n] = i;
        s[n++] = ch;
    }
}  // namespace SuffixArray

// LCP 模板
namespace SuffixArray {
    using std::min;
    int dp[maxn][20];
    void initRMQ(int n) {
        for (int i = 1; i <= n; i++) dp[i][0] = height[i];
        for (int j = 1; (1 << j) <= n; j++)
            for (int i = 1; i + (1 << j) - 1 <= n; i++) dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
        return;
    }

    void initRMQ() { initRMQ(n - 1); }

    int lcp(int a, int b) {
        int ra = rank[a], rb = rank[b];
        if (ra > rb) swap(ra, rb);
        int k = 0;
        while ((1 << (k + 1)) <= rb - ra) k++;
        return min(dp[ra + 1][k], dp[rb - (1 << k) + 1][k]);
    }
}  // namespace SuffixArray

// 调试信息
namespace SuffixArray {
    using std::printf;
    void debug() {
        printf(”n:%d\n”, n);

        printf(”%8s”, ””);
        for (int i = 0; i < n; i++) {
            printf(”%4d”, i);
        }
        printf(”\n”);

        printf(”%8s”, ”s:”);
        for (int i = 0; i < n; i++) {
            printf(”%4d”, s[i]);
        }
        printf(”\n”);

        printf(”%8s”, ”sa:”);
        for (int i = 0; i < n; i++) {
            printf(”%4d”, sa[i]);
        }
        printf(”\n”);

        printf(”%8s”, ”rank:”);
        for (int i = 0; i < n; i++) {
            printf(”%4d”, rank[i]);
        }
        printf(”\n”);

        printf(”%8s”, ”height:”);
        for (int i = 0; i < n; i++) {
            printf(”%4d”, height[i]);
        }
        printf(”\n”);
    }
}  // namespace SuffixArray
```

3.7.2 后缀数组-dc3

```
#include <algorithm>

using namespace std;

/*
注意：
1.maxn开n的十倍大小;
2.dc3(r,sa,n+1,Max+1);r为待后缀处理的数组,sa为存储排名位置的数组,n+1和Max+1都和倍增一样
3.calheight(r,sa,n);和倍增一样
*/
// DC3 算法
namespace SuffixArray {
#define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
#define G(x) ((x) < tb ? (x)*3 + 1 : ((x)−tb) * 3 + 2)

    const int maxn = 1e7 + 5;

    int wa[maxn], wb[maxn], wv[maxn], ws[maxn];
    int s[maxn], sa[maxn];
    int rank[maxn], height[maxn];
    int n;

    void init() { n = 0; }

    int c0(int *r, int a, int b) { return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2]; }

    int c12(int k, int *r, int a, int b) {
        if (k == 2)
            return r[a] < r[b] || (r[a] == r[b] && c12(1, r, a + 1, b + 1));
        else
            return r[a] < r[b] || (r[a] == r[b] && wv[a + 1] < wv[b + 1]);
    }

    void sort(int *r, int *a, int *b, int n, int m) {
        int i;
        for (i = 0; i < n; i++) wv[i] = r[a[i]];
        for (i = 0; i < m; i++) ws[i] = 0;
        for (i = 0; i < n; i++) ws[wv[i]]++;
        for (i = 1; i < m; i++) ws[i] += ws[i − 1];
        for (i = n − 1; i >= 0; i−−) b[−−ws[wv[i]]] = a[i];
        return;
    }

    void dc3(int *r, int *sa, int n, int m) {
        int i, j, *rn = r + n, *san = sa + n, ta = 0, tb = (n + 1) / 3, tbc = 0, p;
        r[n] = r[n + 1] = 0;
        for (i = 0; i < n; i++)
            if (i % 3 != 0) wa[tbc++] = i;
        sort(r + 2, wa, wb, tbc, m);
        sort(r + 1, wb, wa, tbc, m);
        sort(r, wa, wb, tbc, m);
        for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++) rn[F(wb[i])] = c0(r, wb[i - 1], wb[i]) ? p - 1 : p++;
        if (p < tbc)
            dc3(rn, san, tbc, p);
        else
            for (i = 0; i < tbc; i++) san[rn[i]] = i;
        for (i = 0; i < tbc; i++)
            if (san[i] < tb) wb[ta++] = san[i] * 3;
        if (n % 3 == 1) wb[ta++] = n - 1;
        sort(r, wb, wa, ta, m);
        for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
        for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++) sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];
        for (; i < ta; p++) sa[p] = wa[i++];
        for (; j < tbc; p++) sa[p] = wb[j++];
        return;
    }

    void build_height(int n) {
        int i, j, k = 0;
        for (i = 1; i <= n; i++) rank[sa[i]] = i;
        for (i = 0; i < n; height[rank[i++]] = k)
            for (k ? k-- : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++)
                ;
        return;
    }

    void build_height() { build_height(n - 1); }

    void build_sa(int m) { dc3(s, sa, n, m); }

}  // namespace SuffixArray
```

3.7.3 后缀数组-快排

```
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;

namespace SuffixArray {
    using std::printf;

    const int maxn = 1e7 + 5;  // max(字符串长度，最大字符值加1)

    int s[maxn];                       // 原始字符数组（最后一个字符应必须是0，而前面的字符必须非0）
    int sa[maxn];                      // 后缀数组
    int t[maxn], rank[maxn], c[maxn];  // 辅助数组
    int n;                             // 字符个数（包括最后一个0字符）

    void init() { n = 0; }

    int k;
    bool compare_sa(int i, int j) {
        if (rank[i] != rank[j]) {
            return rank[i] < rank[j];
        } else {
            int ri = i + k < n ? rank[i + k] : -1;
            int rj = j + k < n ? rank[j + k] : -1;
            return ri < rj;
        }
    }

    void build_sa(int _) {
        for (int i = 0; i < n; i++) {
            sa[i] = i;
            rank[i] = i < n ? s[i] : -1;
        }
        for (k = 1; k < n; k <<= 1) {
            sort(sa, sa + n, compare_sa);
            t[sa[0]] = 0;
            for (int i = 1; i < n; i++) {
                t[sa[i]] = t[sa[i - 1]] + (compare_sa(sa[i - 1], sa[i]) ? 1 : 0);
            }
            for (int i = 0; i < n; i++) {
                rank[i] = t[i];
            }
        }
    }

    int height[maxn];  // height数组
    void build_height() {
        int i, k = 0;
        for (i = 0; i < n; i++) {
            if (k) k--;
            int j = sa[rank[i] - 1];
            while (s[i + k] == s[j + k]) k++;
            height[rank[i]] = k;
        }
    }
}  // namespace SuffixArray
```

3.8 字符串分割

3.8.1 按字符分割

```
#include <iostream>
#include <cstring>
#include <vector>
using namespace std;

// 字符串分割，分隔符为字符，可为多字符，前后不留空字符串
// *a,b*c,d, 按,*分割 -> {"a","b","c","d"}
// 注意：源字符串s将会被改变，请勿使用string.c_str()
// s源字符串 t传出结果 sep分隔符字符串(分隔符为每个单字符)
void split(char *s, vector<string> &v,const char *sep) {
    char *p = strtok(s, sep);
    while (p) {
        v.push_back(string(p));
        p = strtok(NULL, sep);
    }
}
```

3.8.2 按字符串分割

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// 字符串分割，分隔符为字符串，前后留空字符串
// cabcacac 按c分割 -> {"","ab","a","a",""}
// s源字符串 v传出结果 c分隔符字符串
void split(const string& s, vector<string>& v, const string& c) {
    string::size_type pos1, pos2;
    pos2 = s.find(c);
    pos1 = 0;
    while (string::npos != pos2) {
        v.push_back(s.substr(pos1, pos2 - pos1));

        pos1 = pos2 + c.size();
        pos2 = s.find(c, pos1);
    }
    if (pos1 <= s.length()) v.push_back(s.substr(pos1));
    // 如果要去除最后空串,用下方语句替代上一条
    // if (pos1 != s.length()) v.push_back(s.substr(pos1));
}
```

3.8.3 按字符分割 (STL)

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// 字符串分割，分隔符为字符，可为多字符，前后不留空字符串
// **a,b*c,d, 按,*分割 -> {"a","b","c","d"}
// strtok 的 实现
// s源字符串 t传出结果 sep分隔符字符串(分隔符为每个单字符)
void split(const string &s, vector<string> &v, const string &sep) {
    typedef string::size_type string_size;
    string_size i = 0;
    while (i != s.size()) {
        //找到字符串中首个不等于分隔符的字母；
        int flag = 0;
        while (i != s.size() && flag == 0) {
            flag = 1;
            for (string_size x = 0; x < sep.size(); ++x) {
                if (s[i] == sep[x]) {
                    ++i;
                    flag = 0;
                    break;
                }
            }
        }

        //找到又一个分隔符，将两个分隔符之间的字符串取出；
        flag = 0;
        string_size j = i;
        while (j != s.size() && flag == 0) {
            for (string_size x = 0; x < sep.size(); ++x) {
                if (s[j] == sep[x]) {
                    flag = 1;
                    break;
                }
            }
            if (flag == 0) ++j;
        }
        if (i != j) {
            v.push_back(s.substr(i, j – i));
            i = j;
        }
    }
}
```